Effective and Efficient: Large-scale Dynamic City Express

Siyuan Zhang†, Lu Qin‡, Yu Zheng*, Hong Cheng†

†*The Chinese University of Hong Kong, China*
‡*Centre for QCIS, FEIT, University of Technology, Sydney, Australia*
**Microsoft Research, Beijing, China*
{syzhang,hcheng}@se.cuhk.edu.hk, Lu.Qin@uts.edu.au, yuzheng@microsoft.com

ABSTRACT

City express services are in great demand in recent years. However, the current city express system is found to be unsatisfactory for both the service providers and customers. In this paper, we are the first to systematically study the large-scale dynamic city express problem. We aim to increase both the effectiveness and the efficiency of the scheduling algorithm. To improve the effectiveness, we adopt a batch assignment strategy that computes the pickup-delivery routes for a group of requests received in a short period rather than dealing with each request individually. To improve the efficiency, we design a two-level priority queue structure to reduce redundant shortest distance calculation and repeated candidate generation. We develop a simulation system and conduct extensive performance studies in the real road network of Beijing city. The experimental results demonstrate the high effectiveness and efficiency of our algorithm.

Categories and Subject Descriptors

H.2.8 [**DATABASE MANAGEMENT**]: Database Applications—*Spatial databases and GIS*; H.4.2 [**INFORMATION SYSTEMS APPLICATIONS**]: Types of Systems—*Logistics*

Keywords

City express service; logistics; batch assignment

1. INTRODUCTION

With the development of logistics industry and the rise of E-commerce, city express services have become increasingly popular in recent years [7]. We illustrate how current city express systems work in Fig. 1: A city is divided into several regions (e.g., R_1 and R_2), each of which covers some streets and neighborhoods. A transit station is built in a region to temporarily store the parcels received in the region (e.g., ts_1 in R_1). The received parcels in a transit station are further organized into groups according to their destinations. Each group of parcels will be sent to a corresponding transit station by trucks regularly (e.g., from ts_1 to ts_2). In each region, there are a team of couriers delivering parcels to and receiving parcels from specific locations in the region. When a truck carrying parcels arrives at a transit station, each courier will send a portion of these parcels to their final destinations. Before departing from the transit station, they will pre-compute the delivery routes (e.g., the blue lines in Fig. 1). During the delivery, each courier can receive pickup requests (e.g., r_5, r_6 and r_7) from a central dispatch system or directly from end users. A pickup request is associated with a location and a deadline of pickup time. A courier may change the originally planned route to fetch the new parcels, or decline the pickup request due to the capacity constraint or schedule constraint. All the couriers are required to return to their own transit station by some specific time (so as to fit the schedule of trucks that travel between transit stations regularly), or when fully loaded.

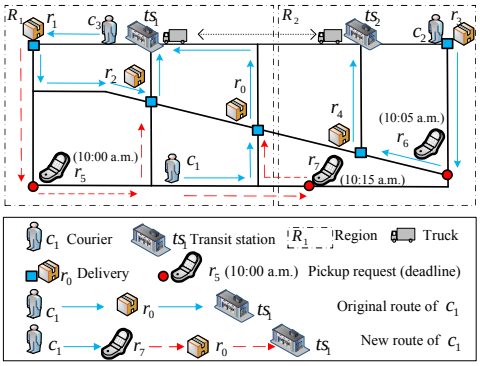


Figure 1: Dynamic City Express

Current central dispatch systems usually adopt a *first-come, first-served* (FCFS) strategy in which early requests have higher priority to be assigned. However, using request arrival time as the first priority may result in inferior scheduling result. For example, if an early request has a large incurred distance to be served, it will prevent the courier from serving more incoming requests which may have smaller incurred distance. On the other hand, most existing solutions for vehicle routing with time window are either based on the static assumption [1, 6] or only applied to Euclidean space [3, 4]. Thus they cannot be applied to solve the city express problem with dynamic requests. In this paper, we study the dynamic city express problem and aim to design a batch assignment solution. After collecting pickup requests from customers within a short period, our solution processes the requests using a *shortest incurred distance first* (SIDF) strategy. Then the system sends the updated schedules to all the couriers and the confirmation or decline messages to the customers.

Contributions. We make the following contributions in this paper. First, we are the first to study the dynamic city express problem which involves the scheduling of multiple couriers to serve pickup requests and delivery tasks in real time. Second, we propose an effective batch solution which processes a batch of requests in the

SIGSPATIAL'15, November 03-06, 2015, Bellevue, WA, USA
 ISBN 978-1-4503-3967-4/15/11 ...$15.00
DOI: http://dx.doi.org/10.1145/2820783.2820838

increasing order of their incurred distance to the courier's planned route. Third, we further improve the computational efficiency of our batch solution by designing a two-level priority queue structure, which can reduce redundant shortest distance computation and avoid repeated candidate generation. Fourth, we conduct extensive performance studies in the real Beijing road network and demonstrate the high effectiveness and efficiency of our solution.

Outline. Section 2 provides the preliminaries, formally defines the dynamic city express problem, and shows the complexity of the problem. Section 3 describes our batch solution. Section 4 presents experimental results on a real city road network. Section 5 concludes the paper.

2. PROBLEM FORMULATION

We model a road network as a directed weighted graph $G(V, E)$, where V is a set of nodes (road intersections), and E is a set of edges (road segments). Each edge $(u, v) \in E$ is associated with a positive weight $w(u, v)$ denoting the time to travel along the edge. Given two nodes u and v in G, we denote the shortest travel time from u to v as $\mathsf{cost}(u, v)$.

Definition 2.1: (Request) Given a road network $G(V, E)$, a *request* is denoted as $r = (l, d)$, where $l(r)$ is the location of the request that lies in the road network G, and $d(r)$ is the deadline to pick up the parcel in the request. □

Each courier also has parcels for delivery on board. We use the same form for a pickup request to represent a delivery task.

Definition 2.2: (Schedule) Let $C = \{c_1, c_2, \cdots, c_n\}$ be the set of couriers. For a courier $c_i \in C$, a schedule for c_i, denoted as $S_i = (r_{i,1}, r_{i,2}, \cdots, r_{i,m_i})$, is a sequence of unserved tasks, such that if following the sequence to pick up/deliver the parcels, the courier can (1) arrive at the location $l(r_{i,j})$ before the deadline $d(r_{i,j})$ for every $1 \leq j \leq m_i$, and (2) return to the transit station after serving r_{i,m_i} within a fixed maximum travel time after setting off. □

We use $l(r_{i,0})$ to denote the current location of the courier c_i, and $l(r_{i,m_{i+1}})$ to denote the location of the transit station that courier c_i will return at the end of his schedule.

Dynamic City Express Problem. Given a set of n couriers, a set of delivery tasks at different transit stations, and a stream of pickup requests, the dynamic city express problem aims to update the schedule of each courier dynamically as new pickup requests stream in, such that (1) all delivery tasks are satisfied, and (2) the pickup requests are satisfied as many as possible.

In our problem, the initial schedules and routes for couriers are pre-computed to satisfy all the delivery tasks, and they may be updated dynamically to satisfy the incoming pickup requests.

The following lemma shows the complexity of the problem when all requests are given in advance. In the dynamic case, the problem becomes even more difficult to handle.

Lemma 2.1: *Given the set of couriers C and all requests in advance, the problem to decide whether a certain ratio P of requests can be satisfied by the couriers is an NP-complete problem.* □

The proof is omitted due to space limit.

3. OUR BATCH ASSIGNMENT SOLUTION

In this section, we describe our proposed algorithm SIDF, which uses a *shortest incurred distance first* strategy to assign requests in a batch. Our method works as follows. Given a set of requests $R = \{r_1, r_2, ..., r_m\}$ received within a short time, we first generate candidate couriers for each request, that is, couriers who can possibly serve the request. We then calculate the incurred distance of each request and assign the requests to couriers in ascending order of their incurred distance. A two-level priority queue structure is designed to speed up the assignment process. Finally, we update all the couriers' schedules by inserting the accepted requests, and decline those, if any, that cannot be handled by the couriers.

3.1 Candidate Courier Generation

As the first step, we compute the set of candidate couriers to serve a new request r. A courier c_i can possibly arrive at the location $l(r)$ before the deadline $d(r)$ from his current location, if $\mathsf{cost}(l(r_{i,0}), l(r)) \leq d(r) - t_{cur}$ holds, where t_{cur} denotes the current time. However, computing $\mathsf{cost}(l(r_{i,0}), l(r))$ is very costly. Thus, we define a lower bound of travel time, denoted as $\underline{\mathsf{cost}}(l(r_{i,0}), l(r))$, which can be computed with a light cost. With the travel cost lower bound, we define the candidate set $\mathsf{cand}(r)$ for a request r as:

$$\mathsf{cand}(r) = \{c_i | \underline{\mathsf{cost}}(l(r_{i,0}), l(r)) \leq d(r) - t_{cur}\}. \quad (1)$$

To compute $\underline{\mathsf{cost}}(l(r_{i,0}), l(r))$, we build a Network Voronoi Diagram (NVD) of the road network G, and treat the transit stations as generators. For each location l in G, we use $ts(l)$ to denote the generator for the region that l lies in. For each node $v \in V$, we precompute $\mathsf{cost}(v, ts(v))$ and $\mathsf{cost}(ts(v), v)$. For any two generators ts_i and ts_j, we precompute $\mathsf{cost}(ts_i, ts_j)$. We also precompute the radius of each generator ts_i, denoted as $\mathsf{radius}(ts_i)$, which is the maximum cost from ts_i to any node in the Voronoi region of ts_i. With the NVD index, given any two locations l_i and l_j in G, the lower bound of $\mathsf{cost}(l_i, l_j)$ can be calculated as:

$$\begin{aligned}\underline{\mathsf{cost}}(l_i, l_j) = &\max\{0, \mathsf{cost}(ts(l_i), ts(l_j)) \\ &- \mathsf{cost}(ts(l_i), l_i) - \mathsf{cost}(l_j, ts(l_j))\}.\end{aligned} \quad (2)$$

Obviously, $\underline{\mathsf{cost}}(l_i, l_j)$ can be calculated in constant time. Note that we can update the NVD index periodically to handle the dynamic update of travel cost in road network.

In order to efficiently identify $\mathsf{cand}(r)$, we first compute the set of candidate Voronoi regions $\mathsf{candts}(r)$ as:

$$\mathsf{candts}(r) = \{ts_j | \underline{\mathsf{cost}}(l(ts_j), l(r)) - \mathsf{radius}(ts_j) \leq d(r) - t_{cur}\}.$$

After computing $\mathsf{candts}(r)$, for each $ts_j \in \mathsf{candts}(r)$, we enumerate all couriers c_i that lie in the Voronoi region of ts_j and add c_i into $\mathsf{cand}(r)$ if $\underline{\mathsf{cost}}(l(r_{i,0}), l(r)) \leq d(r) - t_{cur}$.

Besides, we maintain a candidate request set for each courier c_i, which is a set of requests that courier c_i can reach before their deadlines. $\mathsf{cand}(c_i)$ can be considered as an inverse set of $\mathsf{cand}(r)$, thus, for each $c_i \in \mathsf{cand}(r)$, we simply add r into $\mathsf{cand}(c_i)$.

3.2 Incurred Distance Calculation

For each courier $c_i \in \mathsf{cand}(r)$, we insert request r into the schedule S_i of c_i and calculate the incurred distance due to the insertion. To avoid exponential search space, we do not consider changing the order of existing tasks in a schedule to accommodate r. In the schedule S_i, we denote any two consecutive requests $r_{i,j}$ and $r_{i,j+1}$ as *segment j of S_i* ($0 \leq j \leq m_i$). For a segment j in S_i, it is *valid* w.r.t. request r iff after inserting r between $r_{i,j}$ and $r_{i,j+1}$ in S_i, the deadlines of all requests in the new S_i can be satisfied. The incurred distance of this insertion is computed as:

$$\begin{aligned}\Delta\mathsf{dist}_j(r, S_i) = &\mathsf{cost}(l(r_{i,j}), l(r)) + \mathsf{cost}(l(r), l(r_{i,j+1})) \\ &- \mathsf{cost}(l(r_{i,j}), l(r_{i,j+1})).\end{aligned} \quad (3)$$

Based on this, we define the minimum incurred distance to serve a request r as:

$$\Delta\mathsf{dist}(r) = \min_{c_i \in \mathsf{cand}(r); 0 \leq j \leq m_i;\ \text{segment } j \text{ of } S_i \text{ is valid w.r.t. } r} \Delta\mathsf{dist}_j(r, S_i). \quad (4)$$

According to Eq. 3, calculating the incurred distance $\Delta\mathsf{dist}_j(r, S_i)$ involves calculating $\mathsf{cost}(l(r_{i,j}), l(r))$ and $\mathsf{cost}(l(r), l(r_{i,j+1}))$ using Dijkstra's algorithm (or its variant A*) [2], which is costly in a large road network. In order to minimize the number of exact shortest path computations, we adopt a lazy path computation strategy which computes the exact incurred distance for a segment in ascending order of its lower bound. The rationale is that if the lower bound for a segment is large, we can prune it without computing the exact incurred distance. The lower bound of $\Delta\mathsf{dist}_j(r, S_i)$ is defined as:

$$\underline{\Delta\mathsf{dist}}_j(r, S_i) = \max\{0, \underline{\mathsf{cost}}(l(r_{i,j}), l(r)) + \underline{\mathsf{cost}}(l(r), l(r_{i,j+1})) - \mathsf{cost}(l(r_{i,j}), l(r_{i,j+1}))\},$$

where $\underline{\mathsf{cost}}(l, l')$ for any two locations l and l' can be calculated by Eq. 2 using the NVD index. A priority queue is used to maintain the candidate segments and perform the incurred distance computation in ascending order of the lower bound value.

3.3 Request Assignment and Schedule Update

Given a set of requests $R = \{r_1, r_2, ..., r_m\}$, we compute the incurred distance for each request, and assign them to couriers using a *shortest incurred distance first* strategy. We use a priority queue to maintain the requests, where a request with a smaller incurred distance has higher priority. Then we pop out the top request r from the queue. Suppose the incurred distance of r is given by $\Delta\mathsf{dist}_j(r, S_i)$, we insert r into segment j of schedule S_i. After the assignment of r, the incurred distance of each unserved request $r' \in \mathsf{cand}(c_i)$ needs to be updated due to new spatio-temporal constraints on the schedule S_i caused by the insertion of r. In the next round, the top request with the minimum incurred distance will be popped out from the the queue and assigned similarly. The assignment process terminates when the requests in the priority queue are assigned as far as possible. The remaining unassigned requests are declined.

3.4 Efficiency Implementation

To improve the efficiency, we design a two-level priority queue structure that maintains and reuses intermediate assignments computed in the process. Fig. 2 depicts the two-level priority queue. An entry takes the form $(r, c_i, r_{i,j}, r_{i,j+1}, \Delta\mathsf{dist}_j(r, S_i), \mathsf{flag})$, which stands for inserting r between $r_{i,j}$ and $r_{i,j+1}$ with incurred distance $\Delta\mathsf{dist}_j(r, S_i)$. flag indicates whether the $\Delta\mathsf{dist}_j(r, S_i)$ is an exact value or a lower bound, which can be easily computed from Eq. 3 with NVD index. At the bottom level, for each courier c_i, we maintain a local priority queue $\mathcal{H}_i$ that keeps all the candidate requests $\mathsf{cand}(c_i)$ and their incurred distance or lower bound; and at the top level, we maintain a global priority queue $\mathcal{H}^g$ that keeps the topmost entries popped from each local priority queue. With the two-level priority queue structure, after a new request r_j is assigned to a courier c_i, we only need to update the local priority queue $\mathcal{H}_i$ and the global priority queue incrementally. For other couriers that also contain r_j in their local priority queues, they can discard r_j in a lazy manner. In this way, the total number of shortest distance computation can be largely reduced.

Analysis. The two-level priority queue structure accelerates the assignment process from two aspects: 1) it only computes the exact incurred distance of an entry when it is popped out from $\mathcal{H}^g$. For example, we can avoid computing the exact incurred distance of r_2 because the lower bound of the incurred distance of r_2 is larger than that of r_1 in Fig. 2. In contrast, if not using the two-level priority queue structure, we have to compute the incurred distance of all requests before assignment. 2) It uses the information from all local priority queues to avoid repeated exact distance computation. For example, in Fig. 2, the top valid entry of $\mathcal{H}_1$ contains a possible assignment with exact incurred distance w.r.t. c_1, and we can use it for the assignment of r_5 after it is pushed into $\mathcal{H}^g$. Note that using the two-level priority queue does not affect the assignments, that is, requests are still assigned according to the shortest incurred distance first criterion. But it can substantially improve the computational efficiency.

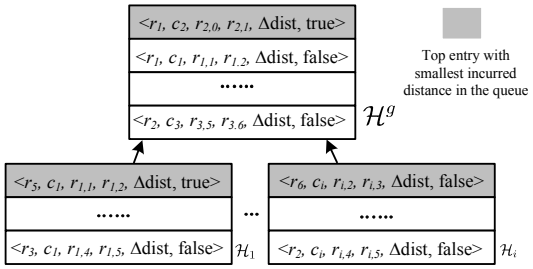


Figure 2: Global Queue and Local Queues

4. PERFORMANCE STUDIES

4.1 Experimental Setup

Data set. We perform experiments in a real road network in Beijing city. We consider a 15km×5km area that lies between northeastern 4th ring road and 5th ring road of Beijing. The road network contains $8,840$ nodes and $11,331$ edges.

Simulation with Parameter Settings. We develop a simulation system to simulate the city express service process as follows. We choose 7 nodes as transit stations by performing the k-medoids algorithm on all the nodes in the road network ($k = 7$). Then we build a network Voronoi diagram with the transit stations as centers. The courier number in a transit station is proportional to the number of nodes in the corresponding Voronoi region. The default value of the total number of couriers is 500. We assume that requests are generated on all nodes in our experiments. Similar to most queuing systems [5], we consider the arrival of a pickup request on a node as a Poisson process with intensity λ, which denotes the average number of pickup requests arriving per hour on the node. The average intensity $\overline{\lambda}$ is $0.6/hr$. A total of 10,800 pickup requests are generated in a simulation for 2 hours. We set the maximum time to confirm a request to be 15 minutes and the deadline for picking up a request to be 30 minutes by default.

Measurements. We test both the effectiveness and the efficiency of our proposed algorithm. All the experimental results are based on a simulation for 2 hours of the city express process using our simulation system. For effectiveness testing, we compare the satisfaction ratio SR of different algorithms, which is computed as:

$$\mathsf{SR} = \frac{\text{\# accepted pickup requests}}{\text{\# issued pickup requests}}$$

We also compare the average incurred distance AID of different algorithms, which is computed as:

$$\mathsf{AID} = \sum_{r \text{ is accepted}} \Delta\mathsf{dist}(r)/\text{\# accepted pickup requests}$$

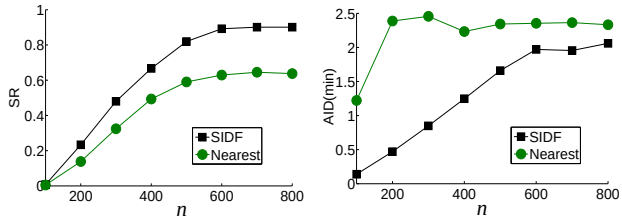
The average incurred distance AID is used to measure the average increased cost to serve an accepted pickup request.

For efficiency testing, we compare the average processing time per request, which is defined as the time spent on computing the schedules for all requests within the 2 hours divided by the number of issued requests. We also compare the average number of accessed nodes to process each request.

Comparison Algorithm. For both effectiveness and efficiency testing, we compare our algorithm SIDF with a baseline method called Nearest. Nearest adopts a first-come, first-served strategy to handle requests, and assigns a request r to its nearest courier c_i that can serve r. If no such courier is found, the request is rejected. For efficiency testing and comparison, we also implement a basic version of our algorithm, denoted as SIDF(Basic), which does not use the two-level priority queue for efficient computation.

4.2 Effectiveness Testing

(Exp-1: Vary n). In this experiment, we vary the number of couriers n from 100 to 800. The experimental results for SR and AID are shown in Fig. 3 (a) and Fig. 3 (b) respectively. SIDF increases SR by 30% on average compared to Nearest. Remarkably, SIDF can satisfy more than 60% of the requests with 400 couriers while Nearest needs 800 couriers to reach the same SR. In other words, our algorithm can potentially help an express service company to save substantial operational cost (i.e., 50%) while achieving the same satisfaction ratio (i.e., SR = 60%). Regarding AID, SIDF has smaller AID than Nearest under all n values, which demonstrates that SIDF can effectively reduce the incurred distance for serving pickup requests by using the shortest incurred distance first strategy and batch processing mode.



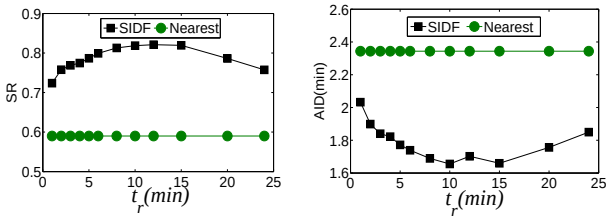
(a) SR (b) AID

Figure 3: Vary n (Effectiveness)

(Exp-2: Vary t_r). We vary t_r, the maximum time to confirm a request, from 1 to 24 minutes and compare SR and AID for Nearest and SIDF. The experimental results for SR and AID are shown in Fig. 4 (a) and Fig. 4 (b) respectively. As Nearest is independent of t_r, both SR and AID remain unchanged for Nearest when we vary t_r. For SR, SIDF is better than Nearest for all t_r values. When t_r increases from 1 to 12, SR for SIDF increases. This is because the batch assignment strategy in SIDF can have a larger room for improvement when t_r increases. However, when t_r further increases, SR for SIDF starts to decrease. This is because when t_r approaches the deadline for picking up a request (i.e., $d(r) = 30$ minutes by default), the time left for couriers to satisfy early arrival requests becomes short. Therefore, the overall SR decreases when t_r increases. For AID, SIDF has a smaller average incurred distance under all t_r values. The AID of SIDF decreases when t_r increases from 1 to 10, but begins to increase when t_r is larger than 15. This is because when t_r is too large, SIDF has to reject some requests that have a small incurred distance with a strict deadline constraint (i.e., the requests issued at the beginning of t_r period).

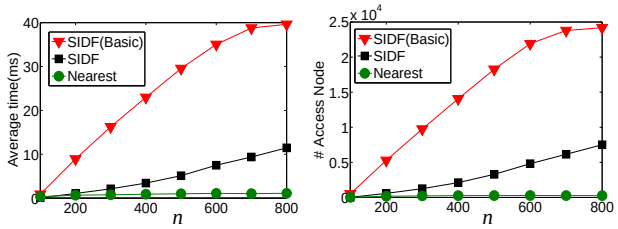
4.3 Efficiency Testing

(Exp-3: Vary n). We compare the average processing time and average number of accessed nodes per request by SIDF, SIDF(Basic) and Nearest. The results for varying n are shown in Fig. 5 (a) and Fig. 5 (b) respectively. SIDF is six times faster than SIDF(Basic) on average, which demonstrates the advantage of the two-level priority queue in reducing the number of exact incurred distance calculation. SIDF can process a request in less than 15ms and thus is suitable for realtime applications. The results for the number of accessed nodes follows the same trend as the average processing



(a) SR (b) AID

Figure 4: Vary t_r in minutes (Effectiveness)



(a) Average Time (b) # Accessed Node

Figure 5: Vary n (Efficiency)

time. Similar results are observed when varying t_r and omitted due to space limit.

5. CONCLUSION

In this paper, we propose a solution to the scheduling problem in dynamic city express. We also develop a simulation platform to confirm both the effectiveness and efficiency of our solution. Our solution increases the satisfaction ratio of pickup requests compared to existing solutions and enjoys high efficiency as well. Moreover, the developed simulation platform can be used to estimate the number of couriers a city express company needs to achieve a certain satisfaction ratio in urban area. For instance, by estimating the satisfaction ratio under different courier numbers in the simulation, we can find that at least 7 couriers/km^2 (i.e., 500 couriers/75 km^2) are needed to keep the satisfaction ratio above 80%, or equivalently, serve 8,640 pickup requests in 2 hours.

6. ACKNOWLEDGMENTS

This work is supported by a Microsoft Research grant on urban informatics, Hong Kong RGC GRF Project No. CUHK 411211, and CUHK Direct Grant No. 4055015 and 4055048. Lu Qin is supported by ARC DE140100999.

7. REFERENCES

[1] O. Bräysy and M. Gendreau. Vehicle routing problem with time windows, part i: Route construction and local search algorithms. *Transportation science*, 39(1), 2005.

[2] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms (3. ed.)*. MIT Press, 2009.

[3] M. Gendreau, F. Guertin, J.-Y. Potvin, and E. Taillard. Parallel tabu search for real-time vehicle routing and dispatching. *Transportation science*, 33(4), 1999.

[4] V. Pillac, M. Gendreau, C. Guéret, and A. L. Medaglia. A review of dynamic vehicle routing problems. *European Journal of Operational Research*, 225(1), 2013.

[5] S. M. Ross et al. *Stochastic processes*, volume 2. John Wiley & Sons New York, 1996.

[6] M. M. Solomon. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations research*, 35(2), 1987.

[7] Y. Zheng, L. Capra, O. Wolfson, and H. Yang. Urban computing: concepts, methodologies, and applications. *ACM Transactions on Intelligent Systems and Technology*, 5(3):38–55, 2014.