Multi-User Mobile Sequential Recommendation: An Efficient Parallel Computing Paradigm

Zeyang Ye
Stony Brook University
zeyang.ye@stonybrook.edu

Lihao Zhang
Stony Brook University
lihao.zhang@stonybrook.edu

Keli Xiao*
Stony Brook University
keli.xiao@stonybrook.edu

Wenjun Zhou
University of Tennessee Knoxville
wzhou4@utk.edu

Yong Ge
University of Arizona
yongge@email.arizona.edu

Yuefan Deng
Stony Brook University
yuefan.deng@stonybrook.edu

ABSTRACT

The classic mobile sequential recommendation (MSR) problem aims to provide the optimal route to taxi drivers for minimizing the potential travel distance before they meet next passengers. However, the problem is designed from the view of a single user and may lead to overlapped recommendations and cause traffic problems. Existing approaches usually contain an offline pruning process with extremely high computational cost, given a large number of pick-up points. To this end, we formalize a new multi-user MSR (MMSR) problem that locates optimal routes for a group of drivers with different starting positions. We develop two efficient methods, PSAD and PSAD-M, for solving the MMSR problem by ganging parallel computing and simulated annealing. Our methods outperform several existing approaches, especially for high-dimensional MMSR problems, with a record-breaking performance of 180x speedup using 384 cores.

CCS CONCEPTS

• **Information systems → Mobile information processing systems**; *Data mining*; • **Computing methodologies → Parallel computing methodologies**;

KEYWORDS

Mobile sequential recommendation, parallel computing, simulated annealing, potential travel distance

ACM Reference Format:
Zeyang Ye, Lihao Zhang, Keli Xiao, Wenjun Zhou, Yong Ge, and Yuefan Deng. 2018. Multi-User Mobile Sequential Recommendation: An Efficient Parallel Computing Paradigm. In *KDD '18: The 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, August 19–23, 2018, London, United Kingdom.* ACM, New York, NY, USA, Article 4, 10 pages. https://doi.org/10.1145/3219819.3220111

*Corresponding author.

KDD '18, August 19–23, 2018, London, United Kingdom

ACM ISBN 978-1-4503-5552-0/18/08…$15.00
https://doi.org/10.1145/3219819.3220111

1 INTRODUCTION

Mobile sequential data (e.g., GPS trajectories, POI check-in sequence, etc.), which captures the characteristics of human mobility, has been widely used in the development of mobile recommender systems. While a number of applications, such as mobile sequential recommendation (MSR), POI recommendation, and travel package recommendation, have been developed, the community of big data analytics is still seeking effective ways to handle two major challenges. First, all recommendations should take the social interests into consideration. Second, making fast and precise recommendations to users require the system to efficiently handle high-dimensional problems. In this paper, we focus on solving these two challenges based on the MSR problem.

The MSR problem formalized by Ge et al. [9] is designed to recommend a travel route for a taxi driver so that the potential travel distance (PTD) before picking up the next passenger is minimized. The recommended travel route consists of a sequence of pick-up locations obtained from historical taxi GPS trace data. The MSR problem has been recognized as an important and classic application in mobile recommender systems [18, 26]. Furthermore, with the advent of the autonomous driving and sharing economy, this problem becomes more important because of the wider adaptation of autonomous driving. The autonomous cars that seek passengers can also be considered as the general case of taxicabs in the MSR problem. Therefore, efficiently solving the MSR problem can benefit both drivers and passengers for the taxi system as well as autonomous driving systems in the near future.

The search algorithms of the MSR problem possess two major challenges. First, the search algorithms must handle multi-user queries and recommend distinct routes for different drivers. This issue was not discussed in [9]. Although Huang et al. [10] discovered that the recommended routes for different taxi drivers could vary under different constraints regarding distances and destinations, they did not provide a clear method to determine these constraints. Second, the search algorithms should be efficient enough for the volatile traffic dynamics. To solve the MSR problem, A two-stage framework which contain an offline pruning and an online searching procedure were usually adopted [9, 10]. The extremely high computational cost of the pruning process inundates these two-stage approaches for the high-dimensional MSR problems, and efficient and precise route recommendation is not practical. To this end, we provide the following solutions.

For the first challenge, we examine the MSR problem and two classic mathematical optimizations: the traveling salesman problem (TSP) and the vehicle routing problem (VRP). Given n locations and

a starting position, the MSR problem finds a route with length of $L \leq N$ while the TSP searches a route with $L = N$. The VRP is a general case of TSP. It recommends K routes to cover the N locations with route length $L_1, L_2, \ldots, L_k$ such that $\sum_{i=1}^{K} L_i = N$. Therefore, we construct a new multi-user MSR (MMSR) problem that aims to recommends K different routes with route length $L_1, L_2, \ldots, L_k$ such that $\sum_{i=1}^{K} L_i \leq N$. For the TSP and VRP, the criterion to evaluate a route usually based on total travel distance. Similarly, for MSR and MMSR problems, we choose the PTD to be the criterion. By building up the MMSR problem, we can thoroughly avoid the duplications of the pick-up locations in the recommended routes.

For the second challenge regarding high computational cost, we propose an efficient parallel framework for solving high-dimensional MMSR problems. We improve a recently proposed stochastic method, SA (simulated annealing) random search [24, 25] and focus on designing its parallel methods for achieving reliable performance of speedup. Based on the literature on parallel computing, attaining consistent speedup for SA is still a challenging problem. That is, increasing the number of cores may not always result in correspondent speedup. The latest published records show that the largest speedup is only 19.6x for solving the traveling salesperson problem [20]. It is difficult to obtain further improvements due to the increasing communication cost among different cores. It is even harder to parallelize SA for solving the VRP, because of the difficulties in assigning different cores to a suitable domain. The MSR problem, serving as a similar problem of TSP and VRP, shares the same difficulties as well.

Therefore, we propose a two-layer parallelization methodology in this paper based on the SA random search to significantly improve the computational efficiency for solving the MMSR problem. Specifically, a domain decomposition and a mixing algorithm are designed to divide the domain in a proper way so that all the computing cores can share the computing efforts without incurring too much additional communication burden. For comparison, we apply two other parallel algorithms, namely, CDR [3] and Lou [14], to SA random search. Besides, we parallelize two benchmark methods proposed in [9] and [10] to show the parallel performance of the other methods in addition to SA. We conduct intensive experiments to compare these methods in both real-world and synthetic datasets.

The contributions of this work are summarized as follows:

- While the original MSR problem only asks for one route for a single user, we construct a new MMSR problem that aims to provide distinct routes for multi-users.
- We develop two parallel methods, PSAD and PSAD-M, for solving the MMSR problem. We show that our methods can efficiently recommend different routes for multi-users based on 100,000 pick-up points, significantly outperform all other benchmarks.
- We break the published speedup record of parallel simulated annealing by discovering that our parallel methods can achieve up to 180x speedup with 384 cores. Compared to the published record of 19.6x when parallel SA is adapted for use in other optimization problems, our framework maximizes the strength of parallel SA in the MMSR problem.

The rest of this paper is organized as follows. In Section 2, we discuss preliminaries regarding the MSR problem. In Section 3, we introduce the MMSR problem and our parallelization techniques for both the benchmark methods and SA random search. In Section 4, we show our experimental results and the analysis of the parameters. We discuss the related work for MSR problem and parallel SA in Section 5, and then we conclude the paper in Section 6.

2 PRELIMINARIES

Let $C = \{c_1, c_2, \ldots, c_N\}$ be a set of N potential pick-up points, c_0 be the current position of the taxi driver and $P = \{P(c_1), P(c_2), \ldots, P(c_N)\}$ be the set of the successful pick-up probabilities corresponding to each potential pick-up point. From perspective of practice, P_c should be updated frequently to represent a dynamic traffic status, and an efficient recommendation system should be able to handle the frequent traffic changes.

Given route length L, we can have a sequence $\overrightarrow{r_i} = (c_{i_1}, c_{i_2}, \ldots, c_{i_L})$ as a route. Then there is in total $M = \binom{N}{L} L!$ sequences in C and we let $R = \{\overrightarrow{r_1}, \overrightarrow{r_2}, \ldots, \overrightarrow{r_M}\}$ be the set of all possible routes through the potential pick-up point.

Definition 2.1 (MSR). The objective is to search for the traveling route with the minimal potential travel distance (PTD) before the next successful pick-up,

$$\min_{\overrightarrow{r_i} \in R} PTD\big(c_0, \overrightarrow{r_i}\big), \tag{1}$$

where the potential travel distance is computed as:

$$PTD\big(c_0, \overrightarrow{r_i}\big) = D\big(c_0, \overrightarrow{r_i}\big) \cdot F\big(\overrightarrow{r_i}\big), \tag{2}$$

where $D\big(c_0, \overrightarrow{r_i}\big)$ is the travel distance when the driver reaches each of the pick-up points along the route. It can be computed as:

$$D\big(c_0, \overrightarrow{r_i}\big) = \left(d\big(c_{i_0}, c_{i_1}\big), \ldots, \sum_{j=1}^{L} d\big(c_{i_{j-1}}, c_{i_j}\big), D_{max}\right), \tag{3}$$

where $c_{i_0} = c_0, \forall i \in [1, M]$; D_{max} is a penalty term for not picking up a passenger along the route, and $d(c_{i_{j-1}}, c_{i_j})$ computes the distance between $c_{i_{j-1}}$ and c_{i_j}. $F(\overrightarrow{r_i})$ records the potential pick-up probability when the driver reaches each pick-up points along the route. It is computed as:

$$F\big(\overrightarrow{r_i}\big) = \left(P\big(C_{i_1}\big), \overline{P(C_{i_1})}P\big(C_{i_2}\big), \ldots, \prod_{j=1}^{L-1} \overline{P(C_{i_{L-1}})}P\big(C_{i_L}\big), \prod_{j=1}^{L} \overline{P(C_{i_j})}\right), \tag{4}$$

where $\overline{P(C_{i_j})} = 1 - P(C_{i_j})$.

According to above definition, the MSR problem is designed for recommending a route for one driver without considering situations of others. In this case, if K drivers asking for the recommendations at the same time, the MSR system will process K independent recommendation search for each the drivers. However, the K recommended routes may overlap with the other, especially when the two drivers are in the same area. Therefore, a good recommendation should consider its influence on others to avoid duplicated recommendations. To this end, we propose a new multi-user mobile sequential (MMSR) problem as follows.

Definition 2.2 (MMSR). Given K drivers requesting sequential recommendation, C_0 is the set of positions of the these drivers, the objective of the MMSR problem is to search for K routes with the minimal multi-user potential travel distance (MPTD) before the successful pick-up of each route:

$$\begin{aligned} &\min_{R_i \subseteq R} PTD_K(C_0, R_i), \\ &\text{s. t. } \overrightarrow{r_{i_a}} \cap \overrightarrow{r_{i_b}} = \emptyset, \forall a, b \in [1, K], a \neq b, \end{aligned} \tag{5}$$

where $R_i = \{\overrightarrow{r_{i_1}}, \overrightarrow{r_{i_2}}, \ldots, \overrightarrow{r_{i_K}}\}$, and the MPTD, $PTD_K(C_0, R_i)$ is computed as:

$$PTD_K(C_0, R_i) = \sum_{j=1}^{K} PTD(c_0, \overrightarrow{r_{i_j}}). \tag{6}$$

When $K = 1$, Eqs. (6) and (2) are equivalent. The MMSR problem is formalized from a more practical view but also with an extremely high computational cost, given $\binom{N}{K \cdot L}(K \cdot L)!$ possible combinations. It is approximately $\left(N - \frac{K}{2}\right)^{K \cdot L}$ times more than the MSR's $\binom{N}{L}L!$ combinations.

3 METHODOLOGY

For the MMSR problem, the number of possible route combinations grows exponentially with an increasing number of users K. On the other hand, there is an urge from users longing for the recommendation. Considering the characteristic of independence for the optimal route searching, we propose to apply parallel computing to allow multiple computing cores to collaboratively resolve the overwhelming computational task.

3.1 Parallelizing SA by Domain Decomposition

SA is a stochastic optimization method aiming to find the global optimization of a non-convex objective function [19]. The serial algorithm[1] of SA consists of four key components (initial condition, move generation, cooling schedule, and stopping criterion) and processes as follows. It first generates an initial solution. Then, the algorithm start iterating. It performs the move generation by perturbing its solution with a small movement. If the new solution becomes better, it updates the current solution. Otherwise, it only accepts the solution based on a probability function, which is positively correlated to a parameter, named *temperature*. The temperature is then cooled down a little. While the algorithm does not meet the stopping condition, it returns to the beginning of the iteration and performs the move generation again. Although MMSR is a new problem, we adopt and adjust the SA random search in [5], which specifies the SA for solving the MSR problem, as the fundamental serial (single-core) algorithm.

We propose a parallel simulated annealing with a domain decomposition (PSAD) method to speed up the searching based on the SA random search in [24]. For P_c computing processes, $Pr_1, Pr_2, \ldots, Pr_{P_c}$, we require that K is divisible by P_c. PSAD first randomly divide the pick-up point set C into P_c mutually exclusive subsets, $C_1, C_2, \ldots, C_{P_c}$, each with $\lfloor N/P_c \rfloor$ or $\lfloor N/P_c \rfloor + 1$ pick-up points and assign C_i to computing process P_i. Since there are K routes for P_c processes, process Pr_i are responsible for $\overrightarrow{r}_{(i-1)K/P_c+1}, \ldots, \overrightarrow{r}_{iK/P_c}$ routes. Pr_i randomly initializes the routes based on their current positions of the taxi drivers and updates C_i by excluding the pick-up points in the routes. Pr_i then performs the iterations following four main procedures as below.

SA Step. In each step, Pr_i performs the move generation, random search in [24], and the exponential cooling in SA. At the same time, Pr_i needs to make sure that C_i and its routes have no pick-up points in common.

Rotation. Because the pick-up points set are divided randomly, the pick-up points in one process may be suitable for the routes in the other processes. Then, parallel operations need to be performed to make sure that each pick-up point should have the same probability to access to all the routes. Therefore, in every $step_{ro}$, all P_c processes perform rotation. In the rotation procedure, PSAD can either rotate the current routes or the pick-up point subsets of each process. We normally rotate routes for less communication cost. While rotating, Pr_i sends its current routes to Pr_{i+1} and Pr_{P_c} sends the routes to Pr_0. In this way, in P_c rotations, all the routes have been exposed to the pick-up point set $\cup_i C_i$.

Interaction. One route may need a pick-up point from the other routes in addition to $\cup_i C_i$. Therefore, after P_c rotations, each process gathers the routes for all the other processes and perform $step_{in}$ SA steps only using the pick-up points in the routes. By this means, after P_c rotations and one interaction, all the routes have been exposed to the pick-up point set C just as a serial SA does.

Shuffling. Different pick-up point subsets result in different parallel performance. A good subset may contain either all the optimal pick-up points or no pick-up point for a route, While a bad subset may contain a few optimal pick-up points for each route. To diminish the effect of the randomness, PSAD reshuffle the pick-up points in these subsets in every P_c rotations.

Finally, by repeating the procedures above, PSAD can decompose the domain C and locate the optimal or near optimal routes for the MMSR problem in parallel.

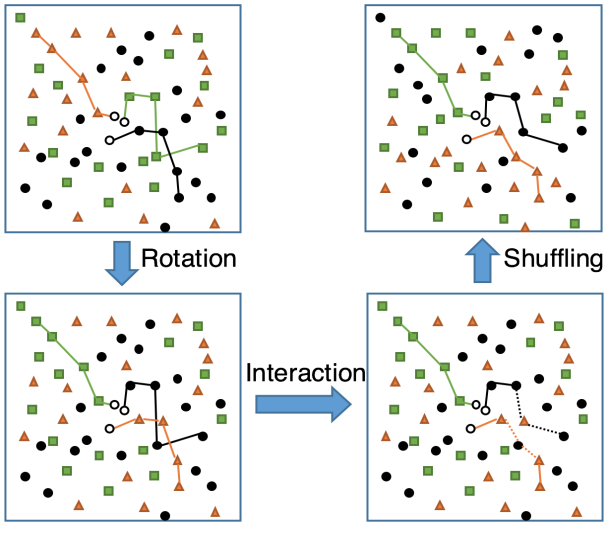


Figure 1: Parallel operations for PSAD.

[1] In this paper, we refer to the single-core version of algorithms as the *serial algorithms*.

We illustrate aforementioned parallel operations by a simplified example in Figure 1. Suppose we have three computing processes (cores), Pr_1, Pr_2, and Pr_3, to handle the MMSR problem. We use three different colors/shapes to represent assigned pick-up points subset and the current considering routes for the three computing processes. The three white circles are the current positions of drivers. First, in the *rotation* operation, the three processes rotate (exchange) their current routes. For example, Pr_1 sends its route to Pr_2; Pr_2 sends its route to Pr_3; and Pr_3 sends its route to Pr_1. Second, in the *interaction* operation, each process gains access to all three routes and optimizes those routes based on their pick-up points subsets. As shown in Figure 1, the black (circle) route and the orange (triangle) route swap one pick-up point with each other after the interaction. The swap reduces the travel distances for both routes. Last, in the *shuffling* operation, each process obtains a new subset of pick-up points from other processes. The three operations are repeated until we locate the optimal route, evaluated by *MPTD*.

3.2 Parallelizing SA by Mixing

One limitation of PSAD is that the parallel size cannot exceed K. We therefore develop an improved method with a mixing process to further speed up the PSAD method. With the add-on effect of mixing, the new parallel method is called parallel SA by domain decomposition and mixing (PSAD-M).

Let $P_c = P_m K$. Similar to PSAD, PSAD-M randomly decomposes the domain into K mutually exclusive subsets, $C_1, C_2, \ldots, C_K$, with P_m processes as a group to deal with one subset. PSAD-M still performs rotation, interaction, and shuffling, and mixing is added between the two consecutive rotation procedures. We hope that by adding mixing, the SA step procedure in PSAD can be accelerated. That is, P_m SA steps in PSAD are equivalent to P_m processes in PSAD-M each making one SA step. In this way, PSAD-M achieves P_m (x) speedup. Then, the way to mix needs to be carefully studied.

There are two extreme cases. First, in PSAD-M, P_m processes mix every step. Specifically, after the P_m processes each making one SA step, they obtain P_m new routes. Among them, P_m processes all adopt the route with the lowest PTD as their current route and perform the next SA step. In this case, all the SA steps are performed based on the most up-to-date route. It is very efficient especially when the temperature is very low where in each SA step, the new routes always get rejected. P_m rejected SA steps in PSAD are equivalent to P_m processes in PSAD-M each making one rejected SA step. Under this circumstance, in terms of steps, there will be P_m (x) speedup. However, in this case, the P_m processes need to communicate every step to obtain the route with the lowest PTD. The additional communication cost offsets the saved computational cost. Second, in PSAD-M, P_m processes does not mix at all. In this case, PSAD-M performs P_m independent PSAD and there is zero communication cost. However, although using P_m times more processes, PSAD-M just runs as P_m independent PSADs with no speedup.

Therefore, there needs to be a mixing pattern and a mixing period to make sure that all the processes mix efficiently to reduce the computational cost without incurring too much communication cost. To construct the mixing pattern, we define two concepts, the distance and the neighborhood.

Definition 3.1. Let $\overrightarrow{r_i} = (C_{i_1}, C_{i_2}, \ldots, C_{i_L})$ and $\overrightarrow{r_j} = (C_{j_1}, C_{j_2}, \ldots, C_{j_L})$. The distance between $\overrightarrow{r_i}$ and $\overrightarrow{r_j}$ is defined as:

$$Dist(\overrightarrow{r_i}, \overrightarrow{r_j}) = \frac{1}{L} \sum_{k=1}^{L} \beta(C_{i_k}, C_{j_k}), \tag{7}$$

where

$$\beta(x, y) = \begin{cases} 0, & \text{if } x = y \\ 1, & \text{if } x \neq y. \end{cases} \tag{8}$$

According to Definition 3.1., we further define the *neighborhood* as follows.

Definition 3.2. The neighborhood of $\overrightarrow{r_i}$ with distance $d_0 \geq 0$ is defined as:

$$\mathcal{N}(\overrightarrow{r_i}, d_0) = \{\overrightarrow{r_j} \mid Dist(\overrightarrow{r_i}, \overrightarrow{r_j}) \leq d_0\}. \tag{9}$$

Notice that $\overrightarrow{r_i} \in \mathcal{N}(\overrightarrow{r_i}, d_0)$. We then design mixing pattern and mixing period to determine an adaptive distance d_0. Then in each group, among the P_m routes from P_m processes at step n, let $\overrightarrow{r_i}$ be the one with the lowest PTD. The mixing pattern and mixing period aim to restrict all P_m routes in $\mathcal{N}(\overrightarrow{r_i}, d_0)$.

Mixing pattern. In PSAD-M, the temperature is initially high and gradually decreases to zero in the end, which indicates its tolerance of the routes with larger PTD, high at the beginning and low at the end of a PSAD-M run. To adapt d_0 to this trend, we introduce a metric, acceptance rate, and a parameter $step_{neigh}$. The acceptance rate is updated every $step_{acc}$ steps and records the average acceptance probability of a new move during these steps. Also, for one process in PSAD-M, the largest distance between any of the two routes in $step_{neigh}$ steps is set to be d_0. There will be a large computational cost if we compute the actual distance, but we can approximate d_0 as the expected distance in $step_{neigh}$ steps with the help of acceptance rate.

Theorem 3.3. *Let acceptance rate be r, the pick-up points number be n, and the route length be L. Then, the expected change in $step_{neigh}$ SA steps is:*

$$d_0 = \frac{LE(A_j)}{L} = 1 - \left(1 - \frac{r}{L} - \frac{r(L-1)}{Ln}\right)^{step_{neigh}}. \tag{10}$$

Proof. Let $\overrightarrow{r_i} = (C_{i_1}, C_{i_2}, \ldots, C_{i_L})$ be the route before $step_{neigh}$ SA steps. Let

$$A_j = \begin{cases} 0, & \text{if } C_{i_j} \text{ does not change in } step_{neigh} \text{ steps} \\ 1, & \text{if } C_{i_j} \text{ changes in } step_{neigh} \text{ steps.} \end{cases}$$

In one SA step, according to the move generation, there is probability $1/L$ to choose C_{i_1} for the first time and $1/n$ for the second time. Based on the acceptance rate, we have

$$P_1 = P(\text{ first choosing } C_{i_j} \text{ and accept}) = r/L$$

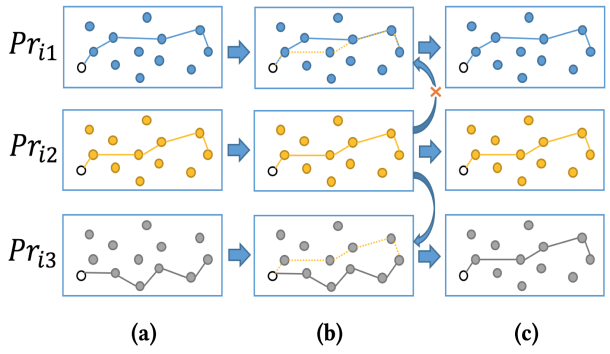


Figure 2: The mixing procedure on PSAD-M.

and

$$P_2 = P(\text{ second choosing } C_{i_j} \text{ and accept}) = \frac{L-1}{L} \cdot \frac{1}{n} \cdot r.$$

Therefore, we get

$$P_3 = P(C_{i_j} \text{ does not change in one steps }) = 1 - P_1 - P_2.$$

The expected change for C_{i_j} in $step_{neigh}$ steps is

$$E(A_j) = 1 - P_3^{step_{neigh}},$$

and the expected change for $\overrightarrow{r_i}$ in $step_{neigh}$ steps is

$$\sum_{j=1}^{L} E(A_j) = LE(A_j).$$

Finally, we get the expected change d_0 as:

$$d_0 = \frac{LE(A_j)}{L} = 1 - \left(1 - \frac{r}{L} - \frac{r(L-1)}{Ln}\right)^{step_{neigh}}.$$

□

According to Theorem 3.3, our PSAD-M can adjust the neighborhood adaptively by setting different values of $step_{neigh}$.

Mixing period. In PSAD-M, from the computational aspect, it is the best to check if all the new routes from P_m processes are within the neighborhood. However, to reduce the communication cost, we design a new strategy for mixing period. Let mixing period be $step_m$. If in the last mixing process, all the new routes stay within the neighborhood, $step_m$ increases by one to allow each process to perform the SA process more independently. Otherwise, $step_m$ decreases by one to restrict the independence for each process.

Figure 2 demonstrates an example of the mixing process in each group. Suppose there are three processes, Pr_{i1}, Pr_{i2}, and Pr_{i3} in group i optimizing the same route. Current routes of Pr_{i1}, Pr_{i2}, and Pr_{i3} are represented by $\overrightarrow{r_{i1}}$, $\overrightarrow{r_{i2}}$, and $\overrightarrow{r_{i3}}$ respectively. In Figure 2 (b), the PTD of $\overrightarrow{r_{i2}}$ is lower than other two, so $\overrightarrow{r_{i2}}$ sends a copy of its route to $\overrightarrow{r_{i1}}$ and $\overrightarrow{r_{i3}}$. According to Eq. (7), $Dist(\overrightarrow{r_{i1}}, \overrightarrow{r_{i2}}) = 0.2$ as these two routes are different for one pick-up point, and $Dist(\overrightarrow{r_{i3}}, \overrightarrow{r_{i2}}) = 0.8$ as they only have the last pick-up point in common. According to Eq. (10), if in previous $step_{neigh}$ SA steps, the d_0 of Pr_{i2} is calculated to be 0.5, then Pr_i1 accepts the route from Pr_{i2} while Pr_{i3} rejects it. In this mixing process, $step_m$ decreases by one since Pr_i3 does not lie in the neighborhood of the best route $\overrightarrow{r_{i2}}$. Finally, after the mixing process, in Figure 2 (c), Pr_{i2} and Pr_{i3} performs SA steps as $\overrightarrow{r_{i2}}$ as their current route while Pr_{i1} uses $\overrightarrow{r_{i1}}$.

4 EXPERIMENTS

In this section, we discuss the data processing and the experimental results.

4.1 Data and Preprocessing

Real-world Data: Beijing (BJ). The data contains 12,000 taxi trajectories during 6-7 pm in Beijing, China. All the trajectories locate on the urban area with longitude from 116.05 to 116.75 and latitude from 39.65 to 40.17, a zone is decomposed into 1,000×1,000 subzones. In each subzone, if the empty taxis have entered it for more than 5 times and have picked up at least one passenger, its centroid is considered as a pick-up point. Its pick-up probability is calculated as the ratio of the successful pick-up number to the number of passing taxis in this subzone. With this, we obtain 21,824 pick-up points whose coordinates are normalized and D_{max} is set to be 1. In these experiments, we optimize 96 routes each with different starting positions. They are set to be a 12 by 8 mesh lie in the coordinates $[0.50, 0.55] \times [0.50, 0.55]$.

Synthetic Data. We also generate synthetic data for 100,000 pick-up points. The pick-up points are uniformly distributed in a $[0, 1] \times [0, 1]$ square and their pick-up probabilities are normally distributed with mean 0.3 and standard deviation 0.05, and bounded in $[0, 1]$. D_{max} and the starting positions are set to be the same as the real-world data.

4.2 Experimental Settings

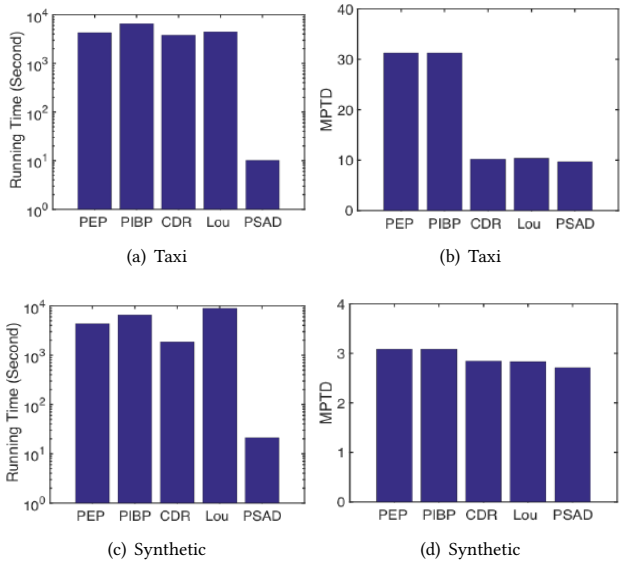
Benchmark Methods. We compare our two methods with five benchmarks which include one serial and four parallel algorithms as listed in Table 1. The two methods we propose in this paper are bold.

Table 1: Benchmark Methods for Comparison

Abbr.	Methods
SARS	SA Random Search [24]
PEP	Parallel Enumerative Process [9]
PIBP	Parallel IBP [10]
CDR	Chu et al. 1999 [3]
Lou	Lou et al. 2016 [14]
PSAD	Parallel Simulated Annealing Domain Decomposition
PSAD-M	PSAD by Mixing

For solving the MSR problem, Ge et al. [9] and Huang et al. [10] proposed several methods that we replicate as benchmarks. The Enumerative process represents the best case in the algorithms in [9], and IBP is the main algorithm in [10].

Parameter Setting. For fair comparison, we carefully determine parameters of all benchmark methods following the suggestions from related work and our own experiments. The method SARS and its parameter set were proposed in [24]. We set the cooling rate $\alpha = (1 - 10^{-6})^{\sqrt{1/K}}$ so that the temperature cools down slower when the number of routes K is larger. For PEP and PIBP, the only



(a) Taxi (b) Taxi

(c) Synthetic (d) Synthetic

Figure 3: Parallel performance of different methods.

parameter is the size of the subdomain. The larger the size, the better the MPTD they can obtain but with a higher computational cost. We set the size to be 17 since any larger sizes can lead to computationally prohibitive cases, which will be shown in later sections. CDR and Lou are two parallel SA frameworks with their own strategies of mixing pattern and mixing period. They can be directly applied to SARS for parallelization. For CDR, its mixing period is a fixed number, while we test 1 to 10 and choose 5 for achieving a near optimal solution without too much communication burden. For Lou, we follow an adaptive way proposed in their work that can automatically determine the mixing period. For PSAD, $step_{in}$ and $step_{ro}$ are chosen as 10 and $step_{ro}$ is carefully analyzed in Section 4.6. For PSAD-M, we set $step_{neigh}$ to be 100 and discuss the performance in Section 4.6. For the parallel method with P_c processes, its cooling rate is P_c times faster so as to accelerate the convergence. Note that PEP and PIBP are deterministic algorithms while the others are stochastic. We conduct 10 rounds of independent experiments and report the average for all stochastic methods for obtaining reliable overall performance.

Experimental Environment All experiments are conducted on the Stony Brook Institute for Advanced Computational Science's cluster with 100 nodes, each with two Intel Xeon E5-2690v3 12 core CPUs and 128 GB DDR4 Memory. Parallel experiments are conducted with the core (or process) number of multiples of 12.

4.3 Comparison of Parallel Methods

Now we discuss the results for computing time and MPTD for PEP, PIBP, CDR, Lou, and PSAD, with $P_c = 96$ based on both BJ and synthetic datasets. Figure 3 shows that our PSAD takes about 10 to 20 seconds to finish searching for the Taxi and synthetic data, two orders of magnitude fast than the others taking more than 1,000 seconds, and it also generates the lowest MPTD. Note that, the MPTD obtained from the synthetic dataset is generally lower than the Taxi dataset. The reason is that the synthetic dataset contains more pick-up points than our Taxi dataset. To sum up, our method consistently outperforms all benchmarks in terms of the computational efficiency as well as the quality of recommendation measured by MPTD.

4.4 Speedup for PSAD

In this section, we investigate the parallel performance of our method (PSAD) and two benchmark parallel techniques (CDR and Lou). The metric of speedup we use is defined as follows.

$$Speedup = \frac{t_{serial}}{t_{parallel}}, \tag{11}$$

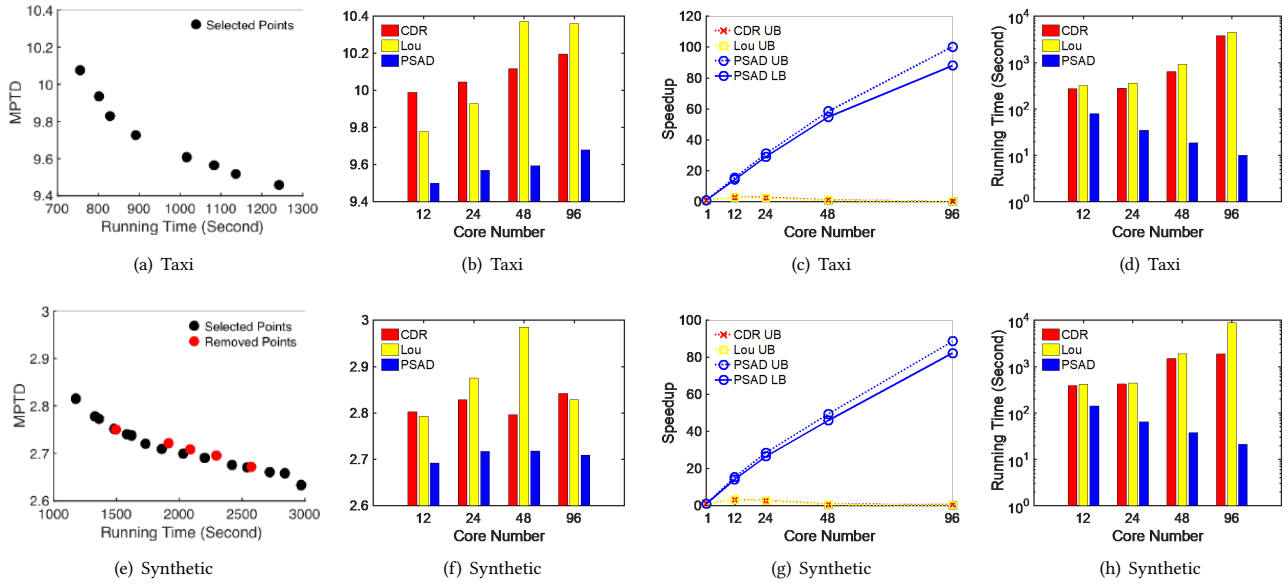
where t_{serial} and $t_{parallel}$ are the serial and parallel timing results respectively. In our experiments, t_{serial} is the running time from SARS and $t_{parallel}$ is the running time from CDR, Lou, PSAD, or PSAD-M.

SA has an important parameter, the cooling rate $\alpha \in (0, 1)$ that balances the computational efforts and the quality of the results. When α is closer to 0, the SA run takes less time but locates a solution with worse quality and vice versa. SARS is the SA framework with random search as a move generation and thus inherits this property. We are able to learn the running time that SA needs for each MPTD level by varying α. For different parallel methods, each method may find solutions in different MPTD levels. This technique is important for obtaining t_{serial} for each parallel method, because different parallel methods may result in different $MPTP$ levels.

In Figure 4 (a) and (e), we demonstrate the average running time and MPTD of SARS with different values of α in 10 runs. The red points lie in the upper right of the others indicate that SARS with these values of α would result in larger MPTD and longer running time than SARS with other values of α. These values of α are removed for obtaining the best performed SA for all parallel techniques we compare. That is, the parallel methods are compared based on the optimal serial method, SARS.

As shown in Figure 4 (b) and (f), we run CDR, Lou, and PSGS based on 12, 24, 48, and 96 cores, and we record the MPTD they obtain. Based on Figure 4 (a) and (e), we can calculate the upper bound and lower bound of t_{serial}. Note that, parallel method a with n cores achieves MPTD at a value of $MPTD_{an}$. In Figure 4 (a) and (e), let s_1 to be the lowest point that is higher than $MPTD_{an}$ and s_2 to be the highest point that is lower than $MPTD_{an}$. If we use the t_{serial} of s_1 to calculate the speedup, it is too strict since we are comparing the timing results of the parallel method to a serial method with worse solution quality. Because the longer the running time, the better the solution quality, the t_{serial} is smaller than it should be. According to Eq. (11), the speedup of the parallel method is smaller than the actual case, and we use it as the lower bound of the speedup. On the opposite, s_2 obtains better solution than the parallel method, leading to the upper bound of the speedup. We now obtain an objective way to estimate the speedup with its lower and upper bounds.

Figure 4 (c) and (g) plot the speedup of our PSAD and the benchmarks. We only show the upper bound of the speedup for CDR, Lou, while providing both upper and lower bounds of the speedup for PSAD. As can be seen, even looking at the lower bound, our PSAD significantly outperforms CDR and Lou. For instance, in the

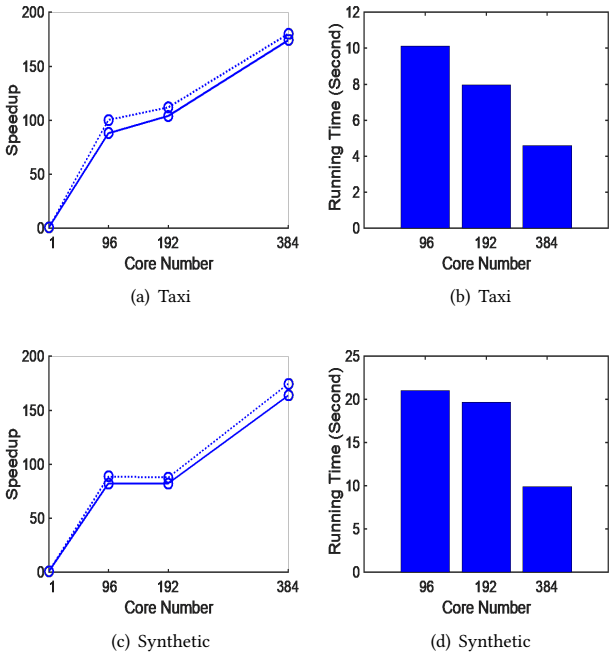


(a) Taxi (b) Taxi (c) Taxi (d) Taxi

(e) Synthetic (f) Synthetic (g) Synthetic (h) Synthetic

Figure 4: Parallel performance of CDR, Lou, and PSAD with different number of cores.

In (a) and (e), the points are the SARS runs with cooling rate $\alpha = (1 - 10^{-6})^{\frac{1}{i}}$ where i is the point from the left to the right. The comparisons of achieved *MPTD* are plotted in (b) and (f) for the taxi data and synthetic one respectively. In (c) and (g), the dash line indicates the upper bound and the solid line is for the lower bound. The red line and the green line are overlapped. The comparisons of computational efficiency are shown in (d) and (h) respectively. Note that the results we report are all based on the average of 10 independent experiments.



(a) Taxi (b) Taxi

(c) Synthetic (d) Synthetic

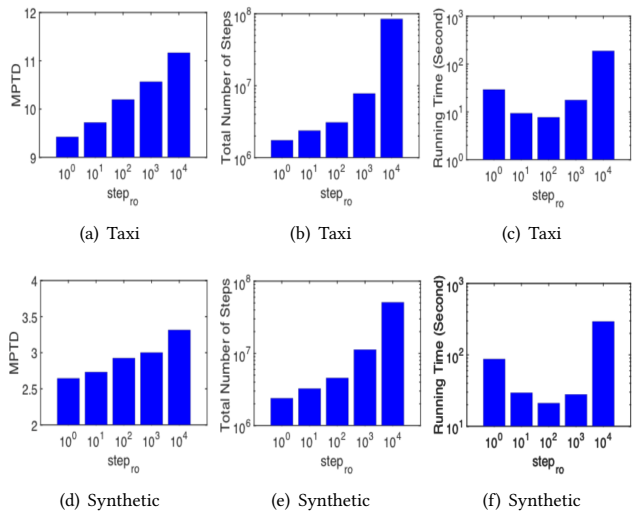
Figure 5: Parallel performance of PSAD-M. Dash lines and solid lines represent lower and upper bounds of the speedup.

Taxi dataset, when the number of cores is set to 96, the lower bound and the upper bound of the speedup of PSAD are 88 and 100 respectively, while CDR and Lou can only reach 2.9 and 2.7. For the synthetic dataset, the speedup is between 82 and 88, while CDR and Lou are lower than 3.3 and 3.2. We finally show the running time of the parallel methods with different numbers of cores in Figure 4 (d) and (h). Combined with Figure 4 (b) and (f), we can learn that PSAD achieves a lower MPTD in a much smaller amount of running time. Together with the lower bound in Figure 4 (c) and (g), we obtain the corresponding timing results of SARS. For the Taxi dataset, our PSAD spends only 10.1 seconds for locating 96 routes while it takes SARS 891 seconds to locate a worse one than our method. Similarly, for the synthetic dataset, PSAD uses 21.0 seconds while SARS uses 1,722 seconds for a worse solution.

All results show that our PSAD consistently outperforms the other parallel methods and can accelerate the serial performance significantly.

4.5 Further Speedup for PSAD-M

We discuss the speed up performance of our second method, PSAD-M. With the limited computing resources, the maximum parallel size that we can access is 384. Based on our experimental settings, our task is to simultaneously locate 96 routes. Therefore, we can assign up to 4 cores into a group for each route searching. As we discussed in Section 4.4, we can calculate the lower bound and upper bounds of the speedup of a parallel method. We plot both bounds of PSAD-M in Figure 5 (a) and (c). PSAD can be considered



(a) Taxi (b) Taxi (c) Taxi

(d) Synthetic (e) Synthetic (f) Synthetic

Figure 6: Parameter selection based on the parallel performance of PSAD.

as a simple case of PSAD-M when the number of cores is 96 where each core itself is a group. As can be seen, the speedup still increases as the number of cores increases. When the number of cores is 384, the speedup of PSAD-M is between 174x and 180x for the Taxi dataset, and between 163 and 176 for the synthetic dataset. Using 384 cores, PSAD-M doubles the speedup obtained from PSAD. For running time, according to Figure 5 (b) and (d), PSAD-M spends 4.6 seconds for the Taxi dataset and 9.9 seconds for the synthetic dataset for locating 96 high-quality routes. The results confirm the consistent improvement in speedup with our PSAD-M, compared to the performance of PSAD.

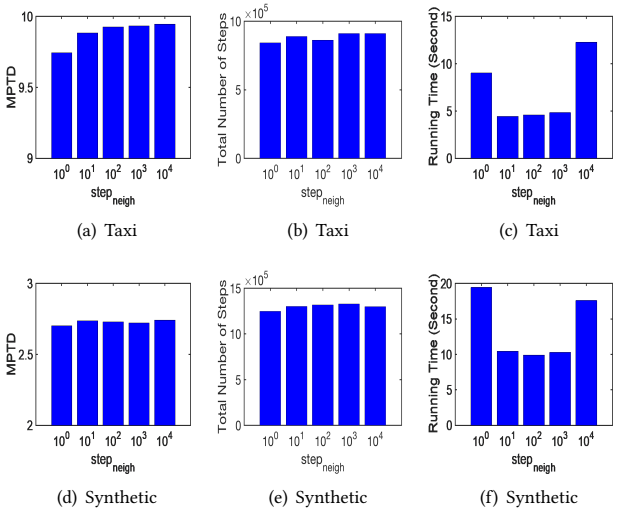
4.6 Parameter Analysis

In this section, we analyze two parameters, $step_{ro}$ of PSAD and $step_{neigh}$ of PSAD-M for Taxi and synthetic datasets.

For $step_{ro}$, we test five cases from $step_{ro}$ being 10^0 to 10^4. As shown in Figure 6 (a) - (b) and (d) - (e), the MPTD and total computational steps grow with an increasing $step_{ro}$. This phenomenon is understandable. For $step_{ro} = 1$, the routes keep rotating to different cores with their pick-up point sets in every step. The rotation is too frequent, so that the algorithm becomes similar to the serial case. The PSAD imitates the serial procedure almost perfectly and therefore is able to achieve a low MPTD with a small amount of computing steps. On the other hand, with a large $step_{ro}$, after $step_{ro}$ steps, the route in each core has already approached a local minimum based on the current pick-up point set. Considering that the current route is nearly the local minimum, it can hardly be affected by the other pick-up point sets for changing to a global minimum solution. Thus, we conclude that a larger $step_{ro}$ will result in a higher MPTD. Also, assume that the pick-up point c_i in core i can improve the route in core j with $i < j$. Then it takes c_i about $(j - i)step_{ro}$ to reach core j, leading to the large increase of the total number of steps if $step_{ro}$ is large.

In Figure 6 (d) and (f), the running time is high for small $step_{ro}$ due to large communication time from frequent rotations. It is high for large $step_{ro}$ due to large computation efforts. The $step_{ro}$ that lies between 10^1 and 10^3 can balance the computation and communication cost. $step_{ro}$ is an essential parameter in PSAD because it determines the frequency for rotation, interaction, and shuffling. Our results show that it is not sensitive in $[10^1 10^3]$, which is a large range indicating the flexibility of the determination of $step_{ro}$. Also, it has a consistent pattern for both Taxi and synthetic dataset. Therefore, there will be little parameter tuning process if one wants to adopt PSAD to a new dataset.

When adding the second parallel technique to PSAD to form PSAD-M, we need to tune an additional parameter $step_{neigh}$. As can be seen from Figure 7, when $step_{neigh}$ is less than 10^4, it will neither affect the quality of the results nor the total number of steps. However, it will affect the communication time resulting in the changes in running time. When $step_{neigh} = 1$, the neighborhood of the relative best route is very small. The cores need frequent communication to stay close to each other resulting in high running time. When $step_{neigh}$ is a large number, in each mixing, it is a common case that only one route is synchronized while others need to wait. Waiting time among the cores increases the running time. It explains the phenomenon in Figure 7. Like $step_{ro}$, $step_{neigh}$ behaves consistently in the two datasets and has a reasonably large suitable interval $[10^1, 10^3]$ for parameter selection.



(a) Taxi (b) Taxi (c) Taxi

(d) Synthetic (e) Synthetic (f) Synthetic

Figure 7: Parameter selection based on the parallel performance of PSAD-M.

5 RELATED WORK

We summarize the related work into two categories: human trajectory mining and parallel simulated annealing.

5.1 Human Trajectory Mining

Different from research based on traditional urban data that focuses on characterizing the market dynamics [6, 7], recent studies on data mining topics seek efficient methods to support dynamic decision

making for mobile users. The MSR problem, as one of the popular problems in recent urban computing research, has been widely discussed and studied [10, 18, 19].

Efforts have also been taken for developing other applications using human trajectory data. Ge et al. [8] developed a taxi business intelligence system for reducing inefficiency in energy consumption and improving customer experience and business performance. Powell et al. [18] proposed novel methods in identifying profitable locations for targeted taxicab based on its current location and time by making use of the historical pick-up points. Qu et al. [19] developed the net profit calculation by taking the gas, traffic, and opportunity cost into consideration and designed a recursive recommendation strategy for the taxicabs to achieve the most profitable route. Yuan et al. [26] proposed a partition-and-group framework to achieve on-line recommendation based on route segments. The recommendation could also be provided for the passengers to balance to demand and supply and targeted at the off-peak hours. Yuan et al. [27] also provided an algorithm to search for the parking places and built up a model that facilitates the recommendation in a whole city. Wang et al. [22] adapted artificial neural network to recognize the high passenger-finding potential road clusters for the taxicabs. Ma et al. [15] studied the taxis in ride-sharing and provided algorithms for taxi searching and scheduling to reduce the total travel distance. Liu et al. [13] designed a workflow modeling framework for healthcare operation and management based on indoor trajectory data. More work that uses human trajectory data can be found in [12, 16, 23, 28, 29].

5.2 Parallel Simulated Annealing

SA was initially proposed by Kirkpatrick et al. [11] and has been widely used for solving combinatorial problems. It is able to locate a high-quality solution but with a significantly large amount of time. Efforts are then spent on parallelizing SA to accelerate the calculation. Due to the serial nature of SA, the computing cores have to frequently communicate with each other. As a result, although the computation time can be reduced in [3], the running time is large because of the significant increase of the communication time [14]. According to [14], among the literature on parallel SA, for the parallel size of 32 or larger, the best speedup record was 19.6x for the TSP [20]. Other work in parallel SA with a parallel size larger than 32 either did not report the actual speedup or failed to conclude the preservation of the quality of their solutions. [17]. Efforts are also spent in combining the parallel SA by domain decomposition. However, the largest speedup reported was 14.1x [5].

To get the inspiration of parallelizing SA on MMSR problem, we try to relate the MMSR problem to the problems in classic operational research and look for the performance when parallel SA is applied to those problems. The MSR problem shares some similarities with the classic traveling salesperson problem [1] as they both recommend a sequence of locations but with relatively different size of location set. The set is larger than the sequence in MSR while in TSP the set and the sequence are the same. The vehicle routing problem (VRP) [4, 21] generalizes the TSP to a fleet of vehicles, which is similar to the MMSR problem that generalizes the MSR problem to multiple routes. The largest speedup for the VRP based on parallel SA was less than 10x [2].

6 CONCLUSIONS

We formalize a new multi-user mobile sequential recommendation (MMSR) problem to generalize the MSR problem. Our MMSR problem aims to simultaneously recommend optimal routes to multiple users (e.g. taxi drivers) with different locations. The first method we propose, PSAD, can efficiently handle high-dimensional MMSR problems and significantly outperforms all existing benchmarks. An improved version of PSAD, named PSAD-M, is also developed to enhance the usage of computing cores for achieving further speedup. The improvement is confirmed by our results. Moreover, we show the stability of our methods under different parameter settings and conclude the reliability of the methods when applying to different datasets.

ACKNOWLEDGMENTS

Keli Xiao acknowledges the support of the National Natural Science Foundation of China (NSFC) (Grant # 91746109).

REFERENCES

[1] David L. Applegate, Robert E. Bixby, Vasek Chvatal, and William J. Cook. 2011. *The traveling salesman problem: a computational study.* Princeton university press.
[2] RaúL BañOs, Julio Ortega, Consolación Gil, Antonio FernáNdez, and Francisco De Toro. 2013. A simulated annealing-based parallel multi-objective approach to vehicle routing problems with time windows. *Expert Systems with Applications* 40, 5 (2013), 1696–1707.
[3] King-Wai Chu, Yuefan Deng, and John Reinitz. 1999. Parallel simulated annealing by mixing of states. *J. Comput. Phys.* 148, 2 (1999), 646–662.
[4] George B Dantzig and John H Ramser. 1959. The truck dispatching problem. *Management Science* 6, 1 (1959), 80–91.
[5] Frederica Darema, Scott Kirkpatrick, and V. Alan Norton. 1987. Parallel algorithms for chip placement by simulated annealing. *IBM Journal of Research and Development* 31, 3 (1987), 391–402.
[6] Frank J Fabozzi and Keli Xiao. 2017. Explosive rents: The real estate market dynamics in exuberance. *The Quarterly Review of Economics and Finance* 66 (2017), 100–107.
[7] Frank J Fabozzi and Keli Xiao. 2018. The Timeline Estimation of Bubbles: The Case of Real Estate. *Real Estate Economics* (2018). https://doi.org/10.1111/1540-6229.12246
[8] Yong Ge, Chuanren Liu, Hui Xiong, and Jian Chen. 2011. A taxi business intelligence system. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 735–738.
[9] Yong Ge, Hui Xiong, Alexander Tuzhilin, Keli Xiao, Marco Gruteser, and Michael Pazzani. 2010. An energy-efficient mobile recommender system. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*. 899–908.
[10] Jianbin Huang, Xuejun Huangfu, Heli Sun, Hui Li, Peixiang Zhao, Hong Cheng, and Qinbao Song. 2015. Backward path growth for efficient mobile sequential recommendation. *IEEE Transactions on Knowledge and Data Engineering* 27, 1 (2015), 46–60.
[11] Scott Kirkpatrick, C Daniel Gelatt, and Mario P Vecchi. 1983. Optimization by simulated annealing. *Science* 220 (1983), 671–680.
[12] Chuanren Liu, Yong Ge, Hui Xiong, Keli Xiao, Wei Geng, and Matt Perkins. 2014. Proactive workflow modeling by stochastic processes with application to healthcare operation and management. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 1593–1602.
[13] Chuanren Liu, Hui Xiong, Spiros Papadimitriou, Yong Ge, and Keli Xiao. 2017. A proactive workflow model for healthcare operation and management. *IEEE transactions on knowledge and data engineering* 29, 3 (2017), 586–598.
[14] Zhihao Lou and John Reinitz. 2016. Parallel simulated annealing using an adaptive resampling interval. *Parallel computing* 53 (2016), 23–31.
[15] Shuo Ma, Yu Zheng, and Ouri Wolfson. 2013. T-share: A large-scale dynamic taxi ridesharing service. In *The 2013 IEEE 29th International Conference on Data Engineering (ICDE)*. 410–421.
[16] Shuo Ma, Yu Zheng, and Ouri Wolfson. 2015. Real-time city-scale taxi ridesharing. *IEEE Transactions on Knowledge and Data Engineering* 27, 7 (2015), 1782–1795.
[17] Samir W Mahfoud and David E Goldberg. 1995. Parallel recombinative simulated annealing: a genetic algorithm. *Parallel computing* 21, 1 (1995), 1–28.

[18] Jason W Powell, Yan Huang, Favyen Bastani, and Minhe Ji. 2011. Towards reducing taxicab cruising time using spatio-temporal profitability maps. In *International Symposium on Spatial and Temporal Databases*. Springer, 242–260.
[19] Meng Qu, Hengshu Zhu, Junming Liu, Guannan Liu, and Hui Xiong. 2014. A cost-effective recommender system for taxi drivers. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. 45–54.
[20] Andrew Sohn. 1996. Generalized speculative computation of parallel simulated annealing. *Annals of Operations Research* 63, 1 (1996), 29–55.
[21] Paolo Toth and Daniele Vigo. 2002. *The vehicle routing problem*. SIAM.
[22] Ran Wang, Chi-Yin Chow, Yan Lyu, Victor CS Lee, Sam Kwong, Yanhua Li, and Jia Zeng. 2018. Taxirec: recommending road clusters to taxi drivers using ranking-based extreme learning machines. *IEEE Transactions on Knowledge and Data Engineering* 30, 3 (2018), 585–598.
[23] Keli Xiao, Qi Liu, Chuanren Liu, and Hui Xiong. 2018. Price Shock Detection With an Influence-Based Model of Social Attention. *ACM Transactions on Management Information Systems* 9, 1 (2018), 2.
[24] Zeyang Ye, Keli Xiao, and Yuefan Deng. 2015. Investigation of Simulated Annealing Cooling Schedule for Mobile Recommendations. In *Proceedings of the 2015 IEEE International Conference on Data Mining Workshop (ICDMW)*. IEEE, 1078–1084.
[25] Zeyang Ye, Keli Xiao, Yong Ge, and Yuefan Deng. 2018. Applying Simulated Annealing and Parallel Computing to the Mobile Sequential Recommendation. *IEEE Transactions on Knowledge and Data Engineering* (2018). https://doi.org/10.1109/TKDE.2018.2827047
[26] Jing Yuan, Yu Zheng, Liuhang Zhang, Xing Xie, and Guangzhong Sun. 2011. Where to find my next passenger. In *Proceedings of the 13th international conference on Ubiquitous computing*. 109–118.
[27] Nicholas Jing Yuan, Yu Zheng, Liuhang Zhang, and Xing Xie. 2013. T-finder: A recommender system for finding passengers and vacant taxis. *IEEE Transactions on Knowledge and Data Engineering* 25, 10 (2013), 2390–2403.
[28] Li Zhang, Keli Xiao, Qi Liu, Yefan Tao, and Yuefan Deng. 2015. Modeling social attention for stock analysis: An influence propagation perspective. In *Proceedings of the 2015 IEEE International Conference on Data Mining (ICDM)*. IEEE, 609–618.
[29] Hengshu Zhu, Enhong Chen, Kuifei Yu, Huanhuan Cao, Hui Xiong, and Jilei Tian. 2012. Mining personal context-aware preferences for mobile users. In *Proceedings of the 2012 IEEE International Conference on Data Mining (ICDM)*. IEEE, 1212–1217.