Make Uber Faster: Automatic Optimization of Uber Schedule Using OpenStreetMap Data

Wanwan Li
Department of Computer Science
George Mason University
Fairfax, VA, US
wli17@gmu.edu

(a) A user is waiting for an Uber.

(b) Uber schedule is optimized with our approach.

Fig. 1: Demo of our approach. (a) shows a user waiting for Uber (the white car is the 3D model we use). (b) shows an Uber schedule optimized with our approach using OpenStreetMap data. Tasks are plotted with an origin as a circle and a destination as a square. Tasks are colored the same as the Uber car to which they are assigned.

***Abstract*—Nowadays, ride-shares services such as Uber and Lyft are becoming more and more popular among people as shown in Figure 1 (a). Therefore, as a consequence, improving Uber drivers' scheduling policies becomes a hot research topic. In this paper, we present Make Uber Faster: a stochastic optimization approach to automatically improve the Uber scheduling strategy (b). After considering the benefits from both sides: Uber drivers and customers, we realize there is still a large among of space to improve the scheduling strategy over the traditional straightforward heuristic strategies such as the nearest customers first strategy and the shortest jobs queues first strategy. By taking advantage of the OpenStreetMap (OSM) dataset, we conduct numerical experiments based on procedural computer animations and simulations to compare different strategies with ours both visually and statistically to demonstrate how our strategy overperforms the others.**

***Index Terms*—scheduling strategy, stochastic optimization, computer animations, procedural simulations**

I. Introduction

Given the advanced technologies of GPS location tracking services provided on smart mobile devices, online platforms are able to share customers' location information with drivers so that it is easier to find nearby drivers and makes it faster to share a ride with the customers. This GPS technology helps grow large-scale rideshare companies such as Uber and Lyft to connect a large number of riders and drivers through cell phone apps. An important question for the rideshare company to think about is how to schedule the drivers efficiently to achieve the goals that can satisfy both the drivers and the riders. This is actually an interesting research topic about scheduling strategy. However, there are some existing straightforward scheduling strategies that are proposed given to some important heuristics such as shorter jobs can start earlier than some longer jobs. This can efficiently decide which job to be started first but they are always somehow limited and losing some other considerations when focusing on one consideration.

There are a lot of research works on exploring the best scheduling strategies. For example, the job-shop scheduling problem is a notoriously difficult problem in combinatorial optimization and its computational study has been well explored by Applegate et al. [1] in 1991. Both conventional and new solution techniques are discussed by Blazewicz et al. [2]. To solve this problems, both genetic algorithms [3]–[5] and some other methods such as branch bounding algorithm [6], multi-purpose machines [7], and tabu-search technique [8] have been studied. Scheduling problems similar to the job-shop scheduling problem such tasks scheduling for parallel and

(a) Default Roads. (b) Merged Roads. (c) Split Roads (Result).

Fig. 2: Navigation Graph Generation. (a) The default roads download with OpenStreetMap data. (b) Roads are merged. (c) Roads are split and the navigation Graph $G = (V, E)$ is generated. As we can see from subfigure (a), the downloaded roads network extracted from the raw OpenStreetMap (OSM) data has less accurate navigation graph representations. According to the splitting operation in (b) and merging operation in (c), the result navigation graph has a more accurate representation. As shown in subfigure (c), the processed result of the navigation graph has the vertices specified as red spheres are representing the roads' intersections or crosses and has the edges specified as colored curves are representing the roads.

distributed computing systems [9], bicriterion scheduling [10], surgical scheduling [11], sequencing scheduling [12], economic lot scheduling [13], voltage scheduling [14], payment scheduling [15], employee scheduling [16], and manufacturing scheduling [17] etc. have also been proposed and solved by researchers. Unfortunately, none of these solutions can be directly copied to solve the Uber scheduling problem that we are going to propose in this paper.

Not surprisingly, there are also plenty of research works discussing about how to optimize the ride-sharing schedule. For example, bee colony optimization approach was proposed by Teodorovic et al. [18] to solve the ride-matching problem. Agatz et al. [19] proposes a dynamic ride-sharing simulation study in Metro Atlanta. At the mean while, queueing-theoretic approach [20], hierarchical data-driven approach [21], stable matching [22], ADA paratransit operation [23], particle swarm optimization [24], Simulation-Based optimization [25] etc. are also proposed for optimizing ride-sharing schedules. However, as the most realistic simulation environment to compare different types of strategies, OpenStreetMap (OSM) data [26], which is widely used in GPS-related applications, provides such a possibility. But none of these works use the OSM data to simulate and validate the optimization process as we do.

Therefore, like other multi-criteria scheduling problems, in order to solve the Uber scheduling problems, we need to consider at least two criteria: the drivers' sides and the riders' sides. For the drivers, we want to balance the workload to make it fairer. That is, we need to avoid the case that several drivers have too many tasks to deal with while some drivers have no tasks at all. As for the rider side, we hope to minimize the waiting time for each customer. That is, we hope to avoid the case that the rider is kept waiting for one driver while some other drivers have no tasks to do. There are some heuristics that can help improve the Uber scheduling strategy, however, we will demonstrate that such heuristics always lead to a locally optimal solution. Contributions of our work include:

- We devise a navigation graph-based stochastic optimization approach to automatically solve the Uber scheduling problem that best satisfies the criteria from both drivers' and riders' sides using a simulated annealing algorithm.
- We simulate the optimized Uber schedules using OpenStreetMap data to visually validate the efficacy of our proposed optimization approach. Full demo video can be watched at here https://youtu.be/KMbIEYlIoOE.
- We conduct a series of numerical experiments by comparing different scheduling strategies with ours through numerical and statistical tests to validate our approach.

II. Technical Approach

In order to stimulate the Ubers' route on a real-world map and optimize the Uber schedules, first, we need to design a data structure storing all necessary navigation information on a map: we call it a navigation graph. After generating the navigation graph, we need to numerically evaluate how well a proposed schedule solution matches with optimization goals, we call them cost functions. Then we need to devise an efficient algorithm to minimize the cost functions by updating the solutions repeatedly through several move strategies that can explore the solution space sufficiently. In this section, we discuss our methods to achieve each step listed above.

A. Problem Representations

Navigation Graph. A navigation graph is denoted as $G = (V, E)$ where vertices $V = \{v_i | i = 1, 2, ..., |V|\}$ represents the road crossings and edges $E = \{(v_i, v_j) \in V^2 | i, j \in [1, |V|] \wedge i \neq j\}$ are the roads. Navigation graph $G = (V, E)$ is the fundamental data structure upon which optimization is applied. We design a pipeline to construct navigation graphs automatically using real-world data. Through an open-source Unity 3D asset, GO Map, which has been developed to support multiple vector map APIs including Mapbox, OpenStreetMap, Esri, and Mapzen, we can download accurate road networks and geolocation data by calling these APIs. Therefore, given these APIs, we can download the realistic urban street layout from the OpenStreetMap (OSM) dataset directly.

However, those downloaded road data by default are segmented irregularly and can not form a navigation graph structure where the vertices should be the road crosses and edges should be the roads. As shown in Figure 2, (a) shows the default road network where there exist lots of mismatches on the road crosses. Therefore, (b) shows the result that we merge the roads wherever there are any two road segments sharing the same ending points close to each other and are contingent on each other with similar tangent directions. After the merging process, we split the road segments according to T-junctures as shown in (c). The condition to meet with a T-juncture is if there is one ending point of a road segment is lying on another road segment. After this splitting process, the navigation graph $G = (V, E)$ is constructed.

Workers, Tasks, and Schedule. In our problem statement, we want to assign n rideshare tasks to m Uber drivers at a particular moment. For simplicity, we set up the initial locations of Uber drivers (workers W) at random vertices in the navigation graph $G = (V, E)$ as $W = \{w_k | k \in [1, m]\}$ and $w_k \in V$. Similarly, we assume all rideshare tasks are starting at a random vertex on the navigation graph as the origin and ending at another random vertex as the destination. We represent the n tasks $T = \{t_k | k \in [1, n]\}$ as a list of arbitrary vertex pairs vertex i and vertex j in navigation graph G. Mathematically, $t_k \subseteq \{(v_i, v_j) | i \neq j \wedge (i, j) \in [1, |V|]^2\}$. We call an assignments of the tasks to the drivers as a schedule. Then a schedule S is mathematically defined as a list of combinations between workers W and tasks T such that $S = \{(w_i, t_j) | i \in [1, m] \wedge j \in [1, n]\}$. Also, we want a schedule to cover all of the tasks, that is $\bigcup_{(w_i, t_j) \in S} \{t_j\} = T$. We store the schedule as a list of task queues so that we can access any task by $S_{i,j}$ as the j^{th} task of the i^{th} worker. To convert a task to a vertex on graph, we use $S^{\text{o}}_{i,j}$ represent the origin vertex of task $S_{i,j}$ and use $S^{\text{d}}_{i,j}$ represent the destination vertex of task $S_{i,j}$. With this representation, $|S_i|$ denotes the total number of tasks that are assigned to the i^{th} worker.

All-Pairs Shortest Paths. In order to evaluate how much time does it take for the Uber driver to finish a task, the driving distance is a prior consideration. In our approach, we assume the driving speed is approximately keeping a constant value v_{driver}. Then given arbitrary tasks with an origin and a destination, the time to finish the task is proportional to its driving distance. For simplicity, we consider the drivers' navigation routes as the shortest paths between every two locations. Let $d(v_i, v_j)$ denote the minimal navigation distance between vertex v_i and vertex v_j in navigation graph G, we calculate $d(v_i, v_j)$ using Floyd–Warshall algorithm [27] to solve the all-pairs shortest paths problem. By definition, minimal distance $d(v_i, v_j)$ is solved through dynamic programming formula: $d(v_i, v_j) = \min\{d(v_i, v_j)^k | k = 0, 1, ...|V|\}$ and calculate $d(v_i, v_j)^k$ using recursive formula:

$$d(v_i, v_j)^{k+1} = \min\left(d(v_i, v_j)^k, d(v_i, v_{k+1})^k + d(v_{k+1}, v_j)^k\right) \tag{1}$$

where base case: $d(v_i, v_j)^0 = ||\mathbf{p}(v_i) - \mathbf{p}(v_j)||$, if vertex v_i and vertex v_j are adjacent to each other on the navigation graph G; $d(v_i, v_j)^0 = +\infty$, otherwise. And $\mathbf{p}(v)$ is a function converts vertex v to a 3D position in world space.

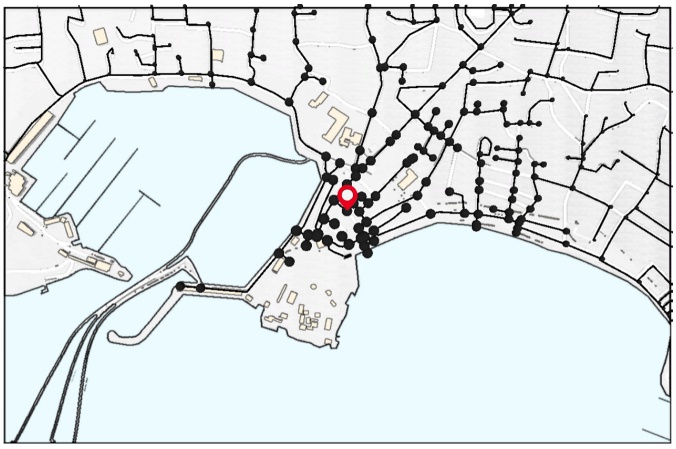


Fig. 3: All-pairs shortest paths generation. Red pin specifies the source vertex as the center, all pairs of the shortest paths that connect the source vertex with those vertices reachable from the source vertex are sorted according to their minimum distances. Minimum distances from arbitrary vertices to the center are visualized through black spheres with varying sizes: larger spheres represent the vertices nearer to the center and smaller spheres represent the vertices further from the center.

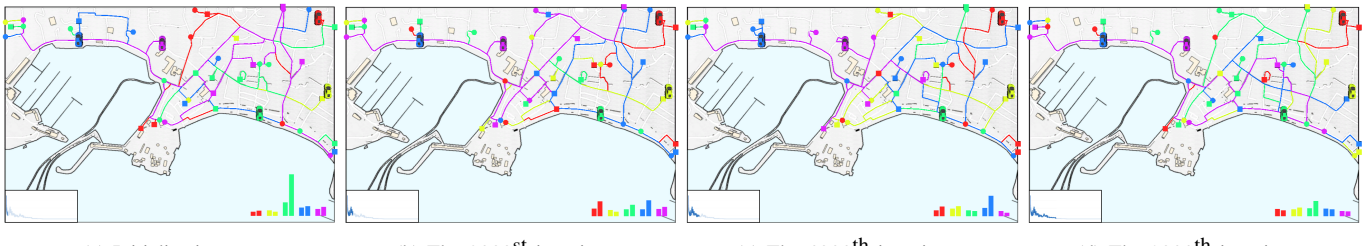
An example of all-pairs shortest paths generation is shown in Figure 3. In this demo, we set up a location near Bodrum Beach in Bodrum, Turkey. We calculate the min navigation distance from every vertex in the map to the center of this map which is labeled by a red pin, the distance of each vertex is plotted with a black sphere whose size decreases as its min distance to the center increases. All vertices not labeled out are those not reachable from the center vertex in this map. As we can see, most of the vertices are reachable to the center vertex and there is an obvious trend that vertices spatially closer to the center vertex tends to have larger sizes of black spheres.

B. Cost Functions

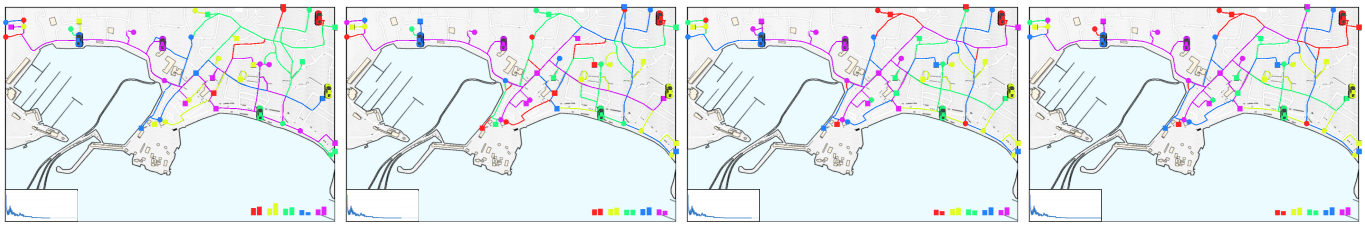
Given the above problem representations, we want to optimize the assignment of n rideshare tasks to m Uber drivers at a particular moment by considering two parts: the driver side and the rider side. We evaluate the schedule S through the total cost function that consists of two cost terms including (1) Driver cost function $C_{\text{driver}}(S)$ that evaluates how much is the maximum time does the drivers take to finish all the tasks and (2) Rider cost function $C_{\text{rider}}(S)$ that evaluates how much is average waiting times for all riders. Mathematically, total cost $C_{\text{total}}(S)$ is defined as:

$$C_{\text{total}}(S) = w_{\text{driver}} C_{\text{driver}}(S) + w_{\text{rider}} C_{\text{rider}}(S), \tag{2}$$

where w_{driver} and w_{rider} are the blending weights for $C_{\text{driver}}(S)$ and $C_{\text{rider}}(S)$ respectively.



(a) Initialization. (b) The 3000st iteration. (c) The 6000th iteration. (d) The 9000th iteration.



(e) The 12000st iteration. (f) The 15000st iteration. (g) The 18000th iteration. (h) The 20000th iteration (Result).

Fig. 4: Optimization Process. In this example, there are five Uber cars in total. Five Uber cars are rendered with five different colors. At the same time, there are 20 tasks assigned to the five drivers. In the beginning, the Uber cars and the tasks are initialized with random locations on the map. (a) The tasks are randomly assigned to five Ubers drivers during the initialization step. Figure (b-h) shows the optimization process based on the number of iterations completed at that iteration. During the optimizations, tasks' are randomly selected and exchanged between Uber drivers. Figure (h) shows the optimization result of the optimization schedule that the maximum finishing time of the driver and the riders' average waiting time are minimized. Subfigures in the bottom left corner plot the total cost function values. Subfigures in the bottom right corner have ten bars with five different colors are corresponding to drivers' five colors: they plot the maximum finishing time of that driver with the same color (left bar) and the total waiting time of all riders assigned to that driver (right bar).

Driver Cost. In order to balance the amount of work assigned to different Uber drivers, we hope to minimize the max time that the last Uber driver has finished all the tasks. Therefore, given an arbitrary schedule S, we want to penalize this schedule by calculating the max working time it takes to finish all the tasks in parallel by the drivers through the driver cost:

$$C_{\text{driver}}(S) = \frac{1}{v_{\text{driver}}} \max_{i\in[1,m]} \left(d(w_i, S^{\text{o}}_{i,1}) + \sum_{j=1}^{|S_i|} d(S^{\text{o}}_{i,j}, S^{\text{d}}_{i,j}) + \sum_{j=2}^{|S_i|} d(S^{\text{d}}_{i,j-1}, S^{\text{o}}_{i,j}) \right) \tag{3}$$

As shown in Equation 3, the time to finish a task consists of three parts: (1) Starting time: time to drive from the start location of the driver towards the origin of the first task, (2) Riding time: time to drive from the origin of a task towards the destination of a task, and (3) Dropping-off time: time to drive from the destination of the previous task towards the of origin of the next task. Actually, driver cost $C_{\text{driver}}(S)$ calculates exactly the longest time for a driver to finish all the assigned tasks that consider these three parts of time consumption.

Rider Cost. As we know, most users want Uber to come as soon as possible after they have placed their orders. Therefore, another important consideration to improve the schedule of the Uber system is the riders' waiting time. To achieve so, we want to penalize any given arbitrary schedule S by calculating the rider's average waiting time through the rider cost:

$$C_{\text{rider}}(S) = \frac{1}{v_{\text{driver}}} \sum_{i=1}^{m} \sum_{k=1}^{|S_i|} \left(d(w_i, S^{\text{o}}_{i,1}) + \sum_{j=1}^{k-1} d(S^{\text{o}}_{i,j}, S^{\text{d}}_{i,j}) + \sum_{j=2}^{k} d(S^{\text{d}}_{i,j-1}, S^{\text{o}}_{i,j}) \right) \Big/ \left(m \sum_{i=1}^{m} |S_i| \right) \tag{4}$$

As shown in Equation 4, there is an accumulative effect of waiting time as the current rider's waiting time includes the previous rider's waiting time. This can always happen when there are more riders than drivers at a particular moment. Just because of this effect, the average waiting time can also be calculated recursively. For minimizing the average waiting time of parallel tasks queues, there is a straightforward heuristic strategy is to do the shortest job first (SJF), therefore, we compare our approach with SJF in the experiment section.

C. Schedule Optimization

As shown in Figure 4, the process of optimizing a Uber schedule on a navigation graph $G = (V, E)$ with our proposed approach is presented.In the beginning, the Uber cars and the tasks are initialized with random locations on the map. The tasks are randomly assigned to five Ubers drivers during the

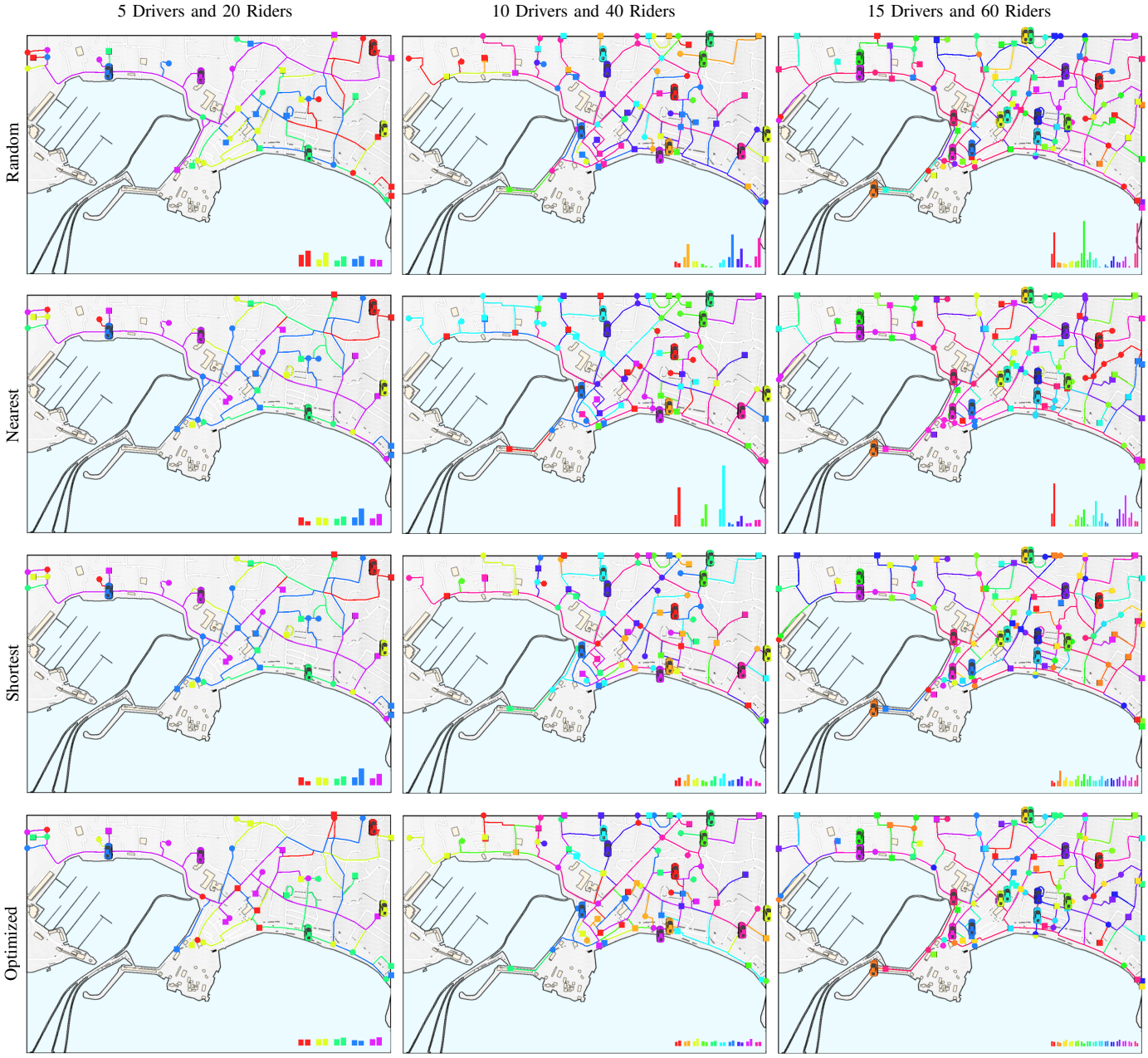


Fig. 5: Testing our program with different parameter settings and comparing with different schedule strategies. Among these results, three columns represent three different parameter settings, they are *5 Drivers and 20 Riders*, *10 Drivers and 40 Riders*, and *15 Drivers and 60 Riders* respectively. Different rows are presenting different schedule strategies, they are *Random Assignment*, *Nearest Tasks First*, *Shortest Tasks Queue First*, and *Stochastic Optimization (Our approach)* respectively.

initialization step. This forms the current schedule solution S at the first iteration. For every new iteration, given to any randomly sampled schedule solution S as the current status, we propose a new schedule solution as S' through three types of move strategies:

- *Reorder a Task.* Randomly select a task i and task j from a driver, exchange the order of these two tasks.
- *Replace a Task.*: Randomly select a task i from driver x and a random task j from driver y, then assign task i to driver y by inserting the task i before the task j.
- *Exchange a Task.*: Randomly select a task i from driver x and task j from driver y, exchange these two tasks.

For a proposed update of schedule S', in the formulation of the simulated annealing approach proposed by Kirkpatrick et al. [28], the acceptance probability function $\Pr(S'|S)$ is:

$$\Pr(S'|S) = \min(1, \frac{f(S')}{f(S)}), \tag{5}$$

where $f(S)$ is a Boltzmann-like objective function related to a Metropolis-Hastings state searching step [29]:

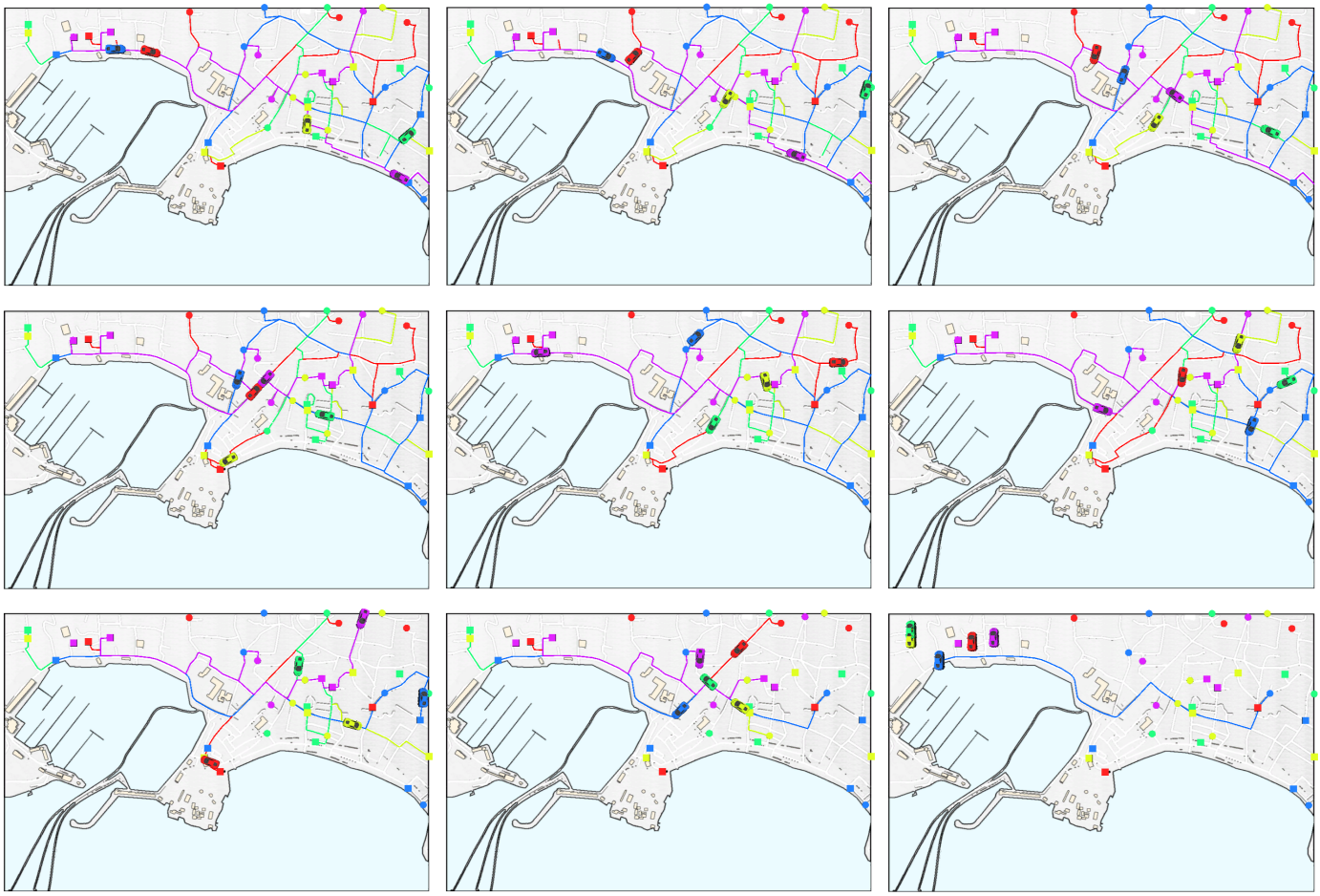


Fig. 6: Uber Schedule Simulation. This figure shows the animation of the Uber scheduling simulation process. In this scenario, there are 5 Ubers drivers assigned with 20 ride-sharing tasks. Every Uber car is moving along the route on the map and navigated by the schedule we have optimized. After the Uber driver finished each task, the route for that task will disappear while only keeping the origins (circles) and destinations (squares). As we can see, almost all of the Ubers finish their tasks at approximately the same time moment. This validates the effect that we minimize the max working time for all of the Ubers.

$$f(S) = \exp(-\frac{1}{t}C_{\text{total}}(S)), \tag{6}$$

where t is the temperature parameter of simulated annealing, which decreases gradually throughout the optimization. As the temperature t decreases over iterations, the optimizer becomes less aggressive and more greedy. By the end, the temperature drops to a low value near zero, the optimizer tends to accept better solutions only. We empirically use temperature $t = 1.0$ at the beginning of optimization and decrease by 0.2 every 1000 iterations until it reaches zero or terminated if the total cost change is smaller than 3% over the past 500 iterations.

III. Experiments

A. Implementations

We have implemented the Make Uber Faster system using a computer graphics engine called Unity 3D with the 2019 version. The simulated annealing algorithms and animations are implemented in the C# programming language. The hardware configurations include Intel Core i5 CPU, 32GB DDR4 RAM, and NVIDIA GeForce GTX 1650 4GB GDDR6 Graphics Card. All programs are running on CPU multi-threads except the graphics rendering are rendered on GPUs.

B. Changing Parameter Settings

In this experiment, we have tested our approach through different tasks settings and compared our optimization approach with different Uber scheduling strategies. As shown in Figure 5, Uber scheduling solutions are presented. Among these results, three columns represent three different parameter settings on the number of drivers and number of riders, they are *5 Drivers and 20 Riders*, *10 Drivers and 40 Riders*, and *15 Drivers and 60 Riders* respectively. Different rows are presenting our approach compared with different existing heuristics-based strategies, they are *Random Assignment*, *Nearest Tasks First*, *Shortest Tasks Queue First*, and *Stochastic Optimization (Our approach)* on four different rows respectively. Each column is initialized with the same random tasks and drivers.

	Max Working Time			Avg Waiting Time		
Colors	5 driver 20 rider		10 driver 40 rider		15 driver 60 rider	
Random	47.27	43.38	57.38	11.58	12.55	13.44
Nearest	67.24	69.59	52.13	15.23	14.66	12.13
Shortest	32.48	30.46	31.96	9.82	8.21	9.13
Optimize	25.89	18.94	20.58	6.83	5.08	4.74

TABLE I: Statistical results. This table demonstrates the statistical results including the max work time and average wait time corresponding to the Uber schedules generated with different parameter settings under different schedule strategies as shown in Figure 5. Among these numbers, three colors represent three different parameter settings, they are *5 Drivers and 20 Riders*, *10 Drivers and 40 Riders*, and *15 Drivers and 60 Riders* respectively. Four rows are presenting four different schedule strategies, they are *Random Assignment*, *Nearest Tasks First*, *Shortest Tasks Queue First*, and *Stochastic Optimization (Our approach)* respectively.

As for the *Nearest Tasks First* strategy, all tasks are assigned to each Uber driver according to whose origins are closed to that driver. Actually, this strategy is the most straightforward method and probably the current Uber app seems to use this strategy [30]. However, as shown in this figure, this strategy seriously depends on whether the Uber drivers are evenly distributed on the map. For example, from the figure, we can infer that it works much better in the *5 Drivers and 20 Riders* case than *10 Drivers and 40 Riders* as the Ubers are scattered more uniformly in the previous one. On average, random strategy even performs better than this strategy.

As for the *Shortest Tasks Queue First* strategy, all tasks are assigned to each Uber driver according to whose assigned tasks' total amount of duration is the minimum one, namely, with the shortest task queue. Actually, this strategy approximates the optimal solution the best. As shown in this figure, compared with other strategies such as the *Random Assignment* and the *Nearest Tasks First*, this strategy always results in a shorter task queue and averages the workloads among the drivers much better. However, when it is compared to the optimization strategy, this strategy is still less perfect.

The *Stochastic Optimization* strategy presented here is using our proposed approach to optimize the Uber schedule within 20000 iterations. As the CPU computation rate is fast, the whole optimization process only takes 25 to 50 secs to achieve an acceptable optimal solution. During the optimization, we initialize the first solution using the *Random Assignment* strategy. As shown in the result, our approach overperforms all other approaches. The rider's max working time and rider's average waiting time for changing parameters shown in Figure 5 are shown in TABLE 1 and the units are minutes. In order to statistically prove our approach overperforms others, we have repeated these experiments 20 times for each setting using random initialization of task locations and Uber locations. Statistical analysis will be presented in the next section.

C. Uber Schedule Simulation

In the end, we simulate the animation of the Ubers according to the schedule that we have optimized. As shown in Figure 6, there are 5 Ubers assigned with 20 tasks. Every Uber is moving along the route on the map and navigated by the schedule we optimized. After Uber finished each task, the route for that task will disappear while only keeping the origins and destinations. As we can see, almost all of the Ubers finish their tasks at approximately the same time moment. This validates the effect that we want to minimize the max working time for all of the Ubers. To avoid that any Uber being assigned too much work while others have finished already.

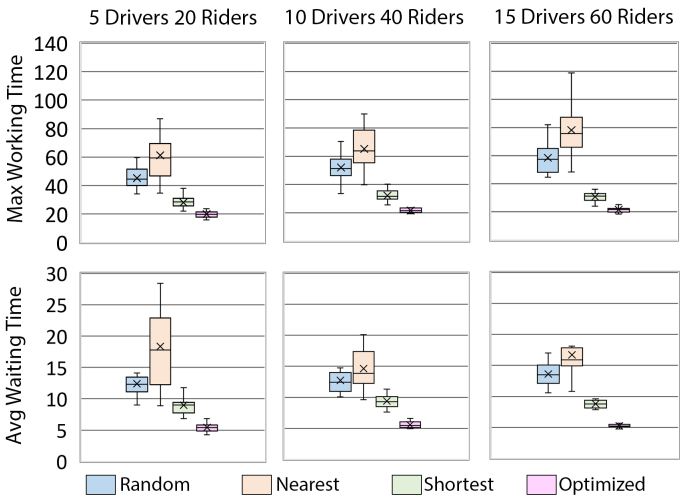


Fig. 7: Numerical experiments results. These box plots are depicting the drivers' max working time (first row) and riders' average waiting time (second row) for each simulation where the units are minutes. Three columns presents three different parameter settings including *5 Drivers and 20 Riders*, *10 Drivers and 40 Riders*, and *15 Drivers and 60 Riders*. Four colors presents four different schedule strategies including *Random Assignment*, *Nearest Tasks First*, *Shortest Tasks Queue First*, and *Stochastic Optimization (Our approach)*.

IV. Results and Discussions

As shown in Figure 7, the simulation results of the numerical experiments are presented. By repeating these experiments 20 times for each task number setting, we randomly initialize the task locations and Uber's initial locations and compare different settings and strategies like what we have shown in Figure 5. The box plots are the max working time and average waiting time for each simulation where the units are minutes. As we can see, there is an obvious pattern that generally *Random Assignment* performs better than the *Nearest Tasks First* and *Shortest Tasks Queue First* strategy performs better than the *Random Assignment* strategy. Therefore, if we statically show that our approach performs significantly better than the *Shortest Tasks Queue First* strategy, we can confirm that our approach overperforms all. According to unpaired T-tests with $\alpha = 0.05$, we have lower bounds and uppers bounds of the confidence interval for the max working time given three settings are [3.2, 3.8], [8.4, 12.4], and [7.4, 11.0]; for the average waiting time are [2.8, 4.2], [3.3, 4.4], and [3.2, 3.8]. Therefore, all confidence intervals do not cover 0. So, ours performs much better than *Shortest Tasks Queue First*.

V. Conclusion

In this paper, we present Make Uber Faster: a stochastic optimization approach to automatically improve the Uber scheduling strategy. By taking advantage of the OpenStreetMap (OSM) dataset, we construct navigation graphs automatically using real-world data. Then we generate all-pairs shortest paths on OSM to simulate Ubers' navigation routes. We optimize the assignment of the rideshare tasks to Uber drivers by minimizing two costs: the Driver Cost which is used for balancing the amount of work assigned to different Uber drivers and the Rider Cost which is used for minimizing the riders' average waiting time. We simulate the optimized Uber schedules using OpenStreetMap data to visually validate the efficacy of our proposed optimization approach. At the same time, we conduct a series of numerical experiments by comparing different scheduling strategies including *Random Assignment* strategy, *Nearest Tasks First* strategy, and the *Shortest Tasks Queue First* strategy with our *Stochastic Optimization* strategy. According to the statistical analysis using unpaired T-tests, we have %95 confidence to claim that our strategy significantly overperforms other strategies.

Limitations and Future Works. However, according to the explanation of our algorithm, there are lots of assumptions made for simplifying the problem statements such as the speed is considered to be always constant so that the task duration is proportional to the distance and the assumption that all tasks assigned at the same time are reaming unchanged. However, in reality, it is important to consider some other external conditions, such as extreme weather conditions, extreme traffic conditions, and human-factors-related conditions, etc. As future work, more dynamic conditions will be included into our simulation and optimization platform. Interactions between the user and the platform will be allowed for interactive simulation and optimization. At the same time, personalized simulation conditions can also be considered during the optimization process such as personalized drivers and riders settings. We believe our optimization approach can be easily extended with such considerations and give acceptable results. Our work will attract more researchers to explore how to use our approach to optimize rideshare scheduling problems within a realistic virtual simulation environment.

References

[1] D. Applegate and W. Cook, "A computational study of the job-shop scheduling problem," *ORSA Journal on computing*, vol. 3, no. 2, pp. 149–156, 1991.

[2] J. Błażewicz, W. Domschke, and E. Pesch, "The job shop scheduling problem: Conventional and new solution techniques," *European journal of operational research*, vol. 93, no. 1, pp. 1–33, 1996.

[3] L. Davis *et al.*, "Job shop scheduling with genetic algorithms," in *Proceedings of an international conference on genetic algorithms and their applications*, vol. 140, 1985.

[4] J. F. Gonçalves, J. J. de Magalhães Mendes, and M. G. Resende, "A hybrid genetic algorithm for the job shop scheduling problem," *European journal of operational research*, vol. 167, no. 1, pp. 77–95, 2005.

[5] F. Pezzella, G. Morganti, and G. Ciaschetti, "A genetic algorithm for the flexible job-shop scheduling problem," *Computers & operations research*, vol. 35, no. 10, pp. 3202–3212, 2008.

[6] P. Brucker, B. Jurisch, and B. Sievers, "A branch and bound algorithm for the job-shop scheduling problem," *Discrete applied mathematics*, vol. 49, no. 1-3, pp. 107–127, 1994.

[7] P. Brucker and R. Schlie, "Job-shop scheduling with multi-purpose machines," *Computing*, vol. 45, no. 4, pp. 369–375, 1990.

[8] M. Dell'Amico and M. Trubian, "Applying tabu search to the job-shop scheduling problem," *Annals of Operations research*, vol. 41, no. 3, pp. 231–252, 1993.

[9] F. A. Omara and M. M. Arafa, "Genetic algorithms for task scheduling problem," in *Foundations of Computational Intelligence Volume 3*. Springer, 2009, pp. 479–507.

[10] L. N. Van Wassenhove and L. F. Gelders, "Solving a bicriterion scheduling problem," *European Journal of Operational Research*, vol. 4, no. 1, pp. 42–48, 1980.

[11] J. H. May, W. E. Spangler, D. P. Strum, and L. G. Vargas, "The surgical scheduling problem: Current research and future opportunities," *Production and Operations Management*, vol. 20, no. 3, pp. 392–405, 2011.

[12] S. Panwalkar, R. Dudek, and M. Smith, "Sequencing research and the industrial scheduling problem," in *Symposium on the Theory of Scheduling and its Applications*. Springer, 1973, pp. 29–38.

[13] S. E. Elmaghraby, "The economic lot scheduling problem (elsp): review and extensions," *Management Science*, vol. 24, no. 6, pp. 587–598, 1978.

[14] T. Ishihara and H. Yasuura, "Voltage scheduling problem for dynamically variable voltage processors," in *Proceedings of the 1998 international symposium on Low power electronics and design*, 1998, pp. 197–202.

[15] R. C. Grinold, "The payment scheduling problem," *Naval Research Logistics Quarterly*, vol. 19, no. 1, pp. 123–136, 1972.

[16] F. Glover and C. McMillan, "The general employee scheduling problem. an integration of ms and ai," *Computers & operations research*, vol. 13, no. 5, pp. 563–573, 1986.

[17] H. V. D. Parunak, "Characterizing the manufacturing scheduling problem," *Journal of manufacturing systems*, vol. 10, no. 3, pp. 241–259, 1991.

[18] D. Teodorović and M. Dell'Orco, "Mitigating traffic congestion: solving the ride-matching problem by bee colony optimization," *Transportation Planning and Technology*, vol. 31, no. 2, pp. 135–152, 2008.

[19] N. Agatz, A. L. Erera, M. W. Savelsbergh, and X. Wang, "Dynamic ride-sharing: A simulation study in metro atlanta," *Procedia-Social and Behavioral Sciences*, vol. 17, pp. 532–550, 2011.

[20] S. Banerjee, C. Riquelme, and R. Johari, "Pricing in ride-share platforms: A queueing-theoretic approach," *Available at SSRN 2568258*, 2015.

[21] X. Chen, F. Miao, G. J. Pappas, and V. Preciado, "Hierarchical data-driven vehicle dispatch and ride-sharing," in *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*. IEEE, 2017, pp. 4458–4463.

[22] X. Wang, N. Agatz, and A. Erera, "Stable matching for dynamic ride-sharing systems," *Transportation Science*, vol. 52, no. 4, pp. 850–867, 2018.

[23] E. J. Gonzales, C. Sipetas, J. Italiano *et al.*, "Optimizing ada para-transit operation with taxis and ride share programs," University of Massachusetts at Amherst, Tech. Rep., 2019.

[24] S. Silwal, V. Raychoudhury, S. Saha, and M. O. Gani, "A dynamic taxi ride sharing system using particle swarm optimization," in *2020 IEEE 17th International Conference on Mobile Ad Hoc and Sensor Systems (MASS)*. IEEE, 2020, pp. 112–120.

[25] N. A. Dehkordi, "Simulation-based optimization frameworks for dynamic ride-sharing," Ph.D. dissertation, Université de Lyon, 2020.

[26] P. Mooney, M. Minghini *et al.*, "A review of openstreetmap data," 2017.

[27] R. W. Floyd, "Algorithm 97: shortest path," *Communications of the ACM*, vol. 5, no. 6, p. 345, 1962.

[28] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *science*, vol. 220, no. 4598, pp. 671–680, 1983.

[29] S. Chib and E. Greenberg, "Understanding the metropolis-hastings algorithm," *The american statistician*, vol. 49, no. 4, pp. 327–335, 1995.

[30] Uber, "How to schedule a ride — uber," https://www.youtube.com/watch?v=aVGw-QkY20c, Jan 2017.