Multi-User Mobile Sequential Recommendation for Route Optimization

KELI XIAO, ZEYANG YE, and LIHAO ZHANG, State University of New York at Stony Brook
WENJUN ZHOU, University of Tennessee Knoxville
YONG GE, University of Arizona
YUEFAN DENG, State University of New York at Stony Brook

We enhance the mobile sequential recommendation (MSR) model and address some critical issues in existing formulations by proposing three new forms of the MSR from a multi-user perspective. The multi-user MSR (MMSR) model searches optimal routes for multiple drivers at different locations while disallowing overlapping routes to be recommended. To enrich the properties of pick-up points in the problem formulation, we additionally consider the pick-up capacity as an important feature, leading to the following two modified forms of the MMSR: MMSR-m and MMSR-d. The MMSR-m sets a maximum pick-up capacity for all urban areas, while the MMSR-d allows the pick-up capacity to vary at different locations. We develop a parallel framework based on the simulated annealing to numerically solve the MMSR problem series. Also, a push-point method is introduced to improve our algorithms further for the MMSR-m and the MMSR-d, which can handle the route optimization in more practical ways. Our results on both real-world and synthetic data confirmed the superiority of our problem formulation and solutions under more demanding practical scenarios over several published benchmarks.

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

Additional Key Words and Phrases: Mobile sequential recommendation, trajectory data analysis, parallel computing, simulated annealing, potential traveling distance

ACM Reference format:
Keli Xiao, Zeyang Ye, Lihao Zhang, Wenjun Zhou, Yong Ge, and Yuefan Deng. 2020. Multi-User Mobile Sequential Recommendation for Route Optimization. *ACM Trans. Knowl. Discov. Data* 14, 5, Article 52 (July 2020), 28 pages.
https://doi.org/10.1145/3360048

K. Xiao and Z. Ye contributed equally to this article.

This work was partially supported by the National Social Science Foundation of China (# 18BFX096) and the National Natural Science Foundation of China (# 91746109).

Authors' addresses: K. Xiao (corresponding author), College of Business, Stony Brook University, 100 Nicolls Rd, Stony Brook, NY 11794; email: keli.xiao@stonybrook.edu; Z. Ye, L. Zhang, and Y. Deng, Department of Applied Mathematics and Statistics, Stony Brook University, 100 Nicolls Rd, Stony Brook, NY 11794; emails: zeyang.ye@hotmail.com, {lihao.zhang, yuefan.deng}@stonybrook.edu; W. Zhou, Department of Business Analytics and Statistics, Haslam College of Business, University of Tennessee Knoxville, 916 Volunteer Blvd., Knoxville, TN 37996-0532; email: wzhou4@utk.edu; Y. Ge, Department of Management Information Systems, Eller College of Management, University of Arizona, 1130 E Helen St, Tucson, AZ 85721; email: yongge@email.arizona.edu.

1556-4681/2020/07-ART52 $15.00
https://doi.org/10.1145/3360048

1 INTRODUCTION

Characteristics of human mobility can be extracted from extensive mobile sequential data (e.g., Global Positioning System (GPS) trajectories and points of interest (POI) check-in sequence), and they can be utilized further in numerous recommendation problems. The *mobile sequential recommendation* (MSR) is one of the earliest applications in mobile recommender systems based on GPS trajectories, followed by various valuable studies in a broad range of topics, including POI recommendation [15], travel package recommendation [18], ride-sharing [20], and the like. In this article, we focus on the MSR for route optimization.

A well-known MSR problem formalized by Ge et al. [11] is to provide route recommendation to a taxi driver with an objective of minimizing the potential traveling distance (PTD) before picking up the next passenger. While some route optimization problems are based on on-demand settings (e.g., the first-mile ride-sharing [4]), the MSR problem aims to achieve route optimization for cases without sharing or vehicle requests. For efficiently address the MSR problem, Huang et al. [12] proposed a backward path-growth method to speed up the route searching process, the method is still not efficient enough to handle high-dimensional MSR problems directly due to the costly offline training process. Ye et al. [34] developed a stochastic method without any offline process based on simulated annealing and substantially reduced the computational time for high-dimensional MSR problems. However, the original form of MSR has not been sufficiently discussed after being proposed. Qu et al. [25] modified the objective function of MSR by considering the cost effect, while the modified objective function is essentially equivalent to the original MSR as suggested by [33]. A major issue of the MSR is that it can only lead to effective solutions for a single user, while complex situations under multi-user scenarios are not considered.

Developing the MSR problem and related algorithms are valuable to both individuals and the human society. In addition to satisfying individual users by providing them appropriate and precise route recommendations, an efficient and effective traffic recommendation system would substantially reduce various types of social costs, including labors, energy, air pollution, and travel time. With the advent of the autonomous driving and sharing economy, considering the MSR problem from the multi-user perspective will be an extremely important problem that affects people's everyday life. The autonomous cars seeking passengers can be viewed as a general case of taxicabs in the MSR problem. Thus, efficiently solving the multi-user MSR (MMSR) problem can not only benefit today's urban traffic system but also will lead to significant innovations on path planning for autonomous driving in the near future.

However, designing search algorithms for MMSR problems is not easy, and major challenges are threefold. First, the search algorithms must handle multi-user queries and recommend distinct routes for different drivers. Although Huang et al. [12] 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, in addition to the information of location and pick-up probability, another important feature must be considered is the pick-up capacity. That is, locations with more waiting passengers must be allowed to be visited more often by drivers. Last, the search algorithm should be efficient enough for volatile traffic dynamics. In existing methods, a two-stage framework containing an offline pruning and an online searching procedure is usually adopted [11, 12]. However, the extremely high computational cost of the offline pruning process indicates these two-stage approaches will still fail in handling high-dimensional MSR problems.

To address the aforementioned issues, (i) we formalize a new MMSR problem that aims to recommend different routes to users such that the duplication of the pick-up locations in the recommended routes should be avoided; (ii) we improve the MMSR by formalizing MMSR-m and

MMSR-d, which consider an important feature of pick-up points, the *pick-up capacity*, to characterize the popularity and demand for each location; and (iii) we propose a parallel framework to efficiently address high-dimensional MMSR problems. Some of above innovations have been documented in an early phase work [35], including the formulation of the original MMSR problem and the development of a parallel simulated annealing method with domain decomposition (PSAD) as well as its enhanced version PSAD-M. We summarize them 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, which significantly outperforms 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 simulated annealing (SA) is adapted for use in other optimization problems, our framework maximizes the strength of parallel SA in the MMSR problem.

In addition, this work differs from [35] and contributes to the literature further in four ways.

- —We propose two modified forms of the MMSR problem, named, MMSR-m and MMSR-d, as improved versions of the MMSR, which take the pick-up capacity into consideration.
- —We discover an important property that can be used in problem settings to reduce the computational cost of MMSR-m and MMSR-d. We theoretically prove the property and verify it in our experiments.
- —A new *push-point* algorithm is developed and then embedded to the PSAD-M method for handling the influence of the pick-up capacity on the searching process for optimal routes. Related results further confirm the effectiveness of PSAD-M and its expandability.
- —We conduct comprehensive experiments and demonstrate the superiority of MMSR-d over other forms of MMSR as well as the original MSR problem.

2 PROBLEM FORMULATION AND IMPROVEMENTS

This section introduces the original MSR problem and then formalizes the new MMSR problem and two improved forms (MMSR-m and MMSR-d). The improvements achieved from the new forms of MMSR are discussed as well. We summarize important notations in Table 1.

2.1 The MSR Problem

The original format of the MSR problem [11] has the following settings. Suppose that we are given a set of N potential pick-up points $C = \{c_1, c_2, \ldots, c_N\}$, the current position of the taxi driver c_0, and the set of the successful pick-up probabilities corresponding to each potential pick-up point $P = \{p(c_1), p(c_2), \ldots, p(c_N)\}$. From the perspective of practice, P should be updated frequently to represent dynamic traffic status, and our recommendation system should be able to handle the frequent traffic changes.

Given the route length L, we have a sequence of pick-up points $\overrightarrow{r_i} = (c_{i_1}, c_{i_2}, \ldots, c_{i_L})$ representing a route. Then, there are in total $M = \binom{N}{L}L!$ sequences in C. $R = \left\{\overrightarrow{r_1}, \overrightarrow{r_2}, \ldots, \overrightarrow{r_M}\right\}$ represents the set of all possible routes through the potential pick-up points. The original form of MSR problem is defined as follows:

Problem 1 (MSR). *Given a single driver at position c_0 and the set of all possible routes R, the objective of the MSR problem is to search for the traveling route $\overrightarrow{r_i} \in R$ with the minimal PTD before*

Table 1. Important Notations and Definitions

Notations	Descriptions
C	The set of pick-up points
P	The set of pick-up probabilities
R	A set of routes
C_0	A set of starting locations
N	The total number of pick-up points
M	The total number of pick-up sequences
L	Route length
K	The total number of routes
c	A pick-up point
pc	The pick-up point capacity
pc_l	The pick-up point capacity of a pick-up point l
c_0	The current position of a taxi
$p(c)$	The pick-up probability of a pick-up point c
$\overrightarrow{r}$	A route
$d(c_i, c_j)$	The traveling distance between c_i and c_j
d_∞	The penalty term
$PTD(c_0, \overrightarrow{r})$	The potential traveling distance of route $\overrightarrow{r}$ with starting location c_0
$PTD_k\ (C_0, R_i)$	The potential traveling distance of routes in R_i with location set C_0

the next successful pick-up. That is,

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

There are two forms of the PTD function, including the original form in [11] and the transformed form in [33]. The two forms of PTD have been proved to be equivalent [33], while we use the latter for its advantages of interpretability. The PTD function can be written as follows:

$$PTD\left(c_0, \overrightarrow{r_i}\right) = d\left(c_0, c_{i_1}\right) + \sum_{j=2}^{L}\left[d\left(c_{i_{j-1}}, c_{i_j}\right) \cdot \prod_{k=1}^{j-1} \overline{p\left(c_{i_k}\right)}\right] + d_\infty \cdot \prod_{j=1}^{L} \overline{p\left(c_{i_j}\right)}, \tag{2}$$

where $\overline{p(c_{i_j})} = 1 - p(c_{i_j})$; $d(c_{i_{j-1}}, c_{i_j})$ represents the distance between $c_{i_{j-1}}$ and c_{i_j}. Note that d_∞ is a penalty term, which can be viewed as the extra distance a driver needs to travel if (s)he fails to find a passenger along the route.[1]

As can be seen in Problem 1, the MSR problem is designed from the view of a single driver without considering the state of others. In this case, if K drivers are asking for recommendations at the same time, the MSR system will process K independent recommendations. However, the K recommended routes may overlap, especially when those drivers are close to each other. A good route recommendation system should avoid duplicated recommendations, because they may lead to serious traffic issues. Therefore, we seek an improved version of the MSR problem, which has an objective function designed for multiple users.

[1]The original form of PTD proposed in [11] has a penalty term as well, say D_∞. According to [33], D_∞ in the original PTD and d_∞ in (2) are interconvertible; $D_\infty = d(c_0, c_{i_1}) + \sum_{j=2}^{L} d(c_{i_{j-1}}, c_{i_j}) + d_\infty$.

2.2 The MMSR Problem Series

2.2.1 The MMSR Problem. To address the issues of the single user-based MSR problem, a new MMSR problem [35] can be formalized as follows.

Problem 2 (MMSR). *Given K drivers requesting sequential recommendations, the set of positions of the these drivers C_0, the objective of the MMSR problem is to recommend K routes for the K drivers, and the multi-user potential traveling distance (MPTD) should be minimized. That is,*

$$\begin{aligned} &\min_{R_i \subseteq R} PTD_K(C_0, R_i), \\ &s.\ t.\ \overrightarrow{r_{i_a}} \cap \overrightarrow{r_{i_b}} = \emptyset, \forall a, b \in [1, K], a \neq b, \end{aligned} \tag{3}$$

where $R_i = \{\overrightarrow{r_{i_1}}, \overrightarrow{r_{i_2}}, \ldots, \overrightarrow{r_{i_k}}\}$; $PTD_K(C_0, R_i)$ represents the MPTD, which is computed as:

$$PTD_K(C_0, R_i) = \sum_{j=1}^{K} PTD(c_0, \overrightarrow{r_{i_j}}). \tag{4}$$

As our goal is to minimize the traveling distance for all the drivers before their next successful pick-ups, the smaller the MPTD, the better quality of the routes. In this way, we can use MPTD to evaluate the recommendation quality[2]. When $K = 1$, Equations (4) and (2) are equivalent, and the MMSR and the MSR are equivalent.

Comparing with MSR, the MMSR problem is formalized based on a more practical way but with a much higher computational cost, given its $\binom{N}{K \cdot L}(K \cdot L)!$ possible combinations. It is approximately $(N - \frac{K}{2})^{K \cdot L}$ times of the $\binom{N}{L}L!$ combinations in MSR. In MMSR, the MPTD function is the accumulation of K PTDs in Problem 1 with one additional requirement that all the routes cannot be overlapped. That is, any pick-up point can only be visited once. Such recommendation eliminates the competition among drivers and aims to optimize the overall PTD. The objective function of MMSR is clear and straightforward, but it still needs to be improved for some important issues, which we summarize as the following four defects of MMSR.

- **Defect 1**. Some pick-up points in a recommended route may never be visited even if there are passengers waiting. Let us consider the following situation. If a driver follows the recommended route and successfully gets a passenger, then (s)he would terminate the recommended route. However, the rest of the potential pick-up points in the recommended route will be skipped and will not be recommended to others. These pick-up points are blocked in the recommendation system, and passengers waiting there will not be served. Thus, we need to allow the system to recycle pick-up points if needed.
- **Defect 2**. The city may not be fully explored. In the real situation, a pick-up point is not a single point on the map but a small area. Based on the settings of MSR and MMSR, as long as a taxi driver passes by, the pick-up point is regarded as "visited." However, one driver will hardly explore all passengers in the area (e.g., someone located in a corner of a street).
- **Defect 3**. Even for a very small area, one pick-up point may contain more than one passengers, but MSR and MMSR only allow one driver to visit each pick-up point. For example, in the train station, there may be a large number of passengers. However, the system only assigns one driver to the train station, and then the rest of the passengers cannot be serviced. Thus, we should allow a pick-up point to be revisited by drivers.

[2]Note that it has been shown in [33] that an equivalent substitute of the PTD is the expected traveling time (ETT) for the drivers to get the next passenger. Following the same idea, our MPTD can also be substituted by a multiple-user version of ETT, say METT, and hence traveling time can be used to evaluate the recommendation quality as well.

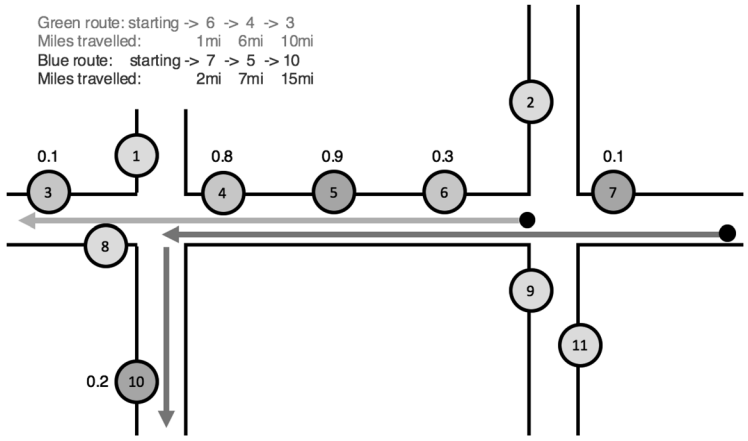


Fig. 1. The miscalculation of the MPTD due to routes overlapping. *Note*: Numbers in the circles indicate different the pick-up points. The numbers beside circles represent related pick-up probabilities. The green line indicates the green route and the pick-up points assigned to the green line are filled with green. Similarly, the blue route uses the color blue. The black dots indicate the starting locations of the routes. For ease of analysis, all the roads in this map are oneway.

—**Defect 4**. To reduce the effect of the above three defects, for MMSR, we always make the area covered by a pick-up point to be small. This design will then result in the large amount of the overlap of the trajectories, which leads to erroneous of the MPTD values.

While Defects 1–3 are easy to understand, we show an example of Defect 4 in Figure 1. In this route assignment, pick-up points "4," "5," and "6" locate in the same block. As points "4" and "5" have large pick-up probabilities, they are usually assigned to different routes so that the overall MPTD can be largely decreased. Assume that D_∞ in this case is 30 miles, the PTD for the green route is 7.6 miles and for blue route is 8.3 miles. The MPTD for these two routes should be the sum of the PTDs, 15.9 miles. However, in the real case, a taxi in the green route would always be ahead of a taxi in the blue route when visiting the points "4," "5," and "6" as its starting location is closer. Although point "5" is not assigned for the green route, the taxi in green will naturally visit it since it is in between points "4" and "6." After all, if the taxi driver in green finds a passenger at point "5," he would pick him/her up instead of ignoring it. Therefore, the actual pick-up sequence of the green route taxi follows ("6," "5," "4," "3"), and the resulting PTD is 3.2 miles. Based on the setting of MMSR, each pick-up point is only capable of one visit. The eligible pick-up points for the taxi in blue include point "7" and "10," and its PTD becomes 24.5 miles; the MPTD for the two routes becomes 27.7 miles. In this example, the relative error for the MPTD is 74% due to the overlapping trajectories.

In summary, the first two defects of the MMSR problem are driver-oriented, and the third and the fourth defects are pick-up point-oriented. They can be addressed by improving the objective functions, and we provide our designs in the rest of this section.

2.2.2 MMSR-m: An Improved Form. To allow multiple visits of a pick-up point, we propose a new concept, the *pick-up point capacity* (denoted by pc), which is the maximum times that a pick-up point can be visited by drivers. The initial form of MMSR sets the $pc = 1$ for all pick-up points. Now, we improve the formulation of MMSR by allowing multiple visits for each pick-up points. We first consider a special case in which all pick-up points has a capacity $pc \geq 1$, and we call the new problem as MMSR-m.

Problem 3 (MMSR-m). *Given K drivers requesting sequential recommendations, the set of positions of the these drivers C_0, the objective of the MMSR-m is to search for K routes that have the minimal MPTD based on a given pick-up point capacity before the successful pick-up of each route:*

$$\min_{R_i \subseteq R} PTD_K(C_0, R_i),$$
$$s.\ t.\ \forall R_{i_j} \subseteq R_i, \forall c_l, if\ c_l \in \bigcap_{\overrightarrow{r_i} \in R_{i_j}} \overrightarrow{r_i}\ ,\ then\ |R_{i_j}| \leq pc \tag{5}$$

where pc is the pick-up point capacity; C_0, R_i, and PTD_K follow the same settings in problem 2 (MMSR).

The MMSR-m allows a pick-up point to be recommended in at most *pc* routes, and it can be considered as a generalized form of MMSR. When $pc = 1$, the MMSR-m is equivalent to the MMSR. Let $pc > 1$, routes are allowed to overlap because one pick-up point can be recommended to more than one passenger.

The *pick-up point capacity* (*pc*) is an important parameter. By assigning each pick-up point the same capacity, we can address the first two issues of the MMSR problem discussed in Section 2.2. As nearby points with high pick-up probabilities can be visited multiple times, they are unlikely to be missed after several rounds of visits. Also, as different drivers may pass a pick-up area through different trajectories in the MMSR-m, the area should be better explored in comparison to cases in the MMSR. The settings of the MMSR-m emphasize the importance of pick-up points nearby with large pick-up probabilities. Therefore, we can expect that the *K* routes recommended based on the MMSR-m are more centralized on the map compared to the MMSR.

By allowing multiple drivers to visit the same location, the chance for a pick-up point being ignored or not fully explored decreases. So that the first two defects can be addressed. However, problems still exist because the MMSR-m does not consider the information of different pick-up probabilities. Even if all drivers follow the route assignments strictly, the third and fourth defects still exist. Therefore, we further propose our final form of the MMSR problem, named MMSR-d.

2.2.3 MMSR-d: The Final Form. Some pick-up points are more important than the others. As we mentioned in the third defect of the MMSR problem, some points not only provide high pick-up probabilities but also contain a large number of passengers. Thus, we should assign them higher capacities than those with low pick-up probabilities and small numbers of waiting passengers.

For the fourth defect, the area represented by a pick-up point should be enlarged to cover a road segment between any two road intersections. Some pick-up points that cover small areas are then grouped as a large pick-up point. The large pick-up point should be settled with a larger pick-up capacity than small ones. The third and fourth defects can be addressed by allowing pick-up points to have different pick-up capacities, and we formalized the final form of the problem (MMSR-d) as follows.

Problem 4 (MMSR-d). *Given K drivers requesting sequential recommendations, the set of positions of the these drivers C_0, the objective of the MMSR-d is to search for K routes that have the minimal MPTD based on the capacity of each pick-up point before the successful pick-up of each route:*

$$\min_{R_i \subseteq R} PTD_K(C_0, R_i),$$
$$s.\ t.\ \forall R_{i_j} \subseteq R_i, \forall c_l, if\ c_l \in \bigcap_{\overrightarrow{r_i} \in R_{i_j}} \overrightarrow{r_i}\ ,\ then\ |R_{i_j}| \leq pc_l \tag{6}$$

where pc_l is the capacity of the pick-up point c_l; C_0, R_i, and PTD_K follows the same settings in problem 2 (MMSR).

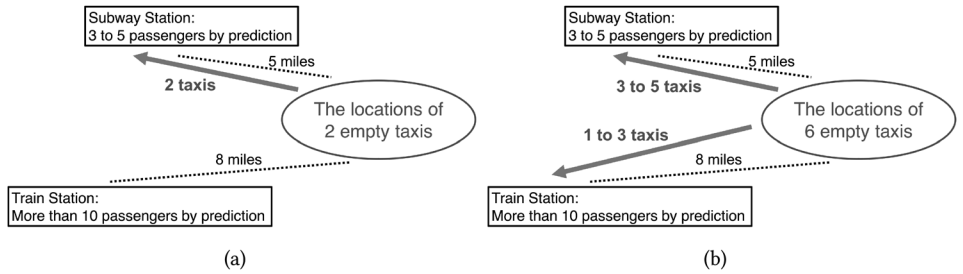


Fig. 2. The taxi number and the prediction burden.

MMSR-d requires that any pick-up point c_l should not take more than pc_l positions in a route assignment R_i. Note that, MMSR-d and MMSR-m are equivalent if pc_l is fixed and does not depend on l. Given the settings of the MMSR-d, a pick-up point that is easy to be recommended may have the following characteristics: (i) a high pick-up probability, (ii) a large pick-up capacity, and (iii) close to the location where the service request is sent.

For some special pick-up points, for example, in the train station, the pick-up capacity is large and may be even greater than K, the number of drivers requesting route recommendations. However, the solution for the MMSR-d problem will be the same when we decrease the capacity of those points to K. After all, any pick-up point can only take at most K positions in a route assignment and any large capacity cannot break this limit. This refers to the following property.

Property 2.1. *In the MMSR-d problem, given a pick-up point c_l and the number of drivers requesting route recommendation services K, and we mistakenly estimate its pick-up point capacity from pc_l to pc_{l_0}. If $pc_l, pc_{l_0} \geq K$, we should obtain the same solution of route recommendation.*

Proof. As $pc_l \geq K$ and $pc_{l_0} \geq K$, with either pick-up point capacity, the pick-up point c_l can be recommended to K routes simultaneously. Since all the possible routes to be optimized are not affected by the capacity of c_l, the solutions are the same. □

This property relieves our burden of estimating the pick-up point capacity, especially when we recommend routes to a small number of drivers. That is, we can simply set the maximum pick-up capacity to be the same as the number of drivers.

To emphasize this property, we give an example in Figure 2. In Figure 2(a), there are two drivers in the same location. They have the following two choices for the next place to visit: (i) a subway station that offers 3–5 passengers by prediction and (ii) a train station that now offers at least 10 passengers. Also, the train station is 3 miles further than the subway station. As there are only two drivers, we can recommend all of them to the subway station. However, in Figure 2(b), if we are asked to recommend routes for six drivers, we need to make the prediction for the subway station more precise. Different predictions may lead to different route recommendations. On the other hand, we do not need to care about the prediction of the train station as it offers much more passengers than we need. In the MMSR-d problem, we do not just recommend the first pick-up point to the drivers. We consider a route that contains a sequence of points, which is a more complicated task. However, because of Property 2.1, we can solve the recommendation problem much easier. We will discuss more in the next section.

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 develop a parallel framework to handle the overwhelming computational task.

3.1 PSAD: Parallelizing SA by Domain Decomposition

SA is a stochastic optimization method aiming to find the global optimization of a non-convex objective function [25]. The serial algorithm[3] 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 starts iterating. It performs the move generation by perturbing its solution with a small movement. If the new solution leads to an improvement, the current solution is updated to the new one. 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 based on a cooling schedule function. The algorithm stops if the stopping condition is met; otherwise, 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 PSAD method to speed up the searching based on the SA random search in [32]. 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: (i) SA Step, (ii) rotation, (iii) Interaction, and (iv) shuffling.

SA Step. In each step, Pr_i performs the move generation, random search in [32], 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 ensure that pick-up points have the same chance to be considered. 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 rotate routes to reduce the communication cost. When rotating, Pr_i sends its current routes to Pr_{i+1}, and Pr_{P_c} sends the routes to Pr_0. Following this way, all pick-up points in the set $\cup_i C_i$ get chances to be selected and placed in the routes within every P_c rotations.

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, the whole pick-up point set C gets chances to be selected and placed in all the routes within a period of P_c rotations and one interaction.

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, our PSAD reshuffles the pick-up points in these subsets in every P_c rotations.

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

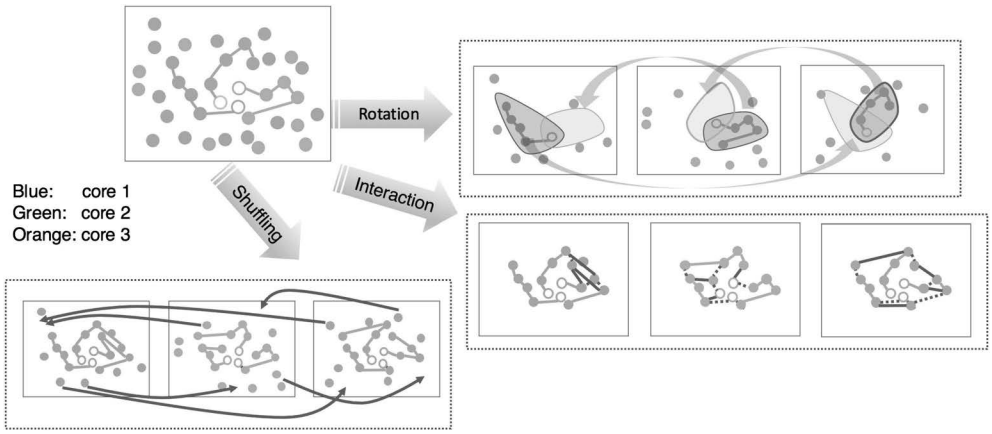


Fig. 3. Parallel operations for PSAD.

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

We demonstrate an example of aforementioned parallel procedures in Figure 3. Suppose we have three computing cores (processes), $Core1$, $Core2$, and $Core3$, to handle the MMSR problem. We use three different colors to represent assigned subsets and the current routes in consideration for the three computing processes. The empty circles are the current positions of drivers. Below we show how the PSAD works with the three-step process, rotation, interaction, and shuffling. First, in the *rotation* operation, the three cores rotate (exchange) their current routes. For example, $Core1$ sends its route to $Core3$; $Core2$ sends its route to $Core1$; and $Core3$ sends its route to $Core2$. 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 3, in all three cores, the pick-up points in two routes are switched after the interaction, leading to a reduction of the traveling distances for both routes. Last, in the *shuffling* operation, each core 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 PSAD-M: 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 times 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 times 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 no 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 1 (Distance). 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:

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

where

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

According to Definition 1, we further define the *neighborhood* as follows.

Definition 2 (Neighborhood). The neighborhood of $\overrightarrow{r_i}$ with distance $d_0 \geq 0$ is defined as:

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

Note that $\overrightarrow{r_i} \in \mathcal{N}(\overrightarrow{r_i}, d_0)$. We then design the *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 3.1. *Given the acceptance rate r, the pick-up points number n, and the route length L, the expected change in $step_{neigh}$ SA steps is computed as:*

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

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

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

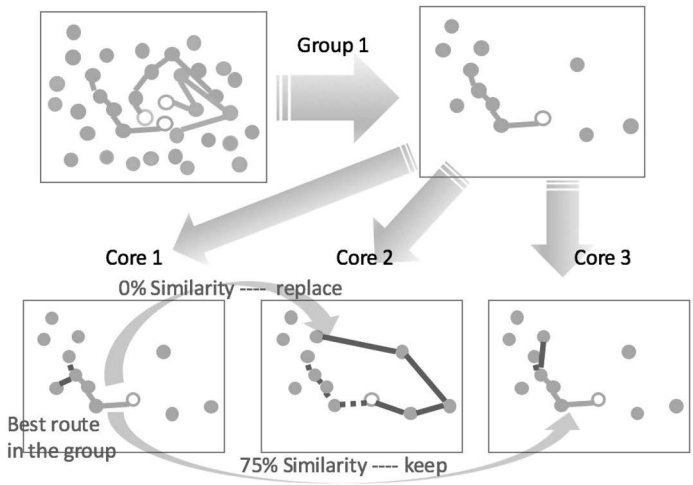


Fig. 4. The mixing procedure on PSAD-M.

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:

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

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:

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

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:

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

Finally, we get the expected change d_0 as:

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

According to Theorem 3.1, our PSAD-M can adjust the neighborhood 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 4 demonstrates an example of the mixing process in each group. Suppose three cores (processes), $Core1$, $Core2$, and $Core3$, together optimizing the same route in group 1. After the optimization procedure in the three cores, assume that the PTD from the $Core1$ is lower than other two, $Core1$ sends a copy of its route to $Core2$ and $Core3$. According to Equation (7), the similarity rate between $Core1$ and $Core2$ is 0% while the rate between $Core1$ and $Core3$ is 75%. According to Equation (10), if in previous $step_{neigh}$ SA steps, the d_0 of $Core1$ is calculated to be 0.5, then $Core3$ accepts the route copy from $Core1$ while $Core2$ rejects it. In this mixing process, $step_m$ decreases by one since $Core2$ does not lie in the neighborhood of the best route from $Core1$. Finally, after the mixing process, in Figure 4(c), $Core1$ and $Core2$ perform SA steps as the previous route from $Core1$ as their current route $Core3$ uses its own route found previously.

In summary, the key challenge that the PSAD and PSAD-M address is the high computational cost of the communication process among different cores. Our methods separate the whole set of pick-up points into several groups and assign them to different computing cores. Since each pick-up point can only appear in one group, it can appear in at most one recommended route. This setting guarantees that all recommended routes are not overlapped. Meanwhile, there is no need for different computing cores to frequently communicate to avoid duplications, so that the communication time is saved.

3.3 Parallelizing SA by Pushing Points

The PSAD and PSAD-M methods are designed for MMSR, because each point can be visited at most once. For MMSR-m and MMSR-d, this situation is different: one pick-up point can appear for multiple times in different routes. For the two new problems MMSR-m and MMSR-d, we design a pushing point strategy, which can be embedded to PSAD and PSAD-M we proposed.

We regard the input dataset of PSAD and PSAD-M as candidate pool, which stores all the candidate pick-up points that can be selected in the final route assignment. For MMSR, the candidate pool contains all the pick-up points. For MMSR-m and MMSR-d, the candidate pool should be larger. If we allow a pick-up point, c_i, to appear for pc times in the final route assignment, we push this pick-up point into the pool for pc times. Therefore, in total, there are pc c_is in the pool. Just like the other points, each c_i has equal chances to be pushed into a route. If c_i has a large pick-up probability and is closed to the taxis' starting locations, it has a large probability to be pushed into each route. Since there are pc of them, they can appear in pc routes. To avoid the same pick-up point appears in the same route for multiple times, we add a checking procedure. Before each pick-up point is inserted to a route, the algorithm checks if it has already been in the route. If so, this pick-up point is rejected. Therefore, by changing the candidate pool, we are able to use PSAD and PSAD-M to solve for MMSR-m and MMSR-d as well. We name the PSAD and PSAD-M with the pushing-point technique as PSAD-p and PSAD-M-p.

We demonstrate an example of the candidate pool for the PSAD-p in Figure 5. All pick-up points are pushed into the candidate pool. Each pick-up point can be pushed multiple times according to its pc. For example, pick-up point "1" can be pushed three times given three pick-up points "1" in the candidate pool. All the points in the pool will be randomly distributed to the three groups and each group receives approximately the same number of pick-up oints. In this example, since there are in total 23 pick-up points, Group 1 receives seven pick-up points, and Groups 2 and 3 receive 8. Each group first uses its pick-up points to initialize the route. Then, each group runs SA to optimize its route by replacing the points between the route and the other candidate pick-up points in its pool. After a preset number of steps, PSAD-p will perform rotation, interaction, and shuffling just like PSAD.

As can be seen, in this example, since pick-up point "2" is pushed twice, two points are in the pool. They can be assigned into at most two different groups and appear in the two different routes.

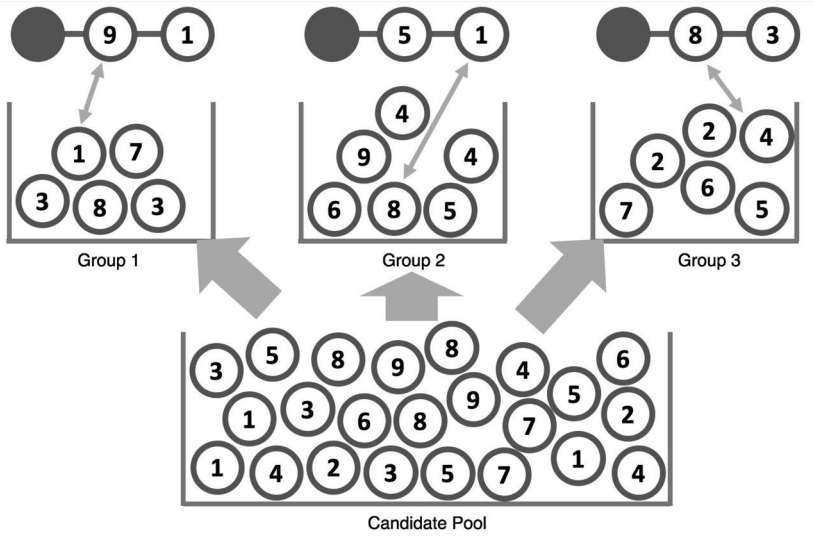


Fig. 5. The candidate pool of PSAD-p. *Note*: Each circle labeled by a number indicates a pick-up point. Same number refers to the same pick-up point. The solid circle indicates the taxis starting locations. The red line indicates a route sequence.

If so, it means that in this route assignment, two routes will include pick-up point "2." In this way, we can make sure that each pick-up point will not be visited for more than *pc* times.

The pushing point technique reduces the communication efforts between computing cores. For parallel SA, each computing core optimizes one route using SA. Without this technique, to make sure that one pick-up point will appear in the routes for less than *pc* times, in each step, each the computing core has to collect all the routes from the other cores. Then this core needs to check whether the recent pick-up point added to the routes along with its appearance in the other routes exceeds its *pc* or not. The checking operation costs for $O(\#Cores \cdot L)$ in each step leading to a significant increase of the total computing time. The pushing point technique reduces this cost to 0 as no communication is required to assure that the duplicated visits of a point is less than *pc* times. By relieving the communication burden, we are able to use a large number of cores to provide solutions for multiple taxis.

To find out the amount of times a point is pushed into the candidate pool, we need to determine *pc*. For MMSR-m, to determine one *pc* for all the pick-up points is equivalent to determine the area each pick-up point covers. If a pick-up point covers a large area, this area is likely to have more passengers and is unlikely to be fully explored by one taxi, the *pc* should then be increased. Therefore, we propose two concepts, the base area and the pick-up point area. The base area is the largest area where at most one passenger appears and one driver can fully explore. The pick-up point area is the area a pick-up point represents. Then, by finding out the number of base areas that a pick-up point area covers, we can determine its *pc*. If the base area and the pick-up area have the same size, the *pc*s for all the pick-up points are identical.

For MMSR-d, to determine the *pc* for each pick-up point, we define a new concept, the *popularity*. Popularity represents by the potential number of passengers to appear in a pick-up point, or more precisely, a pick-up area. By determining the popularity, the *pc*s for each point are found out.

4 DATA AND EXPERIMENTAL SETTINGS

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

Table 2. Benchmark Methods for Comparison

Abbr.	Methods
SARS	SA Random Search [32]
PEP	Parallel Enumerative Process [11]
PIBP	Parallel IBP [12]
CDR	Chu et al. 1999 [5]
Lou	Lou et al. 2016 [19]
PSAD	Parallel Simulated Annealing Domain Decomposition
PSAD-M	PSAD by Mixing

4.1 Data Description

Real-world Data. 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 five 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. d_∞ is determined automatically to ensure that the maximal cruising distance does not exceed 1. In these experiments, we optimize 96 routes with different starting positions. They are set to be a 12 by 8 mesh lie in the coordinates $[0.50, 0.55] \times [0.50, 0.55]$.

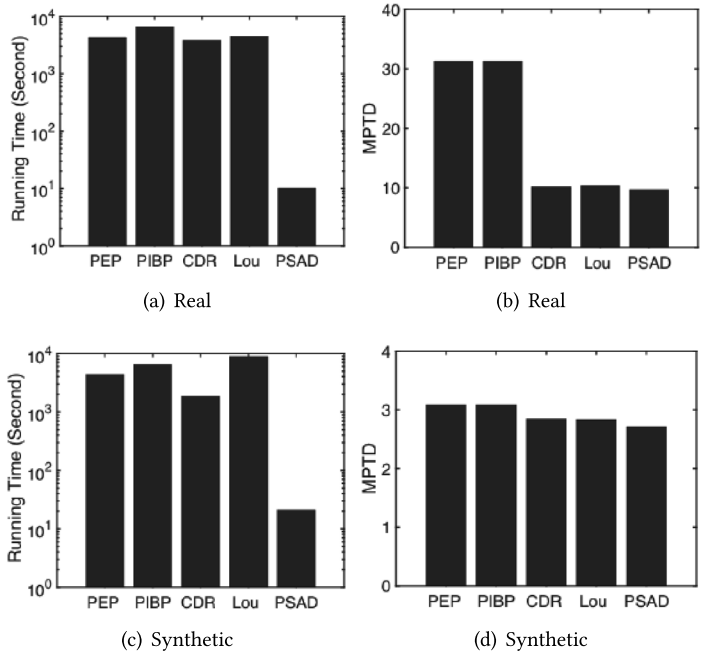
Synthetic Data. We also generate synthetic data for 100,000 pick-up points, which are uniformly distributed in a $[0, 1] \times [0, 1]$ square, and their pick-up probabilities are normally distributed with a mean 0.3 and a standard deviation 0.05, and bounded in $[0, 1]$. The starting positions and d_∞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

4.2 Experimental Settings

Benchmark Methods. We compare our two methods with five benchmarks, including one serial and four parallel algorithms, as summarized in Table 2. Our methods in this article are bold.

To solve the MSR problem, Ge et al. [11] and Huang et al. [12] proposed several methods that we replicate as benchmarks. The Enumerative process represents the best case in the algorithms in [11], and IBP is the main algorithm in [12].

Parameter Settings. 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 [32]. 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 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, we set $step_{ro} = 10$ and carefully analyze in Appendix A. For PSAD-M, we set $step_{neigh}$ to be 100 and discuss the performance in Appendix A. We test $step_{in}$ and $step_{acc}$ for 10, 100, 1,000 and find little variance of the performance. We therefore set $step_{in}$ the same as $step_{ro}$



(a) Real (b) Real

(c) Synthetic (d) Synthetic

Fig. 6. Parallel performance of different methods.

and $step_{acc} = 1{,}000$. 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. we conduct our experiments on the Seawulf Cluster provided by the Institute for Advanced Computational Science at Stony Brook University. Our experimental environment includes 100 nodes, and each of which has two Intel Xeon E5-2690v3 12 core CPUs and 128 GB DDR4 Memory.

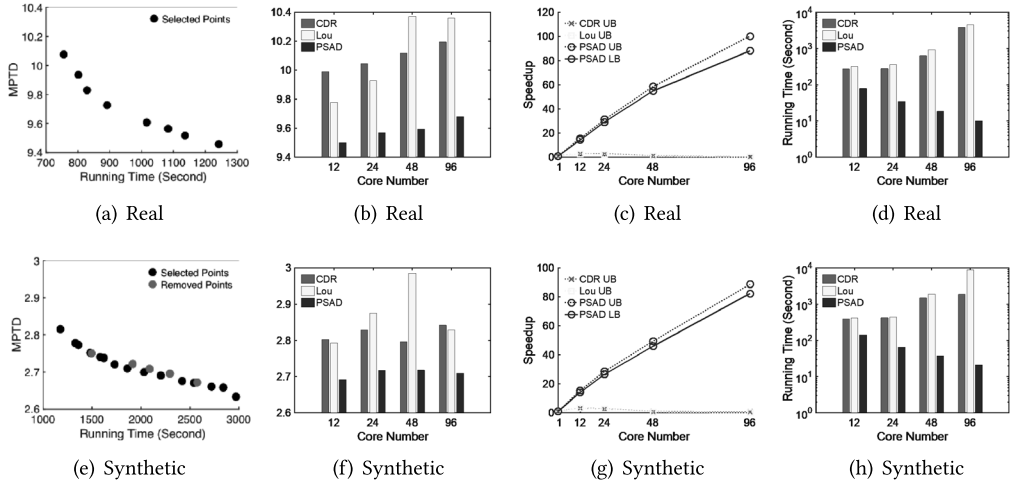
5 RESULTS AND ANALYSIS

5.1 The Overall Performance of PSAD and PSAD-M

5.1.1 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 real and synthetic datasets. Figure 6 shows that our PSAD takes 10–20 seconds to finish the searching job based on both datasets. On the other hand, other benchmarks take more than 1,000 seconds. Also, the PSAD also generates the lowest MPTD. Note that, the MPTD obtained from the synthetic dataset is generally lower than the real dataset. The reason is that the synthetic dataset contains more pick-up points. The results show that our method consistently outperforms all benchmarks in terms of the computational efficiency as well as the quality of recommendation measured by MPTD.

5.1.2 Speedup for PSAD. We further investigate the parallel performance of our method (PSAD) and two benchmark parallel techniques (CDR and Lou). The metric of speedup is defined as follows:

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



(a) Real (b) Real (c) Real (d) Real

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

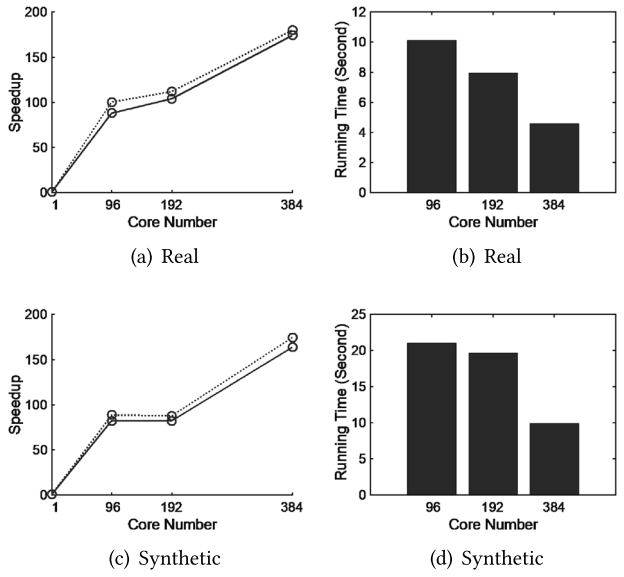
Fig. 7. Parallel performance of CDR, Lou, and PSAD with different number of cores. *Note*: In (a) and (e), the points are the SARS runs with cooling rate $\alpha = (1 - 10^{-6})^{\frac{1}{i}}$ where i is the point from the left to the right. The comparisons of achieved $MPTD$ are plotted in (b) and (f) 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.

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 MPTD levels.

In Figure 7(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 7(b) and (f), we run CDR, Lou, and PSAD based on 12, 24, 48, and 96 cores, and we record the MPTD they obtain. Based on Figure 7(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 7(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 Equation (11), the speedup of the parallel method is smaller than the actual case, and we use it as the lower bound of the



(a) Real (b) Real

(c) Synthetic (d) Synthetic

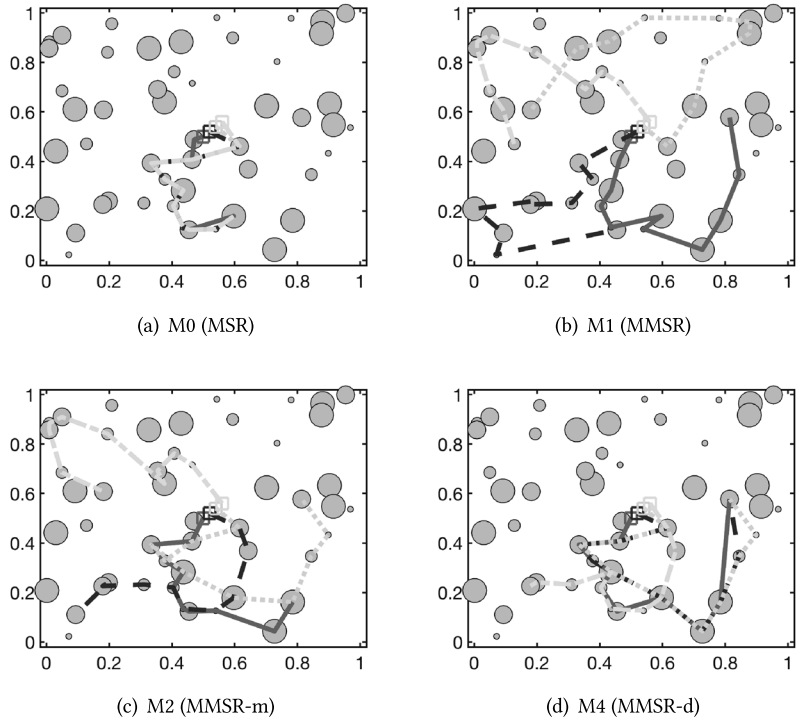
Fig. 8. Parallel performance of PSAD-M. *Note:* Dash lines represent lower bounds and solid lines are upper bounds for speedup.

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 7(c) and (g) plot the speedup of our PSAD and the benchmarks. We only show the upper bound of the speedup for CDR and 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 real-world dataset, when the number of cores is set to 96, the lower bound and the upper bound of the speedup of PSAD are 88x and 100x, respectively, while CDR and Lou can only reach 2.9x and 2.7x. For the synthetic dataset, the speedup is between 82x and 88x, while CDR and Lou are lower than 3.3x and 3.2x. We finally show the running time of the parallel methods with different numbers of cores in Figure 7(d) and (h). Combining with Figure 7(b) and (f), we can learn that PSAD achieves a lower MPTD in a much shorter running time. Together with the lower bound in Figure 7(c) and (g), we obtain the corresponding timing results of SARS. For the real-world dataset, our PSAD spends only 10.1 seconds to locate 96 routes, while it takes SARS 891 seconds to locate a worse solution. For the synthetic dataset, PSAD gets a good solution spending 21.0 seconds, while SARS takes 1,722 seconds to obtain a worse solution.

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

5.1.3 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 four cores into a group for each route searching. We plot both the lower bound and the upper bound of PSAD-M in Figure 8(a) and (c). PSAD can be considered 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 real dataset, and between 163x



(a) M0 (MSR) (b) M1 (MMSR)

(c) M2 (MMSR-m) (d) M4 (MMSR-d)

Fig. 9. A case study for four routes. *Note*: The four lines with different colors indicate the routes recommended to the four different taxi drivers with closed but different starting locations. The larger the size of a pick-up point, the larger the pc. The largest pick-up points have $pc = 4$, and the smallest ones have $pc = 1$.

and 176x for the synthetic dataset. Using 384 cores, PSAD-M doubles the speedup obtained from PSAD. For running time, according to Figure 8(b) and (d), PSAD-M spends 4.6 seconds for the real dataset and 9.9 seconds for the synthetic dataset to locate 96 high-quality route recommendations. The results confirm the consistent improvement on speedup with our PSAD-M, comparing to the performance of PSAD.

5.2 Improvements on the Objective Function

Now, we compare the benefits brought from the new objective functions in MMSR-m and MMSR-d. We have shown that by increasing the parallel size, the PSAD method can significantly reduce the computing time. For the experiments investigating the quality of recommended routes, we use up to 24 computing cores for ease of implementation. To facilitate the discussion, we use M0, M1, M2, M3, M4, and M5 to represent the MSR, MMSR, MMSR-m, MMSR-m with Property 2.1 (MMSR-mP), MMSR-d, and MMSR-d with Property 2.1 (MMSR-dP), respectively.

5.2.1 A Showcase on Synthetic Data. In Section 3, we discussed several characteristics of the three objective functions, MMSR (M1), MMSR-m (M2), and MMSR-d (M4). In Figure 9, we showcase the recommendation quality of the three forms of MMSR along with the original MSR problem (M0), based on synthetic settings. MMSR-mP (M3) and MMSR-dP (M5) are not included as they only work for improving the computational efficiency of M2 and M4, while they do not affect the recommendation quality. Detailed settings are as follows. There are four drivers and 50 pick-up points of which the locations are uniformly distributed in the map. The pick-up probabilities are uniformly generated as 0.2, 0.4, 0.6, and 0.8, and their pick-up capacities (pc) are also uniformly

distributed from a range between 1 and 4. Following our previous experimental settings, we let the starting locations to be concentrated in a small area, say (0.50, 0.50), (0.52, 0.52), (0.54, 0.54), and (0.56, 0.56).

For M0, although the starting locations of the four routes are close, they do not consider the influence of the other routes. Therefore, in Figure 9(a), recommended routes are almost squeezed to one route. The *pc*s of some pick-up points are very small but are passed by all four routes. This would cause serious traffic issues if apply in practice. Moreover, by following this recommendation, the last driver may fail to pick up anyone, because the pick-up points with low pick-up probability may have already been explored by the other three drivers.

By definition, M1 requires all routes to be mutually exclusive. This strategy avoids multiple visits for a single pick-up point, and no driver will visit a pick-up point that has already been explored. This case is represented in Figure 9(b) where the routes expand to the whole map. We can see that some pick-up points are in large sizes, indicating that they should be visited more often than the small pick-up points. However, M1 ignores such information and let some drivers to detour, and hence it results in a long total traveling distance for the four drivers.

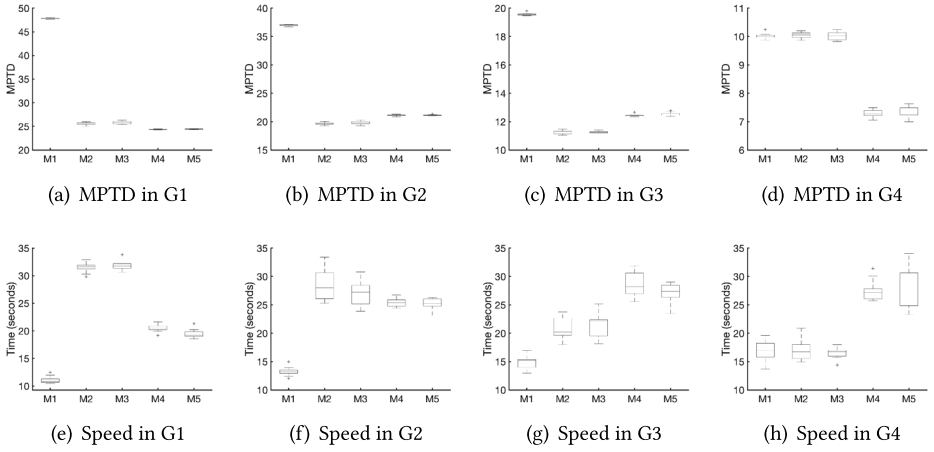
Figure 9(c) shows the recommended routes based on the settings of M2. The objective function is designed to balance between short detouring for the drivers and the small number of duplicated visits for the pick-up points. In the experiments, each pick-up point can be visited by two drivers. As can be seen, the visited pick-up points in Figure 9(c) are more compact than the points in Figure 9(b) and decentralized comparing to those in Figure 9(a). However, Figure 9(c) also shows that some pick-up points suggested by M2 have large pick-up capacity, while they are not fully served. On the other hand, if we increase *pc* for all the pick-up points, the small-size pick-up point in Figure 9(c) may be visited by too many drivers.

To consider different pick-up capacities of pick-up points, we show the recommended routes based on M4 in Figure 9(d). The visited pick-up points are more compact than the points suggested by M2 and M1 but more scattered than those recommended by M0. As shown in Figure 9(d), duplicated visits only appear in large size pick-up points. Following the recommended route of M4, drivers may visit multiple times at the pick-up points with high *pc*s and separate to explore the points with low *pc*s.

To sum up, MSR and MMSR are two extreme cases in the MMSR problem series. The MSR assumes the *pc* to be infinitely large, and the MMSR assumes the *pc* to be one. MMSR-m and MMSR-d improve the setting of *pc* to each pick-up point. MMSR-d has a more reasonable setting that allows different *pc*s for different pick-up points, while MMSR-m assumes an identical *pc*. Thus, we can expect that the MMSR-d to result in the best route recommendation among the four objective functions.

5.2.2 The Overall Performance of the MMSR Problem Series. Now we compare the solutions of MMSR, MMSR-m, MMSR-d based on the real-world data. We apply the PSAD for MMSR. For MMSR-m and MMSR-d, we apply PSAD-p to push more pick-up points to the candidate pool for those points with high *pc*. Furthermore, to check the impact of Property 2.1, we conduct the experiments for PSAD-p with and without this property.

For MMSR, the whole city is equally divided into $1{,}000 \times 1{,}000$ grids and we name such setting as G4. For MMSR-m, we divide the city into 125×125, 250×250, and 500×500 grids and name them as G1, G2, and G3 accordingly. Among the four settings, the pick-up point area for G1 is the largest and that for G4 is the smallest. We regard each block in G4 as a base area. We set the number of drivers requesting recommendation services $K = 96$. For MMSR-m, the *pc* of a pick-up point is set to be the number of base areas it covers ($pc = 64$ in G1, $pc = 16$ in G2, $pc = 4$ in G3, and $pc = 1$ in G4). By such settings, we are able to keep the total number of pick-up points for different



(a) MPTD in G1 (b) MPTD in G2 (c) MPTD in G3 (d) MPTD in G4

(e) Speed in G1 (f) Speed in G2 (g) Speed in G3 (h) Speed in G4

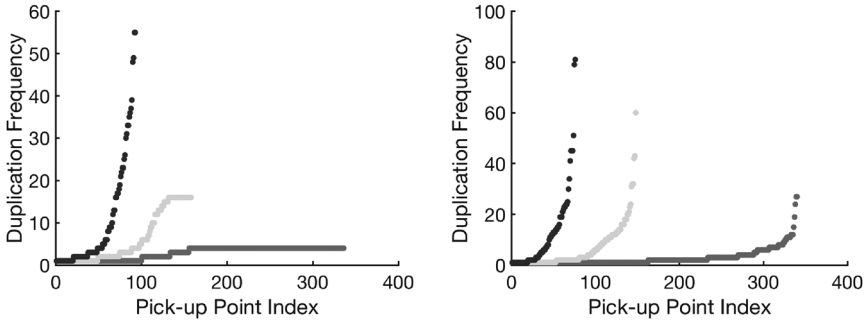
Fig. 10. Overall performance for recommending 96 routes. *Note*: G1 - G4 represent four different ways to divide the map (125×125, 250×250, 500×500, and $1{,}000 \times 1{,}000$). M1 - M5 refer to the five forms of the MMSR problem series (MMSR, MMSR-m, MMSR-mP, MMSR-d, and MMSR-dP).

grids to be similar. Then we can compare the performance of our methods in different grids under the similar settings. For MMSR-d, the pick-up points are the same as those in MMSR-m. For the setting of pc, we let it to be the total number of visited empty taxis within one hour over a scale parameter pc_0. To determine pc_0, we discover that the average pick-up probability for all the points is around 0.1. Then, for 10 empty taxis entering a pick-up point, the probability for picking up at least one passenger is about 0.65. Therefore, we increase pc by one for every 10 passing taxis, and we set $pc_0 = 10$ as initialization.

Figure 10 summarizes the overall performance, in terms of the MPTD as well as the computational efficiency. Each boxplot shows the results from 10 independent experiments. From Figure 10(a) to 10(c), MMSR-m (M2) and MMSR-mP (M3) always lead to a lower MPTD, which indicates a higher recommendation quality, in comparison with M1. By pushing more points, some precious nearby points with high pick-up probabilities can be visited multiple times, resulting in a decrease of the MPTD. However, MMSR-m and MMSR-mP obtain similar MPTD to MMSR in G4, as shown in Figure 10(d). The reasons are the following. G4 contains $1{,}000 \times 1{,}000$ grids, and the area covered by each pick-up point is very small. Considering that there are many pick-up points with low pick-up probabilities in rural areas, we set $pc = 1$ for these cases. Since all pick-up points have the same pick-up capacity in MMSR-m, we set $pc = 1$ for all. If no additional pick-up point is pushed into the candidate pool, PSAD-p works exactly the same as PSAD. Thus, in Figure 10(d), MMSR-m and MMSR-mP obtain MPTD at the same level of MMSR.

For the MMSR-d (M4) and MMSR-dP (M5), we always obtain a better MPTD comparing to MMSR, regardless the setting of grids. As shown in Figure 10(d), MMSR-d and MMSR-dP manage to obtain a lower level of MPTD since they use different pcs for different pick-up points in G4. Although rural pick-up points exist, they do not affect the pc of urban pick-up points. However, MMSR-m and MMSR-mP can achieve better MPTDs than MMSR-d and MMSR-dP in G2 and G3. The results show that although MMSR-d has a more reasonable setting than MMSR-m, MMSR-m may still achieve better MPTD in some special cases. Thus, the two forms of MMSR can be jointly applied for practical purposes.

Figure 10 (e)–(h) demonstrates the results of computing time. As can be seen, MMSR has the smallest computing cost due to its simple settings. For MMSR-m and MMSR-d, the time difference



(a) MMSR-m (b) MMSR-d

Fig. 11. The frequency of duplicated routes. *Note*: The blue line, green line, and red line analyze the duplication frequency for G1, G2, and G3, respectively.

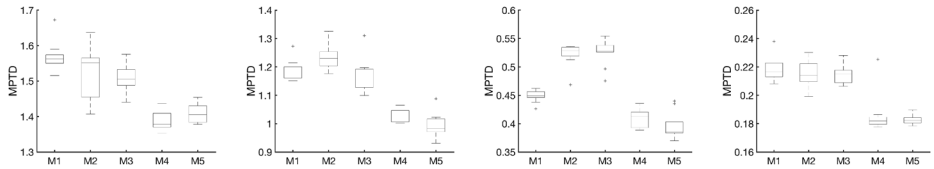
varies under different grid settings (G1–G4). According to Property 2.1, only if *pc* is larger than the total number of drivers, the candidate pool can be reduced. Given 96 drivers, we only see small improvements obtained by applying the property. That is, there are small differences between MMSR-m and MMSR-mP and between MMSR-d and MMSR-dP. Although MMSR-m and MMSR-d have higher computational costs than MMSR, we can see that the computing costs are in the same magnitude level. Note that we only use 24 computing cores in this test, the computing time can be reduced further by increasing the number of computing cores.

The MMSR-m and MMSR-d enable duplicated visits of each pick-up point. For an route assignment, we sort the frequency of duplicated visits of the visited pick-up points from small to large in Figure 11. As can be seen, some visited pick-up points are less popular and do not get many drivers. Some points are very attractive, and they are more often.

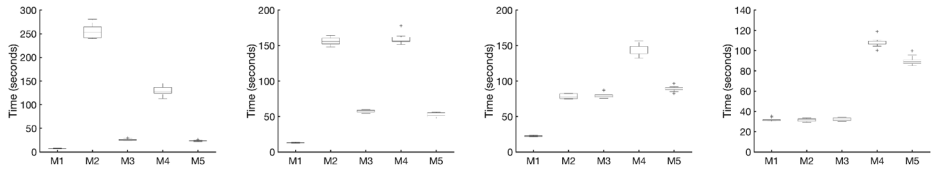
For the MMSR-m problem as shown in Figure 11(a), we can see an unnatural bound for the G2 settings in green and G3 settings in red. This is because MMSR-m requires all *pc* to be the same, the duplicated visit frequency of are bounded by the average *pc*. Figure 11(b) shows a more natural route duplication frequency, as each pick-up point has its own *pc*. Thus, MMSR-d enables us to locate some attractive pick-up points that are also capable of handling a large number of visits. In summary, we can see that by introducing *pc*, some pick-up points will stand out and may take more places in the route recommendation process.

5.2.3 The Impact of Property 2.1. Our Property 2.1 suggests that if the number of drivers K is smaller than the maximum pick-up capacity, we can set the maximum pick-up capacity to K. Thus, to investigate the importance of Property 2.1 in MMSR-m and MMSR-d, we conduct experiments for the MMSR problem series with a small number of drivers. We set $K = 6$. Other settings regarding pick-up points follow the discussions in Section 5.2.2.

Our task in the experiments is to recommend routes for six drivers with their starting locations at $(0.50, 0.50), (0.51, 0.51), \ldots, (0.55, 0.55)$. Figure 12 reports the overall performance of the MMSR problem series, for the MPTD and computing time. The key findings can be summarized as follows. First, As shown in Figure 12(a)–(d), MMSR-d and MMSR-dP can always achieve lower MPTDs than other models, indicating the superior of MMSR-d to other forms of MMSR, in terms of the recommendation quality. Second, after considering Property 2.1, we can achieve similar or even better MPTDs for MMSR-m and MMSR-d. Third, MMSR-m and MMSR-mP can lead to slightly lower MPTDs than the sample form of MMMSR in G1 and G4, while they obtain higher MPTDs than MMSR in G2 and G3. These results indicate that the MMSR-m does not have significant advantages in small K cases. Last, although the simple form of MMSR still performs the fastest among



(a) MPTD in G1 (b) MPTD in G2 (c) MPTD in G3 (d) MPTD in G4



(e) Speed in G1 (f) Speed in G2 (g) Speed in G3 (h) Speed in G4

Fig. 12. Overall performance for recommending six routes. *Note*: G1–G4 represent four different ways to divide the map (125×125, 250×250, 500×500, and $1{,}000 \times 1{,}000$). M1–M5 refer to the five forms of the MMSR problem series (MMSR, MMSR-m, MMSR-mP, MMSR-d, and MMSR-dP).

M1–M5, as shown in Figure 12(e)–12(h), considering Property 2.1 (MMSR-mP and MMSR-dP) can significantly reduce the computational cost. Especially for G1, the computing time of MMSR-mP and MMSR-dP is close to that of MMSR.

6 RELATED WORK

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

6.1 Human Trajectory Mining

Different from research based on traditional urban data that focuses on characterizing the market dynamics [8, 9], 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 [12, 24, 25]. Specifically, Huang et al. [12] improved the algorithm by making use of the iterative property of the PTD functions from the MSR problem so as to significantly reduce the search space. However, due to the high computational cost in the search space reduction process, these algorithms are computationally prohibited for the MSR problem in large dimensions. There are different studies on capacity in taxi driver problems. For addressing the ride-sharing problem, Alonso-Mora et al. [1] investigated the relation between the vehicle number and vehicle capacity for serving the taxi demands. Ma et al. [20] related the taxi capacity to taxi fare. Different from above two papers that focus on the capacity of the taxis, we focus on the capacity of the pick-up points in this work.

Efforts have also been taken to develop other applications using human trajectory data. Ge et al. [10] developed a taxi business intelligence system for reducing inefficiency in energy consumption and improving customer experience and business performance. Powell et al. [24] 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. [25] 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. [36] 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. [37] 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. [29] adapted artificial neural network to recognize the high passenger-finding potential road clusters for the taxicabs. Ma et al. [20] studied the taxis in ride-sharing and provided algorithms for taxi searching and scheduling to reduce the total traveling distance. Ma et al. [21] also improved the recommendation of the ride-sharing for practical concerns by involving monetary constraints for taxicab drivers and passengers. Liu et al. [16] studied workflow by unraveling the patterns hidden in location traces while constructing estimating parameters and constructing workflow states automatically. Also, Liu et al. [17] designed a workflow modeling framework for health-care operation and management based on indoor trajectory data. More work based on human trajectory data can be found in [30, 31, 38, 39].

6.2 Parallel Simulated Annealing

SA was initially proposed by Kirkpatrick et al. [13] 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. Studies are conducted for optimizing the cooling schedules and move generations in speeding up the SA calculations. Nourani and Andresen [23] compared linear, exponential, logarithm, and other schedules by looking into the entropy production rate. They tested the schedules for one three state system and one NP-hard problem and discovered different behaviors of those schedules. Triki et al. [28] studied the cooling schedules through the theoretical concepts including thermodynamic equilibrium and demonstrate that all classic cooling schedules are equivalent. In our experiments we choose the exponential schedule as it achieves a reasonable solutions within a short time. In terms of move generation, Li and Ma [14] adapted simulated annealing to binary problems. They designed a specific move generation for the target problems by adopting a technique similar to the local search in optimizing the continuous functions. In our SA settings, we also use the move generation proposed in Ye et al. [34] for this specific MSR problem.

To further accelerate the calculation, efforts are then spent on parallelizing SA. 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 [5], the running time is large because of the significant increase of the communication time [19]. According to [19], among the literature on parallel SA, for the parallel size of 32 or larger, the best speedup record was 19.6x for the traveling salesperson problem (TSP) [26]. 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. [22]. Efforts are also spent in combining the parallel SA by domain decomposition. However, the largest speedup reported was 14.1x [7].

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 [2] as they both recommend a sequence of locations but with relatively different sizes 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) [6, 27] 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 [3].

7 CONCLUSIONS

We formalized a new MMSR problem series (MMSR, MMSR-m, and MMSR-d) to improve and generalize the original MSR problem for route optimization. Our MMSR problem series aims to

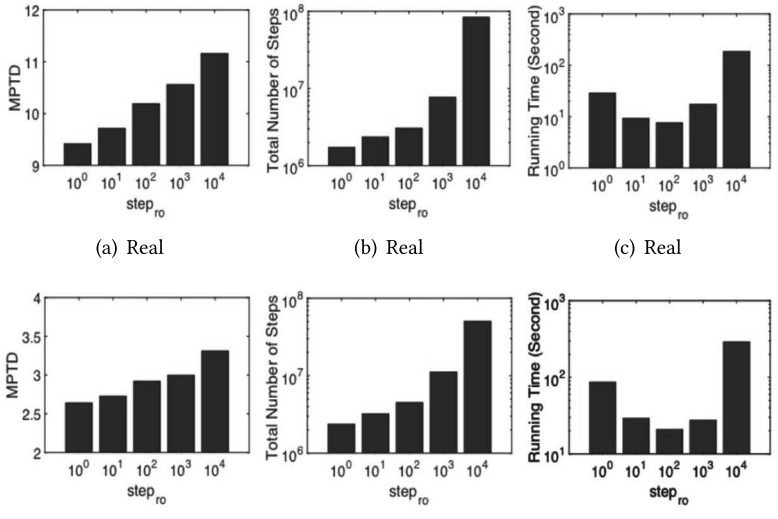
simultaneously recommend optimal routes to multiple users (e.g., taxi drivers). While the MMSR requires all recommended routes to be mutually exclusive with each other, the MMSR-m allows a certain level of route overlap, and the MMSR-d considers more complicated case when pick-up capacities are different from point to point. We proposed two parallel algorithms PSAD and PSAD-M, along with a push-point strategy to address the computing issues among the problem series we proposed. While PSAD and PSAD-M are designed for the MMSR, the push-point strategy can be embedded to the two algorithms for handling the MMSR-m and the MMSR-d. Our results confirm the superiority of the new problem formulation over its original form in providing an effective and efficient solution to multi-user route recommendation.

APPENDIX

A PARAMETER ANALYSIS

We analyze two parameters, $step_{ro}$ of PSAD and $step_{neigh}$ of PSAD-M for both real and synthetic datasets.

For $step_{ro}$, we test five cases from $step_{ro}$ being 10^0 to 10^4. As shown in Figure A.1(a) and (b) and Figure A.1(d) and (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.



(a) Real (b) Real (c) Real

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

Fig. A.1. Parameter selection based on the parallel performance of PSAD.

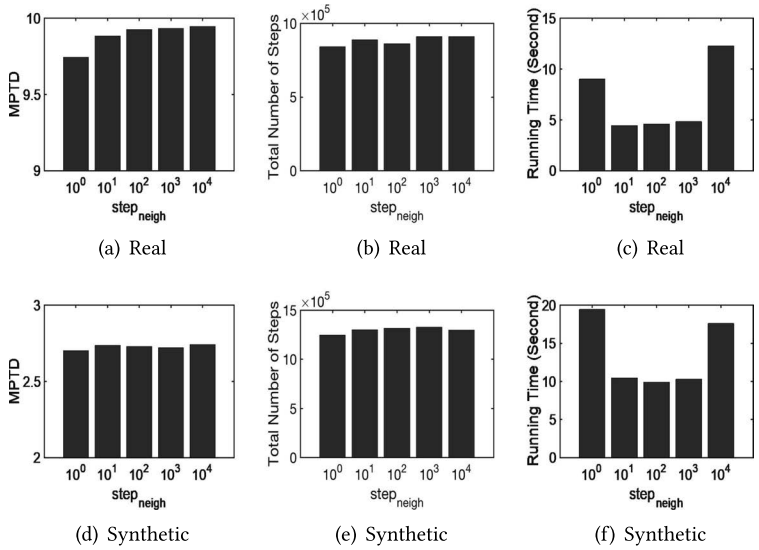


Fig. A.2. Parameter selection based on the parallel performance of PSAD-M.

In Figure A.1(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 A.2, 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 A.2. 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.

REFERENCES

[1] Javier Alonso-Mora, Samitha Samaranayake, Alex Wallar, Emilio Frazzoli, and Daniela Rus. 2017. On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment. *Proceedings of the National Academy of Sciences* 114, 3 (2017), 462–467.

[2] David L. Applegate, Robert E. Bixby, Vasek Chvatal, and William J. Cook. 2011. *The Traveling Salesman Problem: A Computational Study*. Princeton University Press.

[3] 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.

[4] Zheyong Bian and Xiang Liu. 2019. Mechanism design for first-mile ridesharing based on personalized requirements part I: Theoretical analysis in generalized scenarios. *Transportation Research Part B: Methodological* 120 (2019), 147–171. https://www.sciencedirect.com/science/article/abs/pii/S0191261517308044.

[5] King-Wai Chu, Yuefan Deng, and John Reinitz. 1999. Parallel simulated annealing by mixing of states. *Journal of Computational Physics* 148, 2 (1999), 646–662.

[6] George B. Dantzig and John H. Ramser. 1959. The truck dispatching problem. *Management Science* 6, 1 (1959), 80–91.

[7] 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.

[8] 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. https://www.sciencedirect.com/science/article/abs/pii/S1062976917302132.

[9] Frank J. Fabozzi and Keli Xiao. 2019. The timeline estimation of bubbles: The case of real estate. *Real Estate Economics* 47, 2 (2019), 564–594.

[10] 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.

[11] 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.

[12] 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.

[13] Scott Kirkpatrick, C. Daniel Gelatt, and Mario P. Vecchi. 1983. Optimization by simulated annealing. *Science* 220, 4598 (1983), 671–680. https://science.sciencemag.org/content/220/4598/671.

[14] Xuesong Li and Lin Ma. 2012. Minimizing binary functions with simulated annealing algorithm with applications to binary tomography. *Computer Physics Communications* 183, 2 (2012), 309–315.

[15] Bin Liu, Hui Xiong, Spiros Papadimitriou, Yanjie Fu, and Zijun Yao. 2015. A general geographical probabilistic factor model for point of interest recommendation. *IEEE Transactions on Knowledge and Data Engineering* 27, 5 (2015), 1167–1179.

[16] 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.

[17] 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.

[18] Qi Liu, Enhong Chen, Hui Xiong, Yong Ge, Zhongmou Li, and Xiang Wu. 2014. A cocktail approach for travel package recommendation. *IEEE Transactions on Knowledge and Data Engineering* 26, 2 (2014), 278–293.

[19] Zhihao Lou and John Reinitz. 2016. Parallel simulated annealing using an adaptive resampling interval. *Parallel Computing* 53 (2016), 23–31. https://www.sciencedirect.com/science/article/pii/S0167819116000430.

[20] Shuo Ma, Yu Zheng, and Ouri Wolfson. 2013. T-share: A large-scale dynamic taxi ridesharing service. In *Proceedings of the 2013 IEEE 29th International Conference on Data Engineering (ICDE'13)*. 410–421.

[21] 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.

[22] Samir W. Mahfoud and David E. Goldberg. 1995. Parallel recombinative simulated annealing: A genetic algorithm. *Parallel Computing* 21, 1 (1995), 1–28.

[23] Yaghout Nourani and Bjarne Andresen. 1998. A comparison of simulated annealing cooling strategies. *Journal of Physics A: Mathematical and General* 31, 41 (1998), 8373.

[24] Jason W. Powell, Yan Huang, Favyen Bastani, and Minhe Ji. 2011. Towards reducing taxicab cruising time using spatio-temporal profitability maps. In *Proceedings of the International Symposium on Spatial and Temporal Databases*. Springer, 242–260.

[25] 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.

[26] Andrew Sohn. 1996. Generalized speculative computation of parallel simulated annealing. *Annals of Operations Research* 63, 1 (1996), 29–55.

[27] Paolo Toth and Daniele Vigo. 2002. *The Vehicle Routing Problem*. SIAM.

[28] Eric Triki, Yann Collette, and Patrick Siarry. 2005. A theoretical study on the behavior of simulated annealing leading to a new cooling schedule. *European Journal of Operational Research* 166, 1 (2005), 77–92.

[29] 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.

[30] 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.

[31] Tong Xu, Hengshu Zhu, Hui Xiong, Hao Zhong, and Enhong Chen. 2019. Exploring the social learning of taxi drivers in latent vehicle-to-vehicle networks. *IEEE Transactions on Mobile Computing* (2019). https://ieeexplore.ieee.org/abstract/document/8708931.

[32] 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'15)*. IEEE, 1078–1084.

[33] Zeyang Ye, Keli Xiao, and Yuefan Deng. 2018. A unified theory of the mobile sequential recommendation problem. In *Proceedings of the 2018 IEEE International Conference on Data Mining (ICDM'18)*. IEEE, 1380–1385.

[34] Zeyang Ye, Keli Xiao, Yong Ge, and Yuefan Deng. 2019. Applying simulated annealing and parallel computing to the mobile sequential recommendation. *IEEE Transactions on Knowledge and Data Engineering* 31, 2 (2019), 243–256.

[35] 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 *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 2624–2633.

[36] 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.

[37] 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.

[38] 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'15)*. IEEE, 609–618.

[39] 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'12)*. IEEE, 1212–1217.

Received January 2019; revised June 2019; accepted September 2019