A Joint Context-Aware Embedding for Trip Recommendations

Jiayuan He, Jianzhong Qi*, Kotagiri Ramamohanarao
School of Computing and Information Systems
The University of Melbourne, Australia
hjhe@student.unimelb.edu.au, {jianzhong.qi,kotagiri}@unimelb.edu.au

***Abstract*—Trip recommendation is an important location-based service that helps relieve users from the time and efforts for trip planning. It aims to recommend a sequence of places of interest (POIs) for a user to visit that maximizes the user's satisfaction. When adding a POI to a recommended trip, it is essential to understand the context of the recommendation, including the POI popularity, other POIs co-occurring in the trip, and the preferences of the user. These contextual factors are learned separately in existing studies, while in reality, they jointly impact on a user's choice of POI visits. In this study, we propose a POI embedding model to jointly learn the impact of these contextual factors. We call the learned POI embedding a context-aware POI embedding. To showcase the effectiveness of this embedding, we apply it to generate trip recommendations given a user and a time budget. We propose two trip recommendation algorithms based on our context-aware POI embedding. The first algorithm finds the exact optimal trip by transforming and solving the trip recommendation problem as an integer linear programming problem. To achieve a high computation efficiency, the second algorithm finds a heuristically optimal trip based on adaptive large neighborhood search. We perform extensive experiments on real datasets. The results show that our proposed algorithms consistently outperform state-of-the-art algorithms in trip recommendation quality, with an advantage of up to 43% in F_1-score.**

I. Introduction

Tourism is one of the most profitable and fast-growing economic sectors in the world. In 2017, the tourism industry contributed more than 8.27 trillion U.S. dollars to global economy. The massive scale of the tourism industry calls for more intelligent services to improve user experiences and reduce labor costs of the industry. Trip recommendation is one of such services. Trip recommendation aims to recommend a sequence of *places of interest* (POIs) for a user to vist to maximize the user's satisfaction. Such a service benefits users by relieving them from the time and efforts for trip planning, which in return further boosts the tourism industry.

*Primary contact

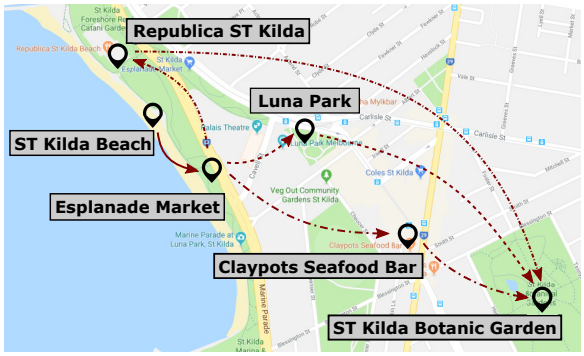


Fig. 1: Impact of co-occurring POIs

Most existing studies on trip recommendations consider POI popularities or user preferences towards the POIs when making recommendations [4], [14]. Several recent studies [3], [20] consider the last POI visit when recommending the next POI to visit. These studies do not model the following two characteristics that we observe from real-world user trips (detailed in Section III). (i) A POI to be recommended is impacted not only by the last POI visit but also all other POIs co-occurring in the same trip. For example, in Fig. 1, a user has just visited "ST Kilda Beach" and "Esplanade Market". She may be tired after the long walk along the beach and the market. Thus, compared with "Luna Park" which is a theme park nearby, the user may prefer a restaurant (e.g., "Republica ST Kilda" or "Claypots Seafood Bar") to get some food and rest. The user plans to visit "Botanic Garden" later on. Thus, she decides to visit "Claypots Seafood Bar" since it is on the way from the beach to the garden. Here, the visit to "Claypots Seafood Bar" is impacted by the visits of not only "Esplanade Market" but also "ST Kilda Beach" and "Botanic Garden." (ii) POI popularities, user preferences, and co-occurring POIs together impact the POIs to be recommended in a trip. In the example above, there can be many restaurants on the way to "Botanic Garden." The choice of "Claypots Seafood Bar" can be impacted by not only "Botanic Garden" but also the

fact that the user is a seafood lover and that "Claypots Seafood Bar" is highly rated. Most existing models [16], [20] learn the impact of each factor separately and simply combine them by linear summation, which may not reflect the joint impact accurately.

In this study, we model the two observations above with a *context-aware POI embedding model* to jointly learn the impact of POI popularities, user preferences, and co-occurring POIs. We start with modeling the impact of co-occurring POIs. Existing studies model the impact of the last POI visit with a first-order Markov model [3], [12], [20]. Such a model requires a large volume of data to learn the impact between every pair of POIs. However, real-world POI visits are sparse and highly skewed. There are not enough observations to learn a Markov model accurately. We address the above data sparsity limitation by embedding the POIs into a space where POIs that co-occur frequently are close to each other. This is done based on our observation that a trip can be seen as a "sentence" where each POI visit is a "word." The occurrence of a POI in a trip is determined by all the co-occurring words (POIs) in the same sentence (trip). This enables us to learn a POI embedding similar to the Word2Vec model [18] that embeds words with a similar context close to each other.

To further incorporate the impact of user preferences into the embedding, we project users into the same latent space of the POIs, where the preferences of each user is modeled by the proximity between the user and the POIs. We also extend the embedding of each POI by adding a dimension (a bias term) to represent the POI popularity. We jointly learn the embeddings of users and POIs via *Bayesian Pairwise Ranking* [21].

To showcase the effectiveness of our proposed context-aware POI embedding, we apply it to a trip recommendation problem name TripRec where a user and her time budget is given. We propose two algorithms for the problem. The first algorithm, *C-ILP*, models the trip recommendation problem as an integer linear programming problem. It solves the problem with an integer linear programming technique [1]. C-ILP offers exact optimal trips, but it may be less efficient for large time budgets. To achieve a higher efficiency, we further propose a heuristic algorithm named *C-ALNS* based on the *adaptive large neighborhood search* (ALNS) technique [22]. C-ALNS starts with a set of initial trips and optimizes them iteratively by replacing POIs in the trips with unvisited POIs that do not break the user time budget. We use the POI-user proximity computed by our context-aware POI embedding to guide the optimization process of C-ALNS. This leads to high-quality trips with low computational costs.

This paper makes the following contributions:

1) We analyze real-world POI check-in data to show the impact of co-occurring POIs and the joint impact of contextual factors on users' POI visits.
2) We propose a novel model to learn the impact of co-occurring POIs. We further propose a context-aware POI embedding model to jointly learn the impact of POI popularities, co-occurring POIs, and user preferences on POI visits.
3) We propose two algorithms C-ILP and C-ALNS to generate trip recommendations based on our context-aware POI embedding model. C-ILP transforms trip recommendation to an integer linear programming problem and provides exact optimal trips. C-ALNS adapts the approximate large neighborhood search technique and provides heuristically optimal trips close to the exact optimal trips with a high efficiency.
4) We conduct extensive experiments on real datasets. The results show that our proposed algorithms outperform state-of-the-art algorithms consistently in the quality of the trips recommended as measured by the F_1-score. Further, our heuristic algorithm C-ALNS produces trip recommendations that differ in accuracy from those of C-ILP by only 0.2% while reducing the running time by 99.4%.

The rest of this paper is structured as follows. Section II reviews related studies. Section III presents an empirical analysis on real-world check-in datasets to show the factors impacting POI visits. Section IV formulates the problem studied. Section V and VI detail our POI embedding model, and trip recommendation algorithms based on the model. Section VII reports experiment results. Section VIII concludes the paper.

II. Related work

We compute POI embeddings to enable predicting POI sequences (trips) to be recommended to users. We review inference models for predicting a POI to be recommended in Section II-A. We review trip generation algorithms based on these models in Section II-B.

A. POI Inference Model

Most existing inference models for trip recommendations assume independence between recommended POIs, i.e., the probability of recommending a POI is independent from that of any other POI [2], [7], [14], [26], [27]. Such an independence assumption loses the POI co-occurrence relationships. We do not discuss studies based on this assumption further.

Three existing models incorporate POI dependency in their inferences. Kurashima et al. [12] adopt Markov

model that represents the dependence of a POI l_{i+1} on its preceding POI l_i in a trip by the transit probability from l_i to l_{i+1}. Rakesh et al. [20] also assume that each POI visit depends on its preceding POI. They unify such dependency with other factors (e.g., POI popularities) into a latent topic model, where the topical distribution of the next POI visit depends on its preceding POI. These two studies suffer from the data sparsity problem since the real-world check-in dataset is too sparse to learn the pairwise POI transit probabilities accurately. Chen et al. [3] overcome the data sparsity problem by factorizing the transit probability between two POIs as the product of the pairwise transit probabilities w.r.t. five pre-defined features: POI category, neighborhood (geographical POI cluster membership), popularity (number of distinct visitors), visit counts (total number of check-ins), and average visit duration. These five features can be considered as an embedding of a POI. However, such an embedding is manually designed and may not reflect the salient features of a POI.

POI dependency is also considered in POI recommendations [5], [6], which aim to recommend an individual POI instead of a POI sequence. Such studies do not consider the dependence among POIs in a trip. Liu et al. [16] also use a latent space for POI recommendations. They first learn the latent vectors of POIs to capture the dependence between POIs. Then, they fix the POI vectors and learn the latent vectors of users from the user-POI interactions. These studies differ from ours in three aspects: (i) Their models learn the impact of POIs and the impact of user preferences independently, while our model learns the impact of the two factors jointly, which better captures the data characteristics and leads to an improved trip recommendation quality as shown in our experiments. (ii) Their models focus on user preferences and do not consider the impact of POI popularities, while ours considers both factors. (iii) These studies do not consider constraints such as time budgets while ours does.

B. Trip Generation

Trip recommendation aims to generate a trip, i.e., a sequence of POIs, that meets user constraints and maximizes user satisfaction. Different user constraints and user satisfaction formulation lead to different trip generation algorithms. Due to the page limit, we only discuss the studies that consider POI dependencies. Hsieh et al. [11] and Rakesh et al. [20] assume a given starting POI l_s, a given time budget t_q, and a given time buffer b. They generate a trip recommendation by starting from l_s and progressively adding more POIs to the trip until the trip time reaches $t_q - b$. They repeatedly add the unvisited POI that has the highest transit probability from the last POI in the trip. As discussed earlier, their transit probabilities depend only on the last POI but not any other co-occurring POIs. Chen et al. [3] assume given starting and ending POIs and a time budget. They formulate trip recommendation as an orienteering problem in a directed graph, where every vertex represents a POI and the weight of an edge represents the transit probability from its source vertex to its end vertex. Our trip recommendation problem shares similar settings. However, Chen et al.'s algorithm does not apply to our problem as we not only consider the transit probabilities between adjacent POIs but also the impact of all POIs in a trip.

III. Observations on POI Check-ins

We start with an empirical study on real-world POI check-in data to observe users' check-in patterns. We aim to answer the following three questions: (1) Are users' check-ins at a POI impacted by other POIs co-occurring in the same trip? (2) Are users' check-ins at a POI impacted by (other users') historical check-ins at the POI, i.e., the popularity of the POI? (3) Are the impact of co-occurring POIs and the impact of POI popularity independent from each other?

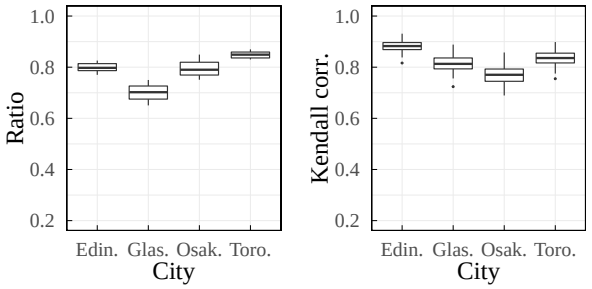
We analyze four datasets [3], [14], extracted from the Yahoo!Flickr Creative Commons 100M dataset [24]. They contain check-ins in the cities of Edinburgh, Glasgow, Osaka, and Toronto respectively. Table I summarizes the statistics of the four datasets.

TABLE I: Dataset Statistics

Dataset	#users	#POI visits	#trips	POIs/trip
Edinburgh	82,060	33,944	5,028	6.75
Glasgow	29,019	11,434	2,227	5.13
Osaka	392,420	7,747	1,115	6.95
Toronto	157,505	39,419	6,057	6.51

Impact of co-occurring POIs. Directly investigating the impact of co-occurring POIs on one POI is non-trivial due to the large number of possible permutations of co-occurring POIs. Thus, we investigate this impact by its symmetric form: the impact of one POI on its co-occurring POIs. Thus, for each dataset, we perform a hypothesis test as follows. We randomly sample 50% of the trips. From the sampled dataset, we compute an $|\mathcal{L}|$-dimensional distribution named the *global occurrence distribution*, where $\mathcal{L}$ is the set of all POIs in the dataset, and dimension i represents the normalized frequency of POI l_i occurring in the sampled trips. Then for each POI l, we compute an $(|\mathcal{L}|-1)$-dimensional distribution named the *conditional occurrence distribution*, where

dimension i represents the normalized frequency of POI l_i occurring together with l in the same trip. For each POI l, we perform a *chi-square two sample test* between its conditional occurrence distribution and the global occurrence distribution, where the null hypothesis is that "the conditional occurrence distribution conforms the same underlying distribution with the global occurrence distribution" and the significance level is 0.05. Note that we remove the dimension corresponding to the occurrence frequency of l in the global occurrence distribution. If the hypothesis is rejected, we say that POI l is an *influential POI* (has influence on its co-occurring POIs). We generate 100 sample datasets and report the ratio of influential POIs among all POIs in each run. Figure 2a shows the result, where the rectangles denote the 25 percentile, median, and 75 percentile of the result ratios. On average, influential POIs take up as least 70.3% (Glasgow) and up to 87.5% (Toronto) of all POIs, which confirms the impact of co-occurring POIs.



(a) Impacted POI pairs (b) Impacted users

Fig. 2: Observations on POI check-ins

Impact of POI popularity. For each city, we randomly split its dataset into two subsets, each of which consists of the POI visits of half of the users. We use one of the subsets as a *historical dataset* and the other subset as a *testing dataset*. Let $|\mathcal{L}|$ be the number of POIs. We compute two $|\mathcal{L}|$-dimensional distributions, named *historical occurrence distribution* and *testing occurrence distribution*, from the two datasets respectively. Dimension i of the two distributions represents the occurring frequency of POI l_i in historical dataset and testing dataset, respectively. We compute the Kendall-rank correlation of the two distributions. A higher correlation value indicates that the POI-ranks in the two datasets are more similar, which means that more popular POIs in the historical dataset are more likely to be visited in the testing dataset. We report the Kendall-rank correlation value of the two distributions in 100 runs of the procedure above (with random selection for dataset splitting). As Fig. 2b shows, all datasets have Kendall correlations beyond 0.7, which demonstrates the importance of POI popularity.

TABLE II: Frequently Used Symbols

Symbol	Description
$\mathcal{R}$	a set of check-in records
$\mathcal{L}$	a set of POIs
$\mathcal{U}$	a set of users
l	a POI
u	a user
s^u	a trip of user u
$\mathbf{l}$	the latent vector of l
$\mathbf{u}$	the latent vector of u
$\mathbf{c}(\mathbf{l})$	the latent vector of the co-occurring POIs of l

Joint impact of co-occurring POIs and POI popularity. The empirical study above confirms the impact of co-occurring POIs and the impact of POI popularity. A side observation when comparing Fig. 2a and Fig. 2b is that these factors have a joint impact rather than independent one. In general, for the cities where co-occurring POIs have a greater impact, POI popularity has a less impact (e.g., Edinburgh), and vice versa (e.g., Osaka). This brings a challenge on designing a model that can learn the impact of the factors jointly and can adapt to the different levels of joint impact across different datasets.

IV. Problem formulation

We aim to learn a context-aware POI embedding such that POIs co-occurring more frequently are closer in the embedded space. We map POIs and users to this embedded space and make trip recommendations based on their closeness in the embedded space.

To learn such an embedding, we use a POI check-in dataset $\mathcal{R}$ (e.g., the datasets summarized in Table I). Each check-in record $r \in \mathcal{R}$ is a 3-tuple $\langle u, l, t\rangle$, where u denotes the check-in user, l denotes the POI, and t denotes the check-in time.

POI visit. Let $\mathcal{U}$ be the set of all users and $\mathcal{L}$ be the set of all POIs in the check-in records in $\mathcal{R}$. We merge a user u's consecutive check-ins at the same POI l into a *POI visit* $v^u = \langle u, l, t_a, t_d\rangle$, where t_a and t_d represent the times of the first and the last (consecutive) check-ins at l by u. With a slight abuse of terminology, we use a POI visit v^u and the corresponding POI l interchangeably as long as the context is clear.

User profile and preferences. POI visits of user u within a certain time period (e.g., a day) form a *historical trip* of u, denoted as $s^u = \langle v^u_1, v^u_2, \ldots, v^u_{|s^u|}\rangle$. All historical trips of user u form the *profile* of u, denoted as $\mathcal{S}^u = \{s^u_1, s^u_2, \ldots, s^u_{|\mathcal{S}^u|}\}$. We learn the POI and user embeddings from the set $\mathcal{S}$ of all historical trips of all

users in $\mathcal{U}$, i.e., $\mathcal{S} = \mathcal{S}^{u_1} \cup \mathcal{S}^{u_2} \cup \ldots \cup \mathcal{S}^{u_{|\mathcal{U}|}}$, and use the learned user embedding to represent user preferences.

TripRec query. To showcase the effectiveness of our POI embedding, we apply it to a trip recommendation problem [14]. Given a query user u_q, this problem aims to recommend a trip tr formed by an ordered sequence of POIs: $tr = \langle l_1, l_2, \ldots, l_{|tr|} \rangle$, such that the value of a *user satisfaction function* is maximized. We propose a novel user satisfaction function denoted by $S(u_q, tr)$ which is detailed in Section VI. Intuitively, each recommended POI makes a contribution to $S(u_q, tr)$, and the contribution is larger when the POI suits u_q's preference better. We use a time budget t_q to cap the number of POIs in tr. The *time cost* of tr, denoted by $tc(tr)$, must not exceed t_q. The time cost $tc(tr)$ is the sum of the *visiting time* at every POI $l_i \in tr$, denoted as $tc_v(l_i)$, and the *transit time* between every two consecutive POIs $l_i, l_{i+1} \in tr$, denoted as $tc_t(l_i, l_{i+1})$:

$$tc(tr) = \sum_{i=1}^{|tr|} tc_v(l_i) + \sum_{i=1}^{|tr|-1} tc_t(l_i, l_{i+1}) \quad (1)$$

We derive the visiting time $tc_v(l_i)$ as the average time of POI visits at l_i. The transit time $tc_t(l_i, l_{i+1})$ depends on the transportation mode (e.g., by walk or car), which is orthogonal to our study. Without loss of generality and following previous studies [8], [14], [26], we assume transit by walk and derive $tc_t(l_i, l_{i+1})$ as the road network shortest path distance between l_i and l_{i+1} divided by an average walking speed of 4 km/h. Other transit time models can also be used. Following [3], [14], [26], we also require l_1 and $l_{|tr|}$ to be at a given starting POI l_s and a given ending POI l_e. We call such a trip recommendation problem the *TripRec query*:

Definition 1 (TripRec Query): A TripRec query q is represented by a 4-tuple $q = \langle u_q, t_q, l_s, l_e \rangle$. Given a query user u_q, a query time budget t_q, a starting POI l_s, and an ending POI l_e, the TripRec query finds a trip $tr = \langle l_1, l_2, ..., l_{|tr|} \rangle$ that maximizes $S(u_q, tr)$ and satisfies: (i) $tc(tr) \leqslant t_q$, (ii) $l_1 = l_s$, and (iii) $l_{|r|} = l_e$.

V. Learning a Context-Aware POI Embedding

Consider a POI l_i, a user u, and a historical trip s of u that contains l_i. The popularity of l_i, the user u, and the other POIs co-occurring in s together form a *context* of l_i. Our POI embedding is computed from such contexts, and hence is named a *context-aware POI embedding*. We first discuss how to learn a POI embedding such that POIs co-occurring more frequently are closer in the embedded space in Section V-A. We further incorporate user preferences and POI popularities into the embedding in Sections V-B and V-C. We present an algorithm for model parameter learning in Section V-D,

A. Learning POI Co-Occurrences

Given a POI l_i, we call another POI l_j a *co-occurring POI* of l_i, if l_j appears in the same trip as l_i. The conditional probability $p(l_i|l_j)$, i.e., the probability of a trip containing l_i given that l_j is in the trip, models the *co-occurrence relationship* of l_i over l_j.

The Markov model is a straightforward solution to learn $p(l_i|l_j)$. This model assumes that the transit probability of each POI pair is independent from any other POIs, leading to a total $|\mathcal{L}|^2$ probabilities to be learned. However, the real-world POI check-in datasets are too sparse to learn these probabilities accurately. Some POI transistions may be rare or not observed at all. To overcome the data sparsity problem and capture the co-occurrence relationships POIs, we propose a model to learn $p(l_i|c(l_i))$ instead of $p(l_i|l_j)$, where $c(l_i)$ represents the set of co-occurring POIs of l_i. Our model is inspired by the *Word2vec* model [18]. The Word2vec model embeds words into a vector space where each word is placed in close proximity with its *context words*. Given an occurrence of word w in a large text corpus, each word that occurs within a pre-defined distance to w is regarded as a context word of w. This pre-defined distance forms a *context window* around a word. In our problem, we can view a POI as a "word", a historical trip as a "context window", the historical trips of a user as a "document", and all historical trips of all the users as a "text corpus". Then, we can learn a POI embedding based on the probability distribution of the co-occurring POIs.

Specifically, we use the architecture of *continuous bag-of-words* (CBOW) [17], which predicts the *target word* given its *context*, to compute the POI embedding. The computation works as follows. Given a POI $l_i \in \mathcal{L}$, we map l_i into a latent d-dimensional real space $\mathbb{R}^d$ where d is a system parameter, $d \ll |\mathcal{L}|$. The mapped POI, i.e., the POI embedding, is a d-dimensional vector $\mathbf{l_i}$. When computing the embeddings, we treat each historical trip as a context window: given l_i in a historical trip s, we treat l_i as the target POI and all other POIs in s as its co-occurring POIs $c(l_i|s)$, i.e., $c(l_i|s) = \{l | l \in s \setminus \{l_i\}\}$. In the rest of the paper, we abbreviate $c(l_i|s)$ as $c(l_i)$ as long as the context is clear.

Let $sim(l_i, l_j)$ be the *co-occurrence similarity* between two POIs l_i and l_j. We compute $sim(l_i, l_j)$ as the dot product of the embeddings of l_i and l_j:

$$sim(l_i, l_j) = \mathbf{l_i} \cdot \mathbf{l_j} \quad (2)$$

Similarly, the co-occurrence similarity between a POI l_i and its set of co-occurring POIs $c(l_i)$, denoted as $sim(l_i, c(l_i))$, is computed as:

$$sim(l_i, c(l_i)) = \mathbf{l_i} \cdot \mathbf{c(l_i)} \quad (3)$$

Here, $\mathbf{c}(\mathbf{l_i})$ is computed as an aggregate vector of the embeddings of the POIs in $c(l_i)$. We follow Henry et al. [10] and aggregate the embeddings by summing them up in each dimension independently. Other aggregate functions (e.g., [25]) can also be used. Then, the probability of observing l_i given $c(l_i)$ is derived by applying the softmax function on the co-occurrence similarity $sim(l_i, c(l_i))$:

$$p(l_i|c(l_i)) = \frac{e^{sim(l_i,c(l_i))}}{Z(\mathbf{c}(\mathbf{l_i}))} = \frac{e^{\mathbf{l_i}\cdot\mathbf{c}(\mathbf{l_i})}}{Z(\mathbf{c}(\mathbf{l_i}))} \tag{4}$$

Here, $Z(\mathbf{c}(\mathbf{l_i})) = \sum_{l\in\mathcal{L}} e^{\mathbf{l}\cdot\mathbf{c}(\mathbf{l_i})}$ is a normalization term.

B. Incorporating User Preferences

Next, we incorporate user preferences into our model. We model a user's preferences towards the POIs as her "co-occurrence" with the POIs, i.e., a user u_j is also projected to a d-dimensional embedding space where she is closer to the POIs that she is more likely to visit (i.e., "co-occur"). Specifically, the co-occurrence similarity between a POI l_i and a user u_j is computed as:

$$sim(l_i, u_j) = \mathbf{l_i} \cdot \mathbf{u_j} \tag{5}$$

Thus, the preference of u_j over l_i can be seen as the probability $p(l_i|u_j)$ of observing l_i given u_j in the space. After applying the softmax function over $sim(l_i, u_j)$, $p(l_i|u_j)$ can be computed as:

$$p(l_i|u_j) = \frac{e^{\mathbf{l_i}\cdot\mathbf{u_j}}}{Z(\mathbf{u_j})} \tag{6}$$

Here, $Z(\mathbf{u_j}) = \sum_{l\in\mathcal{L}} e^{\mathbf{l}\cdot\mathbf{u_j}}$ is a normalization term.

To integrate user preferences with POI co-occurrence relationships, we unify the POI embedding space and the user embedding space into a single embedding space. In this unified embedding space, the POI-POI proximity reflects POI co-occurrence relationships and the user-POI proximity reflects user preferences. Intuitively, we treat each user u_j as a "pseudo-POI". If user u_j visits POI l_i, then u_j (a pseudo-POI) serves as a co-occurring POI of l_i. Thus, the joint impact of user preferences and POI co-occurrences can be modeled by combining the pseudo POI and the actual co-occurring POIs. Given a set of co-occurring POIs $c(l_i)$ and a user u_j, the probability of observing l_i can be written as:

$$p(l_i|c(l_i), u_j) = \frac{e^{\mathbf{l_i}\cdot(\mathbf{u_j}+\mathbf{c}(\mathbf{l_i}))}}{Z(\mathbf{u_j}+\mathbf{c}(\mathbf{l_i}))} \tag{7}$$

Here, vectors $\mathbf{u_j}$ and $\mathbf{c}(\mathbf{l_i})$ are summed up in each dimension, while $Z(\mathbf{u_j}+\mathbf{c}(\mathbf{l_i})) = \sum_{l\in\mathcal{L}} e^{\mathbf{l}\cdot(\mathbf{u_j}+\mathbf{c}(\mathbf{l_i}))}$ is a normalization term.

C. Incorporating POI Popularity

We further derive $p(l_i)$ which represents the popularity of l_i. Existing studies represent the popularity of a POI as the normalized visit frequency at the POI [8], [14], [15]. This simple approach relies on a strong assumption that POI popularity is linearly proportional to the number of POI visits, which may not hold since popularity may not be the only reason for visiting a POI.

Instead of counting POI visit frequency, we propose to learn the POI popularity jointly with the impact of co-occurring POIs and user preferences. Specifically, we add a dimension to the unified POI and user embedding space, i.e., we embed the POIs to an $\mathbb{R}^{d+1}$ space. This extra dimension represents the latent popularity of a POI, and the embedding learned for this space is our *context-aware POI embedding*.

For a POI l_i, its embedding now becomes $\mathbf{l_i} \oplus l_i.p$ where $\oplus$ is a concatenation operator and $l_i.p$ is the latent popularity. The probability $p(l_i)$ is computed by applying the softmax function over $l_i.p$:

$$p(l_i) = \frac{e^{l_i.p}}{\sum_{l\in\mathcal{L}} e^{l.p}} \tag{8}$$

Integrating with the POI contextual relationships and user preferences, the final probability of oberving l_i given u_j and $c(l_i)$ can be represented as:

$$p(l_i|c(l_i), u_j) = \frac{e^{\mathbf{l_i}\cdot(\mathbf{u_j}+\mathbf{c}(\mathbf{l_i}))+l_i.p}}{Z(\mathbf{u_j}+\mathbf{c}(\mathbf{l_i})+l_i.p)} \tag{9}$$

Here, $Z(\mathbf{u_j}+\mathbf{c}(\mathbf{l_i})+l_i.p) = \sum_{l\in\mathcal{L}} e^{\mathbf{l}\cdot(\mathbf{u_j}+\mathbf{c}(\mathbf{l_i}))+l.p}$ is a normalization term.

D. Parameter Learning

We adopt the *Bayesian Pairwise Ranking* (BPR) approach [21] to learn the embeddings of POIs and users. The learning process aims to maximize the posterior of the observations:

$$\Theta = \underset{\Theta}{\text{argmax}} \prod_{u\in\mathcal{U}} s^u \prod_{\mathcal{H}^u} \prod_{l\in s^u} \prod_{l'\notin s^u} P(>_{u,c(l)}|\Theta)p(\Theta) \tag{10}$$

Here, Θ represents the system parameters to be learned (i.e., user and POI vectors) and $P(>_{u,c(l)}|\Theta)$ represents the pairwise margin between the probabilities of observing l and observing l' given u and $c(l)$. Maximizing the above objective function equals to maximizing its log-likelihood function. In addition, we add a regularization term to avoid overfitting. Thus, the above equation can be rewritten as follows:

$$\Theta = \underset{\Theta}{\text{argmax}} \sum_{u\in\mathcal{U}} s^u \sum_{\mathcal{H}^u} \sum_{l\in s^u} \sum_{l'\notin s^u} \log\sigma\Big(\mathbf{l}\cdot\mathbf{c}(\mathbf{l}) + \mathbf{l}\cdot\mathbf{u} + l.p - \mathbf{l'}\cdot\mathbf{c}(\mathbf{l}) - \mathbf{l'}\cdot\mathbf{u} - l'.p\Big) - \lambda||\Theta||^2 \tag{11}$$

Here, $\sigma(\cdot)$ is the sigmoid function and $\sigma(z) = \frac{1}{1+e^{-z}}$. We use stochastic gradient descent (SGD) to solve the optimization problem. Given a trip s^u of user u, we

obtain $|s^u|$ observations in the form of $\langle u, s^u, l, c(l)\rangle$, where $l \in s^u$. For each observation, let l' be a POI that does not appear in s^u. Then, we update Θ along the ascending gradient direction:

$$\Theta \leftarrow \Theta + \eta \frac{\partial}{\partial \Theta}(\log \sigma(z) - \lambda||\Theta||^2) \tag{12}$$

Here, η represents the learning rate and $z = \mathbf{l} \cdot \mathbf{c}(\mathbf{l}) + \mathbf{l} \cdot \mathbf{u} + l.p - \mathbf{l}' \cdot \mathbf{c}(\mathbf{l}) - \mathbf{l}' \cdot \mathbf{u} - l'.p$ represents the distance between the observed POI and a non-observed POI l'.

VI. Trip Recommendation

To showcase the capability of our context-aware POI embedding to capture the latent POI features, we apply it to the TripRec query as defined in Section IV. Let $\mathcal{T}$ be the set of *feasible trips* that satisfy the query hard constraints. The problem then becomes selecting the trip $tr \in \mathcal{T}$ that is most preferred by u_q. The strategy that guides trip selection plays a critical role in recommendation quality. We propose the *context-aware trip quality* (CTQ) score to guide trip selection. Then, we reduce TripRec to an optimization problem of finding the feasible trip with the highest CTQ score.

Context-aware trip quality score. Given a trip tr, we derive its CTQ score, denoted as $S(u_q, tr)$, as a joint score of the closeness of tr w.r.t. two types of contexts: (1) a *fixed context* that is defined by the query hard constraints; and (2) a *dynamic context* that is defined by the recommended POIs in tr.

To compute the closeness between tr and the fixed context of query q, we derive the latent representation $\mathbf{q}$ of q as an aggregation (e.g., summation) of the vectors $\mathbf{u_q}$, $\mathbf{l_s}$, and $\mathbf{l_e}$. The closeness between q and a POI l, denoted as $clo(q, l)$, is computed as the probability of observing l given q:

$$clo(q, l) = \frac{e^{\mathbf{l}\cdot\mathbf{q}}}{\sum_{l' \in \mathcal{L}} e^{\mathbf{l}'\cdot\mathbf{q}}} \tag{13}$$

Then closeness between q and tr, denoted as $clo(q, tr)$, is the sum of $clo(q, l)$ for every $l \in tr$.

The closeness of tr w.r.t. the dynamic context is computed as the sum of the pairwise *normalized co-occurrence similarity* between any two POIs l_i and l_j in tr, denoted as $nsim(l_i, l_j)$:

$$nsim(l_i, l_j) = e^{\mathbf{l_i}\cdot\mathbf{l_j}} / \sum_{l} \sum_{l': l' \neq l} e^{\mathbf{l}\cdot\mathbf{l}'} \tag{14}$$

Overall, the CTQ score $S(u_q, tr)$ is computed as:

$$S(u_q, tr) = \sum_{i=2}^{|tr|-1} \frac{e^{\mathbf{q}\cdot\mathbf{l_i}}}{\sum_l e^{\mathbf{q}\cdot\mathbf{l}}} + \sum_{i=2}^{|tr|-2} \sum_{j=i+1}^{|tr|-1} \frac{e^{\mathbf{l_i}\cdot\mathbf{l_j}}}{\sum_l \sum_{l': l' \neq l} e^{\mathbf{l}\cdot\mathbf{l}'}} \tag{15}$$

Here, we have omitted l_1 and $l_{|tr|}$. This is because all feasible trips share the same l_1 and $l_{|tr|}$ which are the given starting and ending POIs l_s and l_e in the query.

Problem reduction. We construct a directed graph $G = (V, E)$, where each vertex $v_i \in V$ represents POI $l_i \in \mathcal{L}$ and each edge $\overrightarrow{e_{ij}} \in E$ represents the transit from v_i to v_j. We assign *profits* to the vertices and edges. The profit of vertex v_i, denoted as $f(v_i)$, is computed as $f(v_i) = clo(q, l_i)$. The profit of an edge $\overrightarrow{e_{ij}}$, denoted as $f(\overrightarrow{e_{ij}})$, is computed as $f(\overrightarrow{e_{ij}}) = nsim(l_i, l_j)$. For ease of discussion, we use v_1 and $v_{|V|}$ to represent the query starting and ending POIs l_s and l_e, respectively. We set the profits of v_1 and $v_{|V|}$ as zero, since they are included in every feasible trip. We further add *costs* to the edges to represent the trip cost. The cost of edge $\overrightarrow{e_{ij}}$, denoted as $tc(\overrightarrow{e_{ij}})$, is the sum of the transit time cost between l_i and l_j and the visiting time cost of l_j, i.e., $tc(\overrightarrow{e_{ij}}) = tc_v(l_j) + tc_t(l_i, l_j)$.

Based on the formulation above, recommending a trip for query q can be seen as a variant of the *orienteering problem* [9] which finds a path that collects the most profits in G while costs no more than a given budget t_q. We thus reduce the TripRec problem to the following constrained optimization problem:

$$\begin{aligned}
\max\ & \sum_{i=1}^{|V|}\sum_{j=1}^{|V|} x_{ij} \cdot f(v_j) + \sum_{i=2}^{|V|-2}\sum_{j=i+1}^{|V|-1} x_i \cdot x_j \cdot f(\overrightarrow{e_{ij}}) \\
\text{s.t. } & \text{(a) } \sum_{i=1}^{|V|} x_{1i} = x_1 = 1, \quad \text{(b) } \sum_{i=1}^{|V|} x_{i|V|} = x_{|V|} = 1 \\
& \text{(c) } tc_v(v_1) + \sum_{i=1}^{|V|-1}\sum_{j=2}^{|V|} x_{ij} \cdot tc(\overrightarrow{e_{ij}}) \leqslant t_q \\
& \text{(d) } 2 \leqslant p_i \leqslant |V|,\ \forall i \in [2, |V|] \\
& \text{(e) } p_i - p_j + 1 \leqslant (|V|-1)(1 - x_{ij}),\ \forall i, j \in [2, |V|] \\
& \text{(f) } \sum_{j=1}^{|V|} x_{ij} = \sum_{k=1}^{|V|} x_{ki} = x_i \leqslant 1,\ \forall i \in [2, |V|-1]
\end{aligned} \tag{16}$$

Here, x_{ij} and x_i are boolean indicators: $x_{ij} = 1$ if edge $\overrightarrow{e_{ij}}$ is selected, and $x_i = 1$ if vertex v_i is selected. Conditions (a) and (b) restrict the trip to start from v_1 and end at $v_{|V|}$. Condition (c) denotes the time budget constraint. Conditions (d) and (e) are adapted from [19], where p_i denotes the position of v_i in the trip. They ensure no cycles in the trip. Condition (f) restricts to visit any selected POI once.

A. The C-ILP Algorithm

A common approach for the orienteering problems is the *integer linear programming* (ILP) algorithm [3], [14]. However, ILP does not apply directly to our problem. This is because the second term in our objective function in Equation 16, i.e., $\sum_{i=2}^{|V|-1}\sum_{j=i+1}^{|V|-1} x_i \cdot x_j \cdot f(\overrightarrow{e_{ij}})$, is nonlinear. In what follows, we transform Equation 16 to a linear form such that the ILP algorithm [3], [14] can be applied to solve our problem. Such an algorithm finds

the exact optimal trip for TripRec. We denote it as the *C-ILP* algorithm for ease of discussion.

Our transformation replaces the vertex indicators x_i and x_j in Equation 16 with a new indicator x'_{ij}, where $x'_{ij} = 1$ if both v_i and v_j are selected (not necessarily adjacent). We further impose $i < j$ in x'_{ij} to reduce the total number of such indicators by half. This does not affect the correctness of the optimization since $x'_{ij} = x'_{ji}$. Then, Equation 16 is rewritten as follows.

$$\begin{aligned}
&\max \ \textstyle\sum_{i=1}^{|V|}\sum_{j=1}^{|V|} x_{ij}\cdot f(v_j) + \sum_{i=2}^{|V|-2}\sum_{j=i+1}^{|V|-1} x'_{ij}\cdot f(\overrightarrow{e_{ij}}) \\
&\text{s.t. (a)} \sum_{i=1}^{|V|} x_{1i} = 1, \quad \text{(b)} \sum_{i=1}^{|V|} x_{i|V|} = 1 \\
&\text{(f)} \sum_{j=1}^{|V|} x_{ij} = \sum_{k=1}^{|V|} x_{ki} \leqslant 1, \ \forall i \in [2, |V|-1] \\
&\text{(g)} \ x'_{ij} = \sum_{k=1}^{|V|}\sum_{m=1}^{|V|} x_{ik}\cdot x_{jm}, \ \forall i,j \in [2,|V|-1], i<j
\end{aligned} \tag{17}$$

Here, Conditions (c) to (e) are listed in Equation 16 (omitted due to the page limit). Conditions (a), (b), and (f) are the same as those of Equation 16 except that the vertex indicators (x_i) are removed. Condition (g) defines the relationships between x_{ij} and x'_{ij}: if a trip includes a vertex v_i, it must contain an edge starting from v_i (or an edge ending at v_i if $v_i = v_{|V|}$). Thus, for any two vertices v_i and v_j, x'_{ij} equals to 1 if the solution trip contains one edge starting from v_i and another from v_j.

Using x'_{ij}, we transform our objective function into a linear form. Condition (g) is still non-linear (note $x_{ik} \cdot x_{jm}$). We replace it with three linear constraints:

$$\begin{aligned}
&x'_{ij} \leqslant \sum_{k=1}^{|V|} x_{ik}, \forall i,j \in [2,|V|-1], i<j \\
&x'_{ij} \leqslant \sum_{k=1}^{|V|} x_{jk}, \forall i,j \in [2,|V|-1], i<j \\
&x'_{ij} \geqslant \sum_{k=1}^{|V|}\sum_{m=1}^{|V|} (x_{ik}+x_{jm}) - 1, \forall i,j \in [2,|V|-1], i<j
\end{aligned} \tag{18}$$

To show the correctness of the above transformation, assume two vertices v_i and v_j. We consider two cases: (i) At least one vertex (e.g., v_i) is not included in the optimal trip; and (ii) Both vertices are in the optimal trip. Condition (g) in Equation 17 ensures that $x'_{ij} = 0$ in Case (i) and $x'_{ij} = 1$ in Case (ii). We next show that this is also guaranteed by Equation 18. For Case (i), we have $\sum_{k=1}^{|V|} x_{ik} = 0$, which leads to $x'_{ij} \leqslant 0$ according to the first constraint in Equation 18. Since $x'_{ij} \in [0,1]$, we have $x'_{ij} = 0$. For Case (ii), we have $\sum_{k=1}^{|V|} x_{ik} = 1$ and $\sum_{m=1}^{|V|} x_{jm} = 1$. According to the third constraint in Equation 18, we have $x'_{ij} \geqslant 1$. Since $x'_{ij} \in [0,1]$, we have $x'_{ij} = 1$. Combining the two cases, we show that the above transformation retains the constraints of Condition (g).

Algorithm complexity. There are $2 \cdot |E|$ boolean variables in C-ILP, where $|E|$ represents the number of edges in G. To compute the solution, the *lpsolve* algorithm [1] first finds a trip without considering the integer constraints, which can be done in $O(|E|)$ time. Then it refines the trip to find the optimal integral solution. The algorithm uses branch-and-bound to guide the search process. It may need to explore all possible combinations in the worst case, which leads to a worst-case time complexity of $O(2^{|E|})$.

B. The C-ALNS Algorithm

The C-ILP algorithm finds the trip with the highest CTQ score. However, the underlying integer linear program algorithm may incur a non-trivial running time as shown by the complexity analysis above.

To avoid the high running time of C-ILP, we propose a heuristic algorithm named C-ALNS based on *adaptive large neighborhood search* (ALNS) [22]. ALNS is a meta-algorithm to generate heuristic solutions. It starts with an initial solution (a trip in our problem) and then improves the solution iteratively by applying a destroy and a build operator in each iteration. The *destroy operator* randomly removes a subset of the elements (POIs) from the current solution. The *build operator* inserts new elements into the solution to form a new solution. Different destroy/build operators use different heuristics to select the elements to remove/insert. Executing a pair of destroy and build operators can be viewed as a move to explore a neighborhood of the current solution. The aim of the exploration is to find a solution with a higher objective function value.

As summarized in Algorithm 1, our C-ALNS algorithm adapts the ALNS framework as follows: (i) C-ALNS consists of multiple (run_m) ALNS runs (Lines 3). The best trip of all runs and its CTQ score are stored as tr_{opt} and $S(u_q, tr_{opt})$. The best trip within a single run is stored as tr_{r_opt}. The algorithm initializes a solution pool $\mathcal{P}$ (detailed in Section VI-B1) before running ALNS, where a trip from the solution pool is randomly selected to serve as the initial solution of each run of ALNS . (ii) C-ALNS uses multiple pairs of destroy operators $\mathcal{D}$ and build operators $\mathcal{B}$ to enable random selections of the operators to be used ALNS (detailed in Section VI-B2). (iii) C-ALNS uses a *Simulated Annealing* (SA) strategy to avoid falling in local optimum (detailed in Section VI-B3).

1) The Solution Pool: We maintain a subset of feasible trips in the solution pool $\mathcal{P}$, where each trip tr_i is stored with its CTQ score as a tuple: $\langle tr_i, S(u_q, tr_i)\rangle$. At

Algorithm 1: C-ALNS

input : POI Graph G, Query $q = \langle u_q, l_s, l_e, t_q \rangle$
output: Optimal trip tr_{opt}

$tr_{opt} \leftarrow \emptyset, S(u_q, tr_{opt}) \leftarrow -\infty, run \leftarrow 0$;
initialize the solution pool $\mathcal{P}$;
while $run \leq run_m$ **do**
  $tr \leftarrow$ RandomSelect($\mathcal{P}$);
  $tr_{r\_opt} \leftarrow tr$;
  $temp \leftarrow \tau$;
  initialize the weights $\mathcal{D}$ and $\mathcal{B}$;
  $itr \leftarrow 0$;
  while $itr \leq itr_m$ **do**
    $\{d, b\} \leftarrow$ RandSelect($\mathcal{D}, \mathcal{B}$);
    $tr' \leftarrow$ Apply(tr, d);
    $tr' \leftarrow$ Apply(tr, b);
    if $S(u_q, tr') > S(u_q, tr)$ *or* $x^{U(0,1)} < exp(\frac{S(u_q,tr')-S(u_q,tr))}{temp})$ **then**
      $tr \leftarrow tr'$;
      if $S(u_q, tr_{r\_opt}) < S(u_q, tr)$ **then**
        $tr_{r\_opt} \leftarrow tr$;
    $temp \leftarrow temp \times \theta$;
    update the weights of $\mathcal{D}$ and $\mathcal{B}$;
  if $S(u_q, tr_{opt}) < S(u_q, tr_{r\_opt})$ **then**
    $tr_{opt} \leftarrow tr_{r\_opt}$;
  update $\mathcal{P}$;
return tr_{opt}

the beginning of each ALNS run, we select a trip from the solution pool $\mathcal{P}$ and use it as the initial trip for the run. The probability of selecting a trip tr_i is computed as $p(tr_i) = S(u_q, tr_i)/\sum_{tr \in \mathcal{P}} S(u_q, tr)$. At the end of each run, we insert the tuple $\langle tr_{r\_opt}, S(u_q, tr_{r\_opt}) \rangle$ into $\mathcal{P}$, where tr_{r_opt} is the best trip accepted in this run.

We initialize the solution pool with three initial trips generated by a low-cost heuristic based algorithm. This algorithm first creates a trip with two vertices v_1 and $v_{|V|}$ corresponding to the starting and ending POIs l_s and l_e of the query. Then, it iteratively inserts a new vertex into the trip until the time budge is reached. To choose the next vertex to be added, we use the following three different strategies, yielding the three initial trips:

- Choose the vertex v that adds the highest profit to maximize $f^+_\Delta(v) = S(u_q, tr') - S(u_q, tr)$, where tr is the current trip and tr' is the trip after adding v.
- Choose the vertex v that adds the least time cost to minimize $tc^+_\Delta(v)$, where $t^+_\Delta(v) = tc(tr') - tc(tr)$.
- Choose the most cost-effective vertex v that maximizes $f^+_\Delta(v)/t^+_\Delta(v)$.

2) The Destroy and Build Operators: We maintain a set of destroy and build operators as follow.

Destroy operator. Given a trip tr and a removal fraction parameter $\rho \in [0, 1]$, a destroy operator removes $\lceil \rho \cdot (|tr| - 2) \rceil$ vertices from tr. We use four destroy operators with different removal strategies:

Random. This operator randomly selects $\lceil \rho \cdot (|tr| - 2) \rceil$ vertices to be removed.

Least profit reduction. This operator selects $\lceil \rho \cdot (|tr| - 2) \rceil$ vertices with the least profit reduction: $f^-_\Delta(v) = S(u_q, tr) - S(u_q, tr')$, where tr' represents the trip after v is removed from tr. We add randomness to this operator. Given the list of vertices in tr sorted in ascending order of their profits, we compute the next vertex to be removed as $(x^{U(0,1)})^{\psi \cdot \rho}(|tr| - 2)$. Here, x is a random value generated from the Uniform distribution $U(0, 1)$ and the parameter ψ is a system parameter that represents the extent of randomness imposed on this operator. A larger value of ψ leads to less randomness.

Most cost reduction. This operator selects $\lceil \rho \cdot (|tr| - 2) \rceil$ vertices with the largest cost reduction: $t^-_\Delta(v)$. We also randomize this operator in the same way as the least profit reduction operator.

Shaw removal. This operator implements the Shaw removal [22]. It randomly selects a vertex v in tr and removes $\lceil \rho \cdot (|tr| - 2) \rceil$ vertices with the smallest distances to v. We also randomize this operator as we do above.

Build operator. The build operator adds vertices to tr until the time budget is reached. We use four build operators as follows.

Most profit increment. This operator iteratively inserts an unvisited vertex that adds the most profit.

Least cost increment. This operator iteratively inserts an unvisited vertex that adds the least time cost.

Most POI similarity. This operator randomly selects a vertex v_i in tr. Then, it sorts the unvisited vertices by their distances to v_i in our POI embedding space. The unvisited vertices nearest to v_i are added to tr.

Highest potential. This operator iteratively inserts an unvisited vertex v_i that, together with another unvisited vertex v_j, adds the most profit while the two vertices do not exceed the time budget.

Operator choosing. We use a roulette-wheel scheme to select the operators to be applied. Specifically, we associate a weight w to each destroy or build operator, which represents its performance in previous iterations to increase the CTQ score. The probability of selecting an operator o_i equals to its normalized weight (e.g., $o_i.w/\sum_{o \in \mathcal{D}} o.w$ if o_i is a destroy operator). At the beginning of each ALNS run, we initialize the weight of each operator to be 1. After each iteration in a run, we score the applied operators based on their performances. We consider four scenarios: (i) a new global best trip tr_{opt} is found; (ii) a new local best trip within the run is found; (iii) a local best trip within the run is found but it is not new; and (iv) the new trip is worse than the previous trip but is accepted by the Simulated An-

nealing scheme. We assign different scores for different scenarios. The operator scoring scheme is represented as a vector $\pi = \langle \pi_1, \pi_2, \pi_3, \pi_4, \pi_5 \rangle$, where each element corresponds to a scenario, e.g., π_1 represents the score for Scenario (i), and π_5 corresponds to any scenario not listed above. We require $\pi_1 > \pi_2 > \pi_3 > \pi_4 > \pi_5$. Given an operator o_i, its current weight $o_i.w$ and its score $o_i.\pi$, we update the weight of o_i as $o_i.w \leftarrow \kappa \cdot o_i.w + (1 - \kappa) \cdot o_i.w$. Here, κ is a system parameter controlling the weight of the scoring action.

3) Simulated Annealing: We adapt the *simulated annealing* (SA) technique to avoid local optima. Specifically, at the beginning of each ALNS run, we initialize a temperature $temp$ to a pre-defined value τ. After every iteration, a new trip tr' is generated from a previous trip tr. If $S(u_q, tr') < S(u_q, tr)$, we do not discard tr' immediately. Instead, we further test whether $S(u_q, tr') - S(u_q, tr) > -temp \times \log x$ where x is a random value generated from the Uniform distribution $U(0, 1)$ (Algorithm 1, Line 13). If yes, we still replace tr with tr'. We gradually reduce the possibility of keeping a worse new trip by decreasing the value of $temp$ after each iteration by a pre-defined cooling factor θ (Algorithm 1, Line 17).

Algorithm complexity. C-ALNS has run_m ALNS runs, where each run applies itr_m pairs of destroy-build operators. To apply a detroy operator, the algorithm needs to perform $|tr_{avg}|$ comparisons to choose the vertices to remove. To apply a build operator, the algorithm needs to perform $|V|$ comparisons to choose the vertices to add. Here, $|tr_{avg}|$ represents the average length of feasible trips and $|V|$ represents the number of vertices in G. Thus, the time complexity of C-ALNS is $O(run_m \cdot itr_m \cdot (|tr_{avg}| + |V|))$.

VII. Experiments

We evaluate the effectiveness and efficiency of the proposed algorithms empirically in this section. We implement the algorithms in Java. We run the experiments on a 64-bit Windows machine with 24 GB memory and a 3.4 GHz Intel Core i7-4770 CPU.

A. Settings

We use four real-world POI check-in datasets from Flickr (cf. Section III). We perform leave-one-out cross-validation on the datasets. In particular, we use a trip of a user u with at least three POIs as a testing trip tr^*. We use u as the query user, the starting and ending POIs of tr^* as the query starting and ending POIs, and the time cost of tr^* as the query time budget. We use all the other trips in the dataset for training to obtain the context-aware embeddings for the POIs and users.

Let tr be a trip recommended by an algorithm and tr^* be the ground truth. We evaluate the algorithms with three metrics: (i) *Recall* – the percentage of the POIs in tr^* that are also in tr, (ii) *Precision* – the percentage of the POIs in tr that are also in tr^*, (iii) F_1*-score* – the harmonic mean of Precision and Recall. We exclude the starting and ending POIs when computing these three metrics. To keep consistency with two baseline algorithms [3], [14], we further report three metrics denoted as *Recall**, *Precision**, and F_1^**-score*. These metrics are counterparts of Recall, Precision, and F_1-score, but they include the starting and ending POIs in the computation.

We test both our algorithms **C-ILP** (Section VI-A) and **C-ALNS** (Section VI-B). They use the same context-aware POI embeddings as described in Section V. We learn a 13-dimensional embedding with a learning rate η of 0.0005 and a regularization term parameter λ of 0.02. For C-ALNS, we set the removal fraction ρ as 0.2, the operator scoring vector π as $\langle 10, 5, 3, 1, 0 \rangle$, the SA initial temperature as 0.3, and the cooling factor as 0.9995.

Baseline algorithms. We compare with six baseline algorithms, namely **Random**, **Pop**, **MF**, **PersTour**, **POIRank** and its variant **M-POIRank**. Random, Pop, and MF use the same trip generation procedure: they repeatedly add an unvisited POI to the recommended trip until the query time budget exhausts. However, they use different strategies to choose the next POI to visit. Random chooses the next POI randomly. Pop chooses the next POI with the highest popularity (computed by normalized visit frequency) among all unvisited POIs. MF chooses the next POI with the highest preference score (computed by matrix factorization [23]). PersTour [14] recommends the trip that meets the time budget and has the highest sum of *POI scores*. The POI score of a POI l is the weighted sum of its popularity and user interest score, where the popularity is computed with the same method as in Pop, and the user interest score is derived from the query user's previous visiting durations at POIs with the same category as l. We use a weight of 0.5, which is reported to be optimal [14]. POIRank [3] resembles PersTour but differs in how the POI score is computed. It represents each POI as a feature vector of five dimensions: POI category, neighborhood, popularity, visit counts, and visit duration (cf. Section II-B). It computes the POI score of each POI using *rankSVM* with linear kernel and L2 loss [13]. We further test its variant M-POIRank where a weighted *transition score* is added to the POI score. Given a pair of POIs, their transition score is modeled using the Markov model that factorizes the transit probability between the two POIs as the product of the transit probabilities between the

TABLE III: Performance Comparison in Recall, Precision, and F_1-score

City	Edin.			Glas.			Osak.			Toro.		
Algorithm	Rec.	Pre.	F_1	Rec.	Pre.	F_1	Rec.	Pre.	F_1	Rec.	Pre.	F_1
Random	0.052	0.079	0.060	0.071	0.092	0.078	0.057	0.074	0.063	0.045	0.060	0.050
Pop	0.195	0.238	0.209	0.104	0.128	0.112	0.110	0.138	0.121	0.114	0.148	0.125
MF	0.242	0.229	0.233	0.310	0.308	0.307	0.195	0.173	0.181	0.408	0.410	0.407
PersTour	0.455	0.418	0.430	0.589	0.571	0.577	0.406	0.384	0.392	0.431	0.422	0.425
POIRank	0.326	0.326	0.326	0.408	0.408	0.408	0.367	0.367	0.367	0.389	0.389	0.389
M-POIRank	0.318	0.318	0.318	0.387	0.387	0.387	0.328	0.328	0.328	0.379	0.379	0.379
C-ILP (proposed)	**0.555**	**0.527**	**0.538**	**0.659**	**0.646**	**0.651**	**0.497**	**0.492**	**0.494**	**0.618**	**0.601**	**0.608**
C-ALNS (proposed)	*0.554*	*0.527*	*0.537*	*0.657*	*0.645*	*0.649*	*0.496*	*0.491*	*0.493*	*0.616*	*0.598*	*0.607*

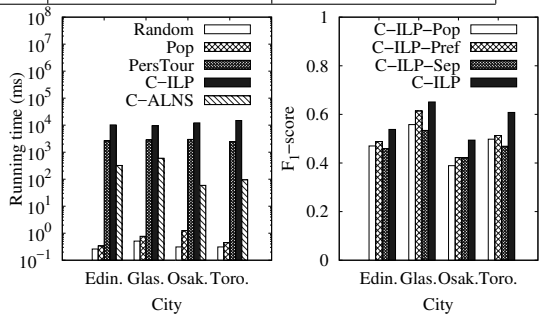
TABLE IV: Performance Comparison in Recall*, Precision*, and F_1^*-score

City	Edin.			Glas.			Osak.			Toro.		
Algorithm	Rec*.	Pre*.	F_1^*	Rec*.	Pre*.	F_1^*	Rec*.	Pre*.	F_1^*	Rec*.	Pre*.	F_1^*
PersTour	0.740	0.633	0.671	0.826	0.782	0.798	0.759	0.662	0.699	0.779	0.706	0.732
POIRank	0.700	0.700	0.700	0.768	0.768	0.768	0.745	0.745	0.745	0.754	0.754	0.754
M-POIRank	0.697	0.697	0.697	0.762	0.762	0.762	0.732	0.732	0.732	0.751	0.751	0.751
C-ILP (proposed)	**0.792**	**0.754**	**0.769**	**0.864**	**0.844**	**0.853**	**0.793**	**0.740**	**0.763**	**0.842**	**0.800**	**0.818**
C-ALNS (proposed)	*0.792*	*0.752*	*0.768*	*0.862*	*0.843*	*0.852*	*0.792*	*0.739*	*0.762*	*0.841*	*0.798*	*0.815*

five POI features of the two POIs.

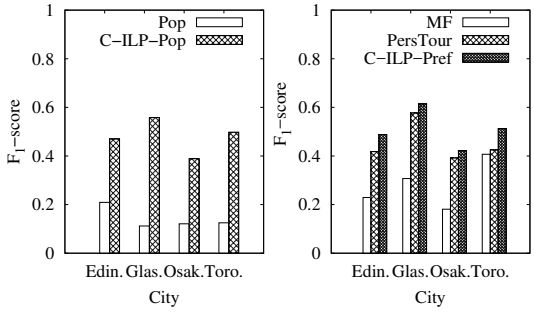
Overall performance. We summarize the results in Tables III and IV (Random, Pop, and MF are uncompetitive and are omitted in Table IV due to space limit). We highlight the best result in **bold** and the second best result in *italics*. We see that both C-ILP and C-ALNS consistently outperform the baseline algorithms. C-ILP outperforms PersTour, the baseline with the best performance, by $25\%, 13\%, 26\%$, and 43% in F_1-score on the datasets Edinburgh, Glasgow, Osaka, and Toronto, respectively. C-ALNS has slightly lower scores than those of C-ILP, but the difference is very small (0.002 on average). This confirms the capability of our heuristic algorithm C-ALNS to generate high quality trips.

We compare the running times of C-ILP and C-ALNS in Figure 3a. For completeness, we also include the running times of Random, Pop, and PersTour, but omit those of MF and POIRank as they resemble that of PersTour. C-ALNS outperforms C-ILP and PersTour by orders of magnitude (note the logarithmic scale). The average running times of C-ILP and PersTour are 10^4 ms and 2.5×10^3 ms, respectively, while that of C-ALNS is only around 300 ms, 600 ms, 60 ms, and 100 ms for the four datasets, respectively. Compared with C-ILP, C-ALNS reaches almost the same F_1-score while reducing the running time by up to 99.4%. Compared with PersTour, C-ALNS obtains up to 43% improvement in F_1-score while reducing the running time by up to 97.6%. Random and Pop have the smallest running times but also very low trip quality as shown in Table III.



(a) C-ILP vs. C-ALNS (b) Impact of factors

Fig. 3: Comparisons among proposed algorithms



(a) POI popularity (b) User preferences

Fig. 4: Impact of model learning capability

B. Results

Impact of different factors. To investigate the contributions of POI popularities, user preferences, and co-occurring POIs in our embeddings, we implement two variants of C-ILP, namely, **C-ILP-Pop** and **C-ILP-Pref**. These two variants use POI embeddings that only learns POI popularities and jointly learns POI popularities and user preferences, respectively. Fig. 3b shows a comparison among C-ILP-Pop, C-ILP-Pref, and C-ILP. We see that the F_1-score increases as the POI embeddings incorporate more factors. This confirms the impact of the three factors. Moreover, we see that on the Edinburgh and Toronto datasets where POIs have more diverse POI co-occurrences (cf. Section III), the improvement of C-ILP (with co-occurring POIs in the embeddings) over C-ILP-Pref is more significant. This demonstrates the effectiveness of our model to learn the POI co-occurrences. We also implement an algorithm

that separately learns the impact of POI popularities, user preferences, and co-occurring POIs, denoted as **C-ILP-Sep**. The algorithm considers equal contribution of the three factors to recommend trips. We see that C-ILP outperforms C-ILP-Sep consistently. This confirms the superiority of joint learning in our algorithm.

Impact of model learning capability. To further show that our proposed POI embedding model has a better learning capability, we compare C-ILP-Pop with Pop in Fig. 4a, since these two algorithms only consider POI popularity. Similarly, we compare C-ILP-Pref with the baseline algorithms that considers user preferences, i.e., MF and PersTour, in Fig. 4b. In both figures, our models produce trips with higher F_1-scores, which confirms the higher learning capability of our models.

VIII. Conclusions

We proposed a context-aware model for POI embedding. This model jointly learns the impact of POI popularities, co-occurring POIs, and user preferences over the probability of a POI being visited in a trip. To showcase the effectiveness of this model, we applied it to a trip recommendation problem named TripRec. We proposed two algorithms for TripRec based on the learned embeddings. The first algorithm, C-ILP, finds the exact optimal trip by solving TripRec as an integer linear programming problem. The second algorithm, C-ALNS, finds a heuristically optimal trip but with a much higher efficiency based on the adaptive large neighborhood search technique. We performed extensive experiments on real datasets. The results showed that the proposed algorithms using our context-aware POI embeddings consistently outperform state-of-the-art algorithms in trip recommendation quality, and the advantage is up to 43% in F_1-score. C-ALNS reduces the running time for trip recommendation by 99.4% comparing with C-ILP while retaining almost the same trip recommendation quality, i.e., only 0.2% lower in F_1-score.

Acknowledgement

This work is partially supported by Australian Research Council Discovery Project DP180103332.

References

[1] M. Berkelaar, K. Eikland, P. Notebaert, et al. lpsolve: Open source (mixed-integer) linear programming system. *Eindhoven U. of Technology*, 2004.

[2] I. Brilhante, J. A. Macedo, F. M. Nardini, R. Perego, and C. Renso. Where shall we go today?: planning touristic tours with tripbuilder. In *CIKM*, pages 757–762, 2013.

[3] D. Chen, C. S. Ong, and L. Xie. Learning points and routes to recommend trajectories. In *CIKM*, pages 2227–2232, 2016.

[4] A.-J. Cheng, Y.-Y. Chen, Y.-T. Huang, W. H. Hsu, and H.-Y. M. Liao. Personalized travel recommendation by mining people attributes from community-contributed photos. In *ACM MM*, pages 83–92, 2011.

[5] C. Cheng, H. Yang, I. King, and M. R. Lyu. Fused matrix factorization with geographical and social influence in location-based social networks. In *AAAI*, pages 17–23, 2012.

[6] S. Feng, X. Li, Y. Zeng, G. Cong, Y. M. Chee, and Q. Yuan. Personalized ranking metric embedding for next new poi recommendation. In *IJCAI*, pages 2069–2075, 2015.

[7] Y. Ge, Q. Liu, H. Xiong, A. Tuzhilin, and J. Chen. Cost-aware travel tour recommendation. In *SIGKDD*, pages 983–991, 2011.

[8] A. Gionis, T. Lappas, K. Pelechrinis, and E. Terzi. Customized tour recommendations in urban areas. In *WSDM*, pages 313–322, 2014.

[9] B. L. Golden, L. Levy, and R. Vohra. The orienteering problem. *Naval Research Logistics*, 34(3):307–318, 1987.

[10] S. Henry, C. Cuffy, and B. T. McInnes. Vector representations of multi-word terms for semantic relatedness. *Journal of Biomedical Informatics*, 77:111–119, 2018.

[11] H.-P. Hsieh and C.-T. Li. Mining and planning time-aware routes from check-in data. In *CIKM*, pages 481–490, 2014.

[12] T. Kurashima, T. Iwata, G. Irie, and K. Fujimura. Travel route recommendation using geotags in photo sharing sites. In *CIKM*, pages 579–588, 2010.

[13] C.-P. Lee and C.-J. Lin. Large-scale linear ranksvm. *Neural Computation*, 26(4):781–817, 2014.

[14] K. H. Lim, J. Chan, C. Leckie, and S. Karunasekera. Personalized tour recommendation based on user interests and points of interest visit durations. In *IJCAI*, pages 1778–1784, 2015.

[15] Q. Liu, Y. Ge, Z. Li, E. Chen, and H. Xiong. Personalized travel package recommendation. In *ICDM*, pages 407–416, 2011.

[16] X. Liu, Y. Liu, and X. Li. Exploring the context of locations for personalized location recommendations. In *IJCAI*, pages 1188–1194, 2016.

[17] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[18] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *NIPS*, pages 3111–3119, 2013.

[19] C. E. Miller, A. W. Tucker, and R. A. Zemlin. Integer programming formulation of traveling salesman problems. *JACM*, 7(4):326–329, 1960.

[20] V. Rakesh, N. Jadhav, A. Kotov, and C. K. Reddy. Probabilistic social sequential model for tour recommendation. In *WSDM*, pages 631–640, 2017.

[21] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback. In *UAI*, pages 452–461, 2009.

[22] S. Ropke and D. Pisinger. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Science*, 40(4):455–472, 2006.

[23] R. Salakhutdinov and A. Mnih. Bayesian probabilistic matrix factorization using markov chain monte carlo. In *ICML*, pages 880–887, 2008.

[24] B. Thomee, D. A. Shamma, G. Friedland, B. Elizalde, K. Ni, D. Poland, D. Borth, and L.-J. Li. Yfcc100m: the new data in multimedia research. *Communications of the ACM*, 59(2):64–73, 2016.

[25] P. Wang, J. Guo, Y. Lan, J. Xu, S. Wan, and X. Cheng. Learning hierarchical representation model for nextbasket recommendation. In *SIGIR*, pages 403–412, 2015.

[26] X. Wang, C. Leckie, J. Chan, K. H. Lim, and T. Vaithianathan. Improving personalized trip recommendation by avoiding crowds. In *CIKM*, pages 25–34, 2016.

[27] C. Zhang, H. Liang, K. Wang, and J. Sun. Personalized trip recommendation with poi availability and uncertain traveling time. In *CIKM*, pages 911–920, 2015.