Online Vehicle Routing: The Edge of Optimization in Large-Scale Applications

Online Vehicle Routing: The Edge of Optimization in Large-Scale Applications

Dimitris Bertsimas, Patrick Jaillet, Sébastien Martin
Operations Research Center, Massachusetts Institute of Technology

March 2018*

Abstract

With the emergence of ride-sharing companies that offer transportation on demand at a large scale and the increasing availability of corresponding demand datasets, new challenges arise to develop routing optimization algorithms that can solve massive problems in real time. In this paper, we develop an optimization framework, coupled with a novel and generalizable backbone algorithm, that allows us to dispatch in real time thousands of taxis serving more than 25,000 customers per hour. We provide evidence from historical simulations using New York City routing network and yellow cabs data to show that our algorithms improve upon the performance of existing heuristics in such real-world settings.

1 Introduction

Urban transportation is going through a rapid and significant evolution. In the recent past, the emergence of the Internet and of the smart-phone technologies has made us increasingly connected, able to plan and optimize our daily commute while large amounts of data are gathered and used to improve the efficiency of transportation systems. Today, real-time ridesharing companies like Uber or Lyft are using these technologies to revolutionize the taxi industry, laying the ground for a more connected and centrally controlled transportation structure, and building innovative systems like car-pooling.

*Accepted for publication in Operations Research

Tomorrow, self-driving and electrical vehicles will likely be the next transportation revolution. A major positive impact on the economy and the environment can be achieved when improving vehicle routing efficiency, using this newly available data and connectivity to its full extent.

A field that can make such important contributions is *vehicle routing*, i.e., the optimization of each vehicle actions to maximize the system efficiency and throughput. In the special case of *taxi routing*, we decide which taxi or ride-sharing vehicle to assign to each ride request. This setting is typically online, as there is little prior demand information available and the vehicle actions have to be decided in a dynamic way. There is more and more central control of these vehicle actions, allowing the design of strategies that surpass myopic agent behaviors. Furthermore, real-world applications are generally at a decidedly large scale: everyday, there are more than 500,000 Yellow Cab, Uber or Lyft rides in New York City — see [23].

In this paper, we present a tractable rolling-horizon optimization strategy for online taxi routing that can be adapted to a variety of applications. Our formulation is guided by the increased degree of control and prior information available in today's ride-sharing dispatching systems. We introduce a novel approach to make vehicle routing optimization formulations tractable at the largest practical scales, involving tens of thousands of customers per hour. This approach is general and can be extended to a variety of vehicle routing problems. We implement these online strategies on real taxi demand data in New York City, dispatching thousands of vehicles in real time and outperforming state-of-the-art algorithms and heuristics, thus showing the edge of optimization.

1.1 Related Work

Dynamic vehicle routing is the general problem of dispatching vehicles to serve a demand that is revealed in real time.

In the *pick-up and delivery* problem, vehicles have to transport goods between different locations. When vehicles are moving people, the routing problem is referred to as *dial-a-ride* in [5]. Taxi routing is a special case of dial-a-ride problem with time-windows, where vehicles can transport only one customer at a time, with pick-up time windows but no destination time windows. Customers are also associated with a pick-up time window, which is a typical model of customer flexibility in diverse applications of vehicle routing as in [11, 1, 9]. This constraint can be relaxed, and vehicle routing with soft time windows, for example in [15], penalizes a late pick-up instead of disallowing it. We use "hard" time windows in this work, though our approach can be extended to the soft time window case.

[27] argues that taxi routing has received relatively small attention in the field of vehicle routing, as the practical problem sizes are typically large, and last-minute requests leave "limited space for optimization". Nonetheless, with the emergence of ride-sharing smart-phone applications, it has become easier to request for a last-minute ride even when the available vehicles are far away, and to book a ride in advance. Such possibilities can be modeled in vehicle routing formulation using pick-up time windows and prior request times, and have been studied for different applications in [16, 34]. We will demonstrate that the additional prior information they provide can be leveraged when optimizing the routing decisions, even at the largest scale.

With the recent interest in real time ride-sharing, several large scale online decision systems for taxi scheduling have been proposed and implemented, though these applications focus more on managing large-scale decision systems than optimizing vehicle actions. [21] balances supply and demand in a discretized space and time, but does not consider microscopic routing decisions. [25, 26, 19, 29] implement large scale systems of taxi-pooling, in which vehicles can transport several customers at the same time. These approaches focus on how to best match different requests, but less on routing vehicles from request to request. [35] studies taxi routing with autonomous vehicles, taking into account congestion in a network-flow formulation.

Several strategies have been proposed for dynamic vehicle routing. Simple online routing algorithms can be studied in a worst-case approach using competitive ratios as in [17]. However, when some prior information is available, practical approaches are re-optimization and rolling-horizon algorithms — see [5, 31]: a "static" (or "offline") solution is constantly updated as new demand information becomes available, and this solution is used to decide the vehicles actions in real-time. This strategy has been applied with success, and [34] show that the quality of the online decision depends on the quality of the offline solutions that are used at each iteration. Together with the rolling-horizon, [2] uses a scenario-based approach with consensus, [3] successfully adds a waiting and relocation strategy, and [22] uses a double-horizon to take into account long-term goals. The offline decision problems can then be translated into well-defined optimization formulations, and sometimes solved to optimality. These formulations are typically solved using column generation as in [1], which is also used in the online setting in [9]. Custom branching algorithms, as found in [13, 11], are used for specific routing problems. [4] formulates the decision problems using constraint programming. For problems similar to taxi routing, with identical vehicles and paired pick-up and delivery, variants on network flow formulations have been proposed, for example [34] uses such a formulation to optimize truckload pickup and delivery. Unfortunately, these exact algorithms rarely scale past a few dozens or hun-

dreds of customers and vehicles, depending on the application. We will use such formulations in this paper, though applying them on problems with tens of thousands of customers and thousands of vehicles.

A classical way to solve these static vehicle routing problems at a larger scale is the use of heuristics, discussed in the survey [8]. Existing solutions can be locally improved by exploring their neighborhood: for example, 2-OPT is a famous general-purpose heuristic that was first introduced for the traveling salesman problem (TSP) in [10]. In practice, combinations of insertions-based greedy construction heuristics, local-improvement and exploration heuristics are used, as in [32, 8, 22]. For sizable problems such as taxi routing, we found that using a first-accept local-improvement heuristic similar to the 2-OPT* algorithm presented in [28] was a good trade-off when limited computational time is available, and we use it as a benchmark for our work. Unfortunately, these heuristics are usually special-purposed and have to be adapted to each particular new problem. State-of-the-art algorithm of vehicle routing include popular meta-heuristics such as Tabu Search applied in [12], Evolutionary Algorithms, Ant Colony Algorithms, Simulated Annealing and hybrid algorithms that combine the advantages of different methods, as in [4]. In this paper, we were not able to successfully apply any of these algorithms, because of the size of our problem and the very small time available for computations.

To the best of our knowledge, there is no large-scale benchmark for dynamic vehicle routing, as emphasized in [27]. To test our algorithms, we chose the New York City Taxi and Limousine Commission dataset, available at [23] and frequently used in the literature. This massive dataset contains all ride-sharing and taxi trips in NYC, starting from 2009, for a total of more than 1 billion rides. A comprehensive description of this data-set is available in [33]. It has been used for vehicle routing decisions in [26, 35, 29]. We used the OpenStreetMap map data [14] to reconstruct the real city network, together with the work in [6] to infer link travel-times from the taxi data. Furthermore, we used Julia, a programming language with a focus on numerical computing introduced in [7], in combination with the optimization modeling library described in [18], to create a large scale simulation and visualization for real-world routing, and support our experiments.

1.2 Our Contributions

This paper explores taxi routing, in the contemporary context of increased connectivity and prevalent data: we formulate, solve, scale and apply optimization formulations to real-world settings. Overall, we show that some seemingly intractable optimization formulations in vehicle routing can be

Figure 1: Our taxi simulation software, displaying online taxi routing in Manhattan with 5000 taxis and 26,000 customers. The red circles represent taxis, and the green squares represent customers being transported or waiting. The software shows the taxi movements in real or accelerated time, and implements all the online and offline algorithms we discuss in this paper. It has been designed to run on a standard laptop.

scaled to the largest problem sizes. This is desirable, as these formulations generalize much better than special-purpose heuristics to the various operational constraints of real-world applications.

Motivated by the centralization and modernization of taxi routing in the ride-sharing industry, we formulate the online taxi problem as a pick-up and delivery problem, using re-optimization and an efficient network flow mixed-integer optimization formulation similar to [34], to leverage any prior information of ride requests to make better decisions. In an extensive empirical study with synthetic data, we show that in situations of high demand, optimal solutions to the offline taxi routing problem are usually significantly superior to the output of common local-improvement and greedy routing algorithms. This confirms the edge of optimization formulations on simple heuristics for the taxi routing problem, and outlines what practical situations make these formulations easy or difficult to solve in practice.

To scale our formulations to real-world applications with thousands of taxis and tens of thousands of customers, we use the specific structure of taxi-routing applications with high demand to dramatically reduce the problem size. We additionally introduce a novel "backbone" algorithm, that first computes a restricted set of candidate actions that are likely to be optimal, allowing us to efficiently solve a much sparser problem. On a very time-constrained re-optimization schedule, with only 15 seconds to solve a vehicle routing problem involving thousands of taxis, we show that our new algorithm performs significantly better than other popular large-scale routing heuristics.

We created an open-source simulation software (available in the TaxiSimulation Julia package [20]) using state-of-the-art technologies, allowing us to simulate and visualize taxi-routing in real-world setting, and presented in Figure 1. We use New York City taxi ride data and the complete Manhattan routing network to apply our algorithms in practical settings, leading us to confirm the results we got from synthetic data. The insights we get from such applications are relevant to the current and future taxi and ride-sharing industry, and our models have the potential to be extended to a variety of other applications. For reproducibility, the input, output and all the code of our experiments are shared with the published version of this paper.

Section 2 introduces and defines the online taxi routing problem, and we study in Section 3 its offline counterpart, formulating it using mixed integer optimization and comparing it to established heuristics on synthetic data. In Section 4, we demonstrate how to scale this formulation to the applications of interest. We finally apply the offline algorithms to large online taxi problems in Section 5, using re-optimization, and with real demand data in NYC.

2 The Online Taxi Routing Problem

In this section, we introduce the online taxi routing problem and the notations we will use throughout this paper. This model captures any prior information we may have on customer requests, due to prior booking or customer pick-up flexibility.

2.1 Model and Data

We consider the *online taxi routing problem*, a special case of the *online dial-a-ride problem with time windows*. In this application, vehicles are only allowed to serve one customer at a time.

Let $\mathcal{C}$ be the set of all customers. A customer $c \in \mathcal{C}$ is associated with a pick-up time window (t_c^{min}, t_c^{max}), corresponding to its minimal and maximal possible pick-up times. In the online setting, we also introduce a confirmation time t_c^{conf}, at which customer c is provided with a guarantee to be picked up (or is rejected), and a request time $t_c^{request}$, at which the customer's information becomes available. Note that $t_c^{request} \leq t_c^{conf}$, as the confirmation of future pick-up can only happen after the pick-up request.

We represent each customer as a node in a directed graph $\mathcal{G}$. An arc (c', c) in $\mathcal{G}$ represents the possibility for a vehicle to pick-up customer c immediately after servicing customer c'. Each arc (c', c) is associated with a travel time $T_{c',c}$ such that we must have $t_{c'} + T_{c',c} \leq t_c$, where $t_{c'}$ and t_c are the respective pick-up times of customers c' and c. $T_{c',c}$ typically represents the time for a taxi to serve customer c' and to drive to the pick-up location of c. Each arc is also associated with a profit $R_{c',c}$, that represents the quantity we want to maximize. In this work, we use it to represent the profit of the taxi company, and set $R_{c',c}$ to the fare paid by customer c minus the cost of driving from the drop-off point of c' to the pick-up point of c and to its destination.

We restrict ourselves to the case where $\mathcal{G}$ is acyclic. In other words, the pickup time windows can only allow one customer to be picked-up before or after another one, but never both. There is an arc $c \to c'$ in $\mathcal{G}$ if and only if:

$$t_c^{min} + T_{c,c'} \leq t_{c'}^{max} \tag{1}$$

There exists a cycle of length 2 if and only if Equation (1) is verified from c to c' and also from c' to c. By combining the two equations we obtain:

$$(t_c^{max} - t_c^{min}) + (t_{c'}^{max} - t_{c'}^{min}) \geq T_{c,c'} + T_{c',c} \tag{2}$$

Negating Equation (2) gives us a sufficient condition for the absence of any 2-cycle:

$$(t_c^{max} - t_c^{min}) + (t_{c'}^{max} - t_{c'}^{min}) < T_{c,c'} + T_{c',c} \quad \forall c, c' \in \mathcal{C} \tag{3}$$

In other words, the condition states that the sum of the lengths of the pick-up time windows within the cycle should be less that the total cycle travel-time, and the exact same reasoning shows that this condition works with any cycle length. A stronger sufficient condition to avoid any cycle is therefore that each pick-up time window is smaller than the following ride time:

$$(t_c^{max} - t_c^{min}) < T_{c,c'} \quad \forall c, c' \in \mathcal{C} \tag{4}$$

This condition is a little too extreme, but it nonetheless holds for our taxi routing application: the time windows we use in this paper are of the order of 5 minutes, and the vast majority of taxi trips considered in this paper take more than 5 minutes (and most of them a lot longer). The few trips that are smaller than 5 minutes are still satisfying Equation (3). Taxi routing is not the only application where this assumption holds: any dynamic vehicle routing application with time windows that are no bigger than the typical trip length will have no or few cycles. For in-between applications that just have a few cycles, a simple pre-processing step can remove the cycles, or else adding sub-tour elimination constraints will be very fast and tractable. Nonetheless, for applications with larger (or infinite) time windows and thus a large number of cycles in $\mathcal{G}$, the algorithms of this paper would have to be adapted.

Let $\mathcal{K}$ be the set of all taxis, that are supposed to be identical and whose initial positions are represented as additional nodes in $\mathcal{G}$. Each taxi k is parametrized by a initial time of service t_k^{init} at which it becomes available. For each customer c that can be the first pick-up of taxi k from its original position, we add the arc (k, c) to $\mathcal{G}$. This arc is associated with a travel-time $T_{k,c}$, typically the time for taxi k to go to c's pick-up location, and a profit $R_{k,c}$, typically the fare paid by c minus the driving costs.

2.2 Decisions

A solution to the taxi routing problem is a subset of arcs of $\mathcal{G}$ that designate the sequences of customers assigned to each taxi. Each customer must only be picked-up by at most one taxi, while respecting its pick-up time window constraint: if arc (c', c) is in the solution (a taxi serves the two customers sequentially), then we must have $t_{c'} + T_{c',c} \leq t_c$.

The goal is to maximize the total profit of the solution, as described by the parameters $R_{c',c}$ and $R_{k,c}$. Additionally, the problem is solved in an online setting, where the information of the existence of customer c is only revealed at time $t_c^{request}$: the node appears in the graph, together with its arcs to and from other known nodes. In this setting, the decision to add the

arc (c', c) to the solution has to be made early enough, when the taxi that is serving customer c' can still pick-up customer c on time. Moreover, the decision whether to pick-up or reject customer c must be made before t_c^{conf}.

2.3 Interpretation

This formulation is general enough to model many optimization objectives. For example we can minimize total empty driving time instead of profit by modifying the R parameters accordingly. Setting $R_{c',c} = 1 \ \forall c \in \mathcal{C}$ will maximize the throughput: the total number of customers that we can serve. Also, setting $t_c^{request} = 0 \ \forall c \in \mathcal{C}$ corresponds to the full-information offline problem, and $t_c^{request} = t_c^{min} \ \forall c \in \mathcal{C}$ corresponds to the fully online problem without any prior information.

Note that travel-times T are deterministic once revealed. Also, the destination of customer c is revealed at time $t_c^{request}$, as it allows us to plan the next moves as we know the travel-times to the next customers. As seen in the introduction, we focus on well-connected taxi systems, that already ask customers for their destination when requesting a ride.

A typical setting to compute the travel-times T is to consider a routing network, where each customer will be associated with a pair of origin and destination nodes. Assuming stationary travel-times for each edge of the network and some additional routing rules such as "taxis use the fastest path", we can compute the times T, possibly including additional constant service time to pick-up and drop-off each new customer. As we will discuss in Section 5, these travel-time forecasts can be adjusted in a dynamic way along with the re-optimization process.

3 Offline Routing: the Edge of Optimality

In this section, we introduce the offline taxi routing problem, that naturally appears when using a re-optimization strategy for the online taxi routing. We present a mixed integer optimization (MIO) formulation of the offline problem, along with a set of heuristics. These different algorithms are compared on synthetic data, and we develop an empirical intuition about the effect of a variety of practical settings on the problem difficulty and the algorithms performances. We show that in situations of high demand, provably optimal solutions to the routing problem outperform their heuristic counterpart by a large margin.

3.1 A Re-Optimization Approach to Online Taxi Routing

When all the demand information is known beforehand, the problem is called *offline* (or static). In the offline problem, there is no uncertainty about future customers. This is equivalent to setting all the request times to the beginning of the instance, i.e., $t_c^{request} = 0$ for each customer c. In this case, the taxi routing problem introduced in Section 2 becomes the *Offline Taxi Routing Problem*. It is a well-defined optimization problem, and the feasible solutions maximizing profit are offline optimal. While most real-world taxi routing problems are not offline, the profit of an offline optimal solution is an upper bound to the profit of any strategy applied to the corresponding online problem.

If an efficient solution method for the offline problem is available, it can be used to solve the online problem through *re-optimization*. This online strategy repeatedly solves the offline problem with the known customers, and implements the first taxi actions as time goes by. Formally, given a re-optimization rate Δt^{update} (in our case we will always use $\Delta t^{update} = 30$ seconds), iteration k of the strategy solves the offline problem with all unserved customers known at time $t = k\Delta t^{update}$, i.e., the set of customers $\left\{c \in \mathcal{C},\ t_c^{request} \leq k\Delta t^{update}\right\}$. We then implement all the taxis actions that take place between $t = k\Delta t^{update}$ and $t = (k+1)\Delta t^{update}$ in the previously computed offline solution. A detailed description of our implementation of the re-optimization strategy is presented is Section 5.1.

The re-optimization online strategy has been shown to work well in practice, see [5] and [34]. However, its efficiency relies on the quality of the available offline solution methods, and their ability to give good solutions in a time that needs to be less than Δt^{update}. In the examples of this paper we set a limit of 15 seconds for the computation of an offline solution within a re-optimization iteration.

In the rest of this section, we introduce and compare different offline algorithms, focusing on their tractability and the quality of the solutions they produce.

3.2 Offline Solution Methods

We formulate the offline taxi-routing problem using MIO, so that we can use a MIO solver to compute provably optimal solutions. We also introduce a set of heuristics that provide good feasible solutions and can serve as a benchmark.

MIO formulation. We translate the taxi problem decisions, objective and constraints from Section 2 into a linear mixed-integer optimization formulation.

Each arc of graph $\mathcal{G}$ is associated with a binary decision variable (x or y), representing whether this arc is used in the solution. For each customer c, we also add the binary variables p_c to represent them being picked-up or rejected, together with the continuous decision variable t_c to represent their pick-up time.

$$y_{k,c} = \begin{cases} 1, & \text{if customer } c \text{ is picked-up by taxi } k \text{ as a first customer,} \\ 0, & \text{otherwise.} \end{cases}$$

$$x_{c',c} = \begin{cases} 1, & \text{if customer } c \text{ is picked-up immediately after customer } c' \text{ by a taxi,} \\ 0, & \text{otherwise.} \end{cases}$$

$$p_c = \begin{cases} 1, & \text{if customer } c \text{ is picked-up by a taxi,} \\ 0, & \text{if } c \text{ is rejected.} \end{cases}$$

$$t_c = \quad \text{pick-up time of customer } c.$$

We maximize the total profit, which is the sum of the profits associated with each arc of $\mathcal{G}$ in the solution.

$$\text{maximize} \sum_{k \in \mathcal{K},\, c \in \mathcal{C}} R_{k,c} y_{k,c} \; + \sum_{c',c \in \mathcal{C}} R_{c',c} x_{c',c} \tag{5}$$

To enforce that each taxi is associated with a unique sequence of customers to pick-up, we implement a set of network-flow constraints on the variables x, y and p.

$$p_c \;=\; \sum_{k \in \mathcal{K}} y_{k,c} \;+\; \sum_{c' \in \mathcal{C}} x_{c',c} \qquad \forall c \in \mathcal{C} \tag{6}$$

$$\sum_{c \in \mathcal{C}} x_{c',c} \;\leq\; p_{c'} \qquad \forall c' \in \mathcal{C} \tag{7}$$

$$\sum_{c \in \mathcal{C}} y_{k,c} \;\leq\; 1 \qquad \forall k \in \mathcal{K} \tag{8}$$

$$x_{c',c} \;\in\; \{0,1\} \qquad \forall c', c \in \mathcal{C} \tag{9}$$

$$y_{k,c} \;\in\; \{0,1\} \qquad \forall k \in \mathcal{K}, c \in \mathcal{C} \tag{10}$$

$$p_c \;\in\; \{0,1\} \qquad \forall c \in \mathcal{C} \tag{11}$$

Eq. (6) defines p_c: a customer c is picked up if and only if a (unique) taxi k serves her as a first customer (variable $y_{k,c}$) or after another customer c'

(variable $x_{c',c}$). Eq. (7) guarantees that each customer c' is either picked-up and followed by at most one other customer c ($\sum_{c\in\mathcal{C}} x_{c',c} \leq p_{c'} = 1$) or not picked up and thus not followed by any customers ($\sum_{c\in\mathcal{C}} x_{c',c} \leq p_{c'} = 0$). Eq. (8) states that each taxi k has at most one first customer. Together these constraints can be interpreted as "flow constraints" on the network $\mathcal{G}$, and guarantee that each feasible solution is a set of edges in $\mathcal{G}$ that corresponds to a set of non-intersecting paths starting from taxis nodes. Our assumption that there is no cycle in the graph plays an important role here: it allows us to avoid cycle-breaking constraints that usually appear in vehicle routing with large time windows.

We add the pick-up time window constraints:

$$t_c^{min} \leq t_c \leq t_c^{max} \qquad \forall c \in \mathcal{C} \qquad (12)$$

$$t_c - t_{c'} \geq \left(t_c^{min} - t_{c'}^{max}\right) + \left(T_{c',c} - \left(t_c^{min} - t_{c'}^{max}\right)\right) x_{c',c} \qquad \forall c, c' \in \mathcal{C} \qquad (13)$$

$$t_c \geq t_c^{min} + \left(t_k^{init} + T_{k,c} - t_c^{min}\right) y_{k,c} \qquad \forall c \in \mathcal{C},\ k \in \mathcal{K}. \qquad (14)$$

Eq. (12) bounds the pick-up times to the customer time windows. Eqs. (13) and (14) are two strengthened *Big M* sets of constraints that make sure that the sequence of customers assigned to each taxi is compatible with their respective pick-up times. For example, if customer c' is served by a taxi immediately before customer c (i.e., $x_{c',c} = 1$), Eq. (13) becomes $(t_c - t_{c'}) \geq T_{c',c}$, which is exactly the meaning of the travel-time $T_{c',c}$ as defined in Section 2. Conversely, if $x_{c',c} = 0$, Eq. (13) becomes $(t_c - t_{c'}) \geq t_c^{min} - t_{c'}^{max}$, which is always true given the time window constraint (12).

The MIO formulation (5)-(14) has $O(|\mathcal{K}|.|\mathcal{C}| + |\mathcal{C}|^2)$ constraints and decision variables. Nonetheless, not all variables $x_{c',c}$ and $y_{k,c}$ need to be defined. For example, we do not need the decision variable $x_{c',c}$ if $t_{c'}^{min} + T_{c',c} \geq t_c^{max}$, because the pick-up time constraint (13) will force $x_{c',c} = 0$. It is therefore sufficient to only consider the decision variables corresponding to the actions that are compatible with the pick-up time windows, which correspond by definition to the arcs of graph $\mathcal{G}$. Let $N = |\mathcal{K}| + |\mathcal{C}|$ be the number of vertices in graph $\mathcal{G}$ and E be the number of arcs. We obtain $O(E + N)$ constraints and decisions variables, which is why this formulation is particularly efficient, owing to the fact that decision variables $x_{c',c}$ are not indexed by the taxi k that serves customers c and c'. We can then use a MIO solver to get an optimal integer solution to the offline taxi-routing problem. We call this optimal algorithm `MIOoptimal`.

Max Flow Heuristic. In the previous MIO formulation, the constraints (6)-(11), together with the objective (5) represent a max-flow problem with

integer bounds on the flow variables. Thus, extreme points of the formulation are integral, and we can use the simplex algorithm to get an optimal integral solution. Unfortunately, time window constraints (12)-(14) break this integrality property.

However, in the special case where the pick-up times are fixed, we obtain the following integrality result:

Theorem 1. *If each customer has a fixed pick-up time, i.e., the time windows are limited to one unique pick-up time* $t_c^{min} = t_c^{max} = t_c^*, \quad \forall c \in \mathcal{C}$ *then the mixed-integer formulation* (5)-(14) *is integral.*

Proof. First, replacing $t_c^{min} = t_c^{max} = t_c^*$ into Constraint (12), we obtain $t_c = t_c^*$. By substituting the decision variable t_c with its value t_c^* in Constraint (13), we obtain

$$(t_c^* - t_{c'}^*) \;\geq\; (t_c^* - t_{c'}^*) + (T_{c',c} - (t_c^* - t_{c'}^*))\, x_{c',c},$$

which is equivalent to

$$(T_{c',c} - (t_c^* - t_{c'}^*))\, x_{c',c} \leq 0. \tag{15}$$

If $T_{c',c} - (t_c^* - t_{c'}^*) > 0$, then we must have $x_{c',c} = 0$ and the formulation is equivalent to a formulation in which variable $x_{c',c}$ is removed.

If $T_{c',c} - (t_c^* - t_{c'}^*) \leq 0$, then Equation (15) is always true no matter what the value of $x_{c',c}$ is. As a consequence, Constraint (13) is inactive on the decision variables x, y and p.

The same reasoning applies to Constraint (14). Therefore, the feasibility region of the decision variables x, y and p is the same as the one defined by the network-flow constraints (6)-(11). This formulation is integral. □

When pick-up times are fixed, the integrality result means that we can solve the offline problem efficiently, for example using the simplex algorithm. We use Theorem 1 to design a heuristic for the offline taxi-routing problems with time windows. If we assign to each customer c a fixed pick-up time t_c^* such that $t_c^{min} \leq t_c^* \leq t_c^{max}$, then the optimal solution for the max-flow problem with fixed pick-up times t_c^* is feasible for the general formulation with time windows. Note that these solutions are often sub-optimal, as they do not use the time windows flexibility to pick-up customers more efficiently. Empirically, setting $t_c^* = t_c^{max}$ will yield good solutions, as taxis have more time to go from their first position to the first customers. On the other hand, when time windows are small, the solutions are often near-optimal or even optimal on some problems. We call this heuristic `maxflow`.

Baseline Heuristic: Greedy Insertion. A simple approach to the offline taxi routing problem is to assign the customers to taxis in a greedy way. We iterate through the customers by order of t_c^{min} (earliest customers first), and we assign them to the closest available taxi or reject them if no taxi is available. This heuristic is related to insertion-based solutions construction algorithm as presented in [8]. We name this algorithm `greedy`, and its formal implementation is detailed in Appendix A.1. Because of its simplicity, tractability and wide-spread use, `greedy` will be our baseline for the offline taxi routing problem.

Local-Improvement with 2-OPT. Traditional solution methods for large scale vehicle routing include heuristics that locally improve a feasible solution in an iterative way. The 2-OPT algorithm is a popular local-improvement heuristic, first introduced for the TSP in [10]. We implement an optimized version of the 2-OPT* algorithm presented in [28], in order to compare our MIO formulation to state-of-the-art fast heuristics. We initialize it with the `greedy` solution, and stop it when it reaches a locally optimal solution. We name this algorithm `2-OPT`, and its details are presented in Appendix A.2.

We chose not to use more complex meta-heuristics with exploration, such as Tabu-Search presented in [12]. While we acknowledge these meta-heuristics avoid local optima and are common in vehicle routing, we could not find a way to implement a version of Tabu Search that worked with the limited time budget of a few seconds of online decision making and the large problem size with tens of thousands of customers.

3.3 Application on Synthetic Data

We generate random synthetic instances of offline taxi routing to evaluate the algorithms presented in Section 3.2. We compare the running time and the quality of solutions in different scenarios, in order to gain insights that we will use to solve large-scale real-world problems.

Routing in synthetic city. To compare the solutions of `MIOoptimal` to the solutions provided by the other algorithms, we have designed a way to generate synthetic routing problems. We need these synthetic problems, as real-world problems are rarely small enough to be solved to optimality by state-of-the-art commercial solvers. We have built synthetic instances that can be solved to optimality while being large and complex enough to provide interesting insights.

The synthetic routing network represents a simplified city and its suburbs.

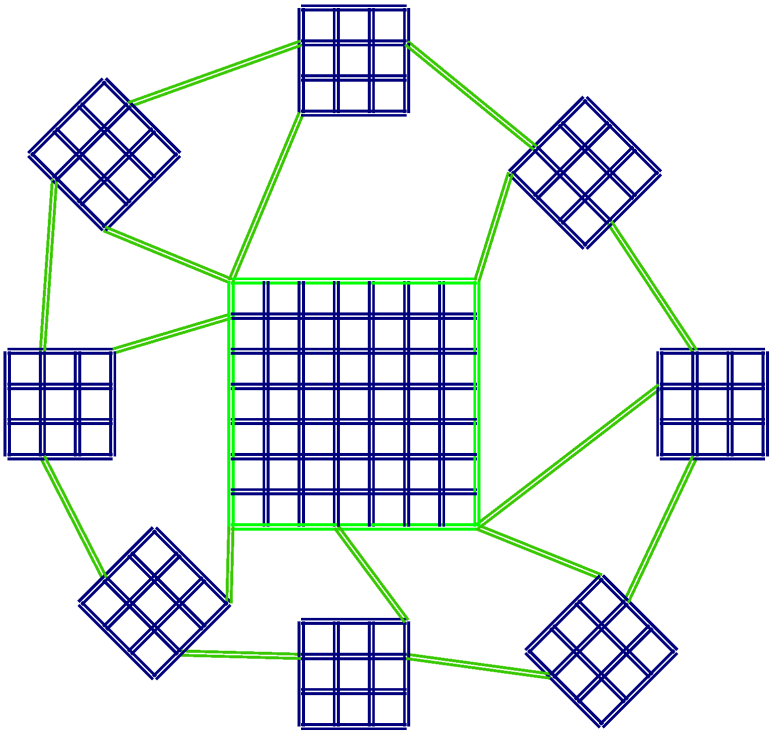

Figure 2: Routing network used to generate the synthetic instances of taxi routing. Each road is two-way, and the green connecting arcs have faster travel-times. The travel-times are chosen such that the mean trip-time between two vertices in the routing graph is around 10 minutes. The network has been designed to represent commuting effect between residential area and city-center. Customers trips are randomly generated as Poisson processes on each intersection of the routing network.

The graph has 192 nodes and 640 bi-directional arcs. The downtown area is represented by a 8 times 8 square, and the 8 suburbs are represented by 4 times 4 squares. Travel-times are slower inside the city and suburbs, and faster in connecting links and around the city. The routing network is represented in Figure 2. This network and all the algorithms are implemented using our open-source simulation and visualization framework in Julia.

On this network, we create a random one-hour instance of taxi-routing. We choose the actions of a fleet of 20 taxis with uniformly distributed initial locations. Customers are randomly generated as a Poisson process with a fixed rate, with the origin and destination of each trip uniformly drawn across the network nodes. The fares are set to be proportional to the distance of the trips, which are defined as the paths that minimize the total travel-time. We compute the time parameters of the taxi routing problems $T_{c',c}$ using the total time of these shortest paths, to serve customer c' and then go to customer c origin. The profit parameters $R_{c',c}$ are set equal to the profit, i.e., the difference between the fare paid by c and a cost proportional to the driving time.

The travel-times $T_{c',c}$ are selected so that the highway links (green in Figure 2) are twice as fast as the other links, and so that a taxi can serve up to 3-4 customers per hour. The profit $R_{c',c}$ are computed such that there is a cost of \$5 per hour of driving and \$1 per hour of waiting, and a customer fare of \$80 per hour of driving.

Results We study the influence of the time windows and the level of demand on the behavior of our algorithms. We select three levels of demand, setting the rates of the customer Poisson processes so that the expectation of the total number of customers is respectively 40, 70 and 140 customers. 40 customers (low demand) corresponds to optimal solutions in which taxis are idle half the time and are able to serve all customers. 70 customers (medium demand) corresponds to a matched supply and demand: almost all customers are accepted and taxis are driving most of the time. 140 customers (high demand) represents a surge scenario, where taxis can only accept 50% of the customers on average. We give all customers a fixed time window around their preferred pick-up time, from 1 to 6 minutes. We average our results across 20 random simulations for each set of parameters. Table 1 compares the profit of the solutions of each algorithm to the `greedy` baseline, and Figure 3 shows the computational time needed by the commercial solver Gurobi to compute the optimal solution (`MIOoptimal`).

Time Window	Demand	Algorithms Increase in Profit		
		`MIOoptimal`	`2-OPT`	`maxflow`
1 min.	low	1.94%	1.13%	1.27%
	medium	8.24%	3.70%	5.75%
	high	15.92%	8.19%	**12.82%**
3 min.	low	1.73%	1.22%	1.10%
	medium	9.00%	5.36%	2.75%
	high	14.11%	7.24%	4.98%
6 min.	low	1.42%	0.96%	-0.86%
	medium	9.38%	5.84%	-2.60%
	high	**19.93%**	**11.64%**	3.17%

Table 1: Comparing the offline solvers to the `greedy` baseline. Each row corresponds to a different setting of synthetic customer data: we vary the customers time windows (flexibility in the pick-up time), and the level of demand (number of customers). We show the improvement in profit of our algorithms compared to the `greedy` heuristic, averaged across 20 randomly generated simulations. Low demand corresponds to 40 customers, and taxis are typically idle half of the time. Medium demand is 70 customers, which represents a balanced supply and demand. High demand is 140 customers and at least half of the customers are typically rejected. We represent in bold the most favorable situation for each algorithm.

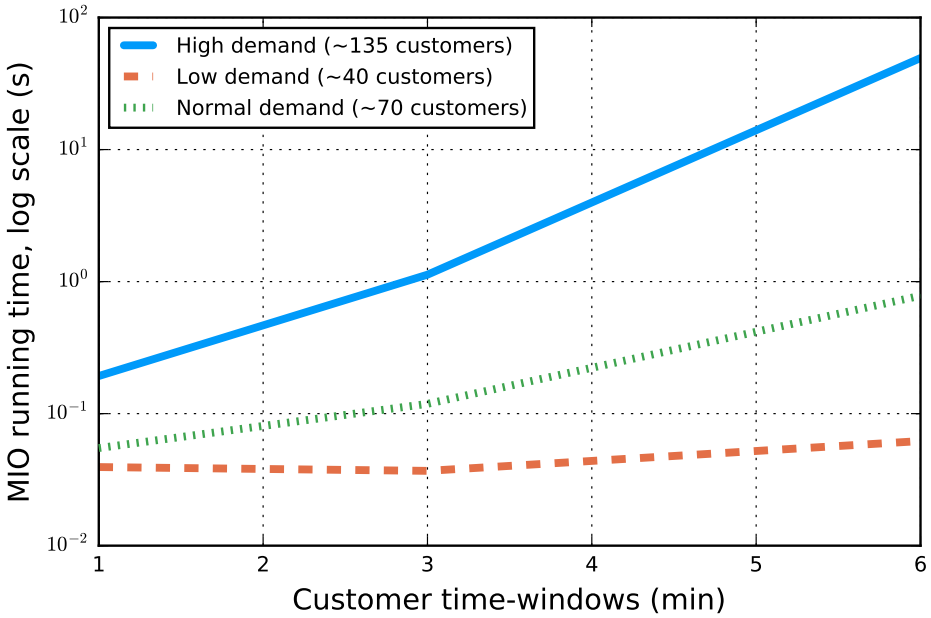

Figure 3: Time for `MIOoptimal` to find the optimal solutions. The times are averaged across 20 simulations. The three curves represent the different levels of demand, and the x-axis is the length of the customers time windows in minutes. Generally, a larger time windows and more customers lead to an exponentially higher computational time. Note the logarithmic scale of the y-axis.

3.4 The Edge of Optimality

The results of the simulations on synthetic data, presented in Table 1 and Figure 3, allow us to split the level of demand and the length of the time windows into three main categories of vehicle routing problems. These settings represent fundamentally different optimization problems, and we study how the solution methods compare in each one of them.

First, when demand is low, the `greedy` heuristic performs almost optimally, always within 2% of the optimum in our simulations. This situation is intuitive: most taxis being free, assigning the nearest free taxi to a customer performs well in practice. In this situation, optimization is not extremely useful and we recommend using `greedy`, which is the fastest and the most interpretable.

When demand is medium to high, taxis are mostly busy and `greedy` is typically far from optimality. There is an edge in using optimization: taking into account future customers and using time window flexibility allows for better solutions. We identify two main settings in this situation.

When the time windows are small, `maxflow` is very close to optimality. Indeed, we have seen in Section 3.2 that this solution method is actually optimal when pick-up times are fixed. As commercial linear optimization solvers are typically very fast and scale well, we recommend using this heuristic in practice. It performs significantly better than `greedy` and `2-OPT` while being close to the optimal solution provided by `MIOoptimal`.

The most interesting case is when demand is medium to high and time windows are not small. This situation is the most useful in practice, as a 3-6 minutes time window is a fair estimation of customer patience. Indeed, at the time we write this article, the media reports a median Uber customer waiting time of 2-10 minutes, depending on the city, though we could not find any official statistics. High demand scenarios are also typical in taxi routing, with peak-hours everyday. In this case, optimal solutions outperform the locally-optimal solutions provided by `2-OPT` and have a strong edge on `greedy` solutions. `maxflow` can perform poorly when the time windows are large. We recommend using `MIOoptimal` when possible, but the problem can be significantly harder to solve to optimality, as shown in Figure 3. The mixed-integer optimization solver takes an exponential time, in the level of demand and in the time window length, to converge to provable optimality and also to provide near-optimal feasible solutions. When `MIOoptimal` is too slow to be used in practice, the locally optimal solution provided by `2-OPT` is a reasonable alternative, and is widely used for large scale vehicle routing, as stated in [8]. Scaling the optimal formulation to real-world applications and outperforming the local-optimization methods is the objective of the next

section.

4 Scaling Optimization to Real-World Applications

Using optimal solvers for the offline taxi routing problem leads to a significant improvement in the solution quality, particularly when customer demand matches or exceeds the vehicle supply. `MIOoptimal` nevertheless becomes quickly intractable when the number of customers and taxis increases: to obtain a proof of optimality in less than one hour with a typical laptop, the limit is around 150 customers for a 5 minutes time window.

In this section, we show how to leverage the structure of real-world vehicle routing applications to make mixed-integer optimization formulations tractable. We construct a large scale and real-world taxi routing problem in New York City using Yellow-Cab demand data. This problem involves thousands of taxis and customers, and each iteration of the re-optimization process corresponds to a mixed-integer optimization formulation with more than 10 million binary decision variables, and needs to be solved in seconds. We propose an algorithm that is tractable at this scale of taxi routing, outperforms the state-of-the-art and combines the advantages of local-search and global optimization to get near-optimal results within the allowed computational time.

4.1 Sparsifying the Flow Graph

One way to increase the tractability of `MIOoptimal` is to decrease the number of binary decision variables in the mixed integer optimization formulation presented in Section 3.2. Equivalently, removing some well-chosen arcs from the flow graph $\mathcal{G}$ will reduce the solution space and make optimization easier. Nonetheless, we risk removing some arcs that are in the optimal solution and hence decrease the quality of the result. If we find arcs that are less likely to be optimal than others, removing them can increase tractability without decreasing the optimal solution quality too much, and, given the limited computational time, lead to better practical solutions.

Our results on synthetic data in Section 3.4 show that high demand scenarios are the hardest offline taxi routing problems and are the most favorable and interesting for optimization-based algorithms. As a result, we focus on scenarios in which demand matches or exceeds supply. In the optimal solution for such a problem, a taxi is unlikely to wait or drive empty for too long before getting a customer. Indeed, if we have a large number of taxis

spread throughout the city and a high demand, taxis will probably pick-up customers that are nearby in space and time. When closer customers are available, we do not expect a taxi to drive empty and wait a long time to pick-up a far-away customer: we can safely remove the corresponding arcs from $\mathcal{G}$.

Formally, we define a cost function between nodes in graph $\mathcal{G}$. For nodes representing customers c' and c, we define the cost $C(c', c)$ to be the shortest possible "lost time" that a taxi will have to spend waiting or driving empty when serving customer c' and c sequentially:

$$C(c', c) = \max\left(T_{c',c},\, t_c^{min} - t_{c'}^{max}\right) - T_{c'},$$

where $T_{c'}$ is the time to transport customer c' from its origin to its destination. For nodes representing a taxi k and a customer c, we define $C(k, c)$ to be the minimal time (including wait) it takes for the taxi to reach and pick-up c as a first customer:

$$C(k, c) = \max\left(T_{k,c},\, t_c^{min} - t_k^{init}\right).$$

We want to keep the arcs in $\mathcal{G}$ that have the lowest cost, as they represent actions of picking up "nearby" customers, and thus more likely to be optimal. These costs are just used as an indicator of the quality of each edge, and will be used to remove the edges that are really unlikely to be in an optimal solution.

For a given sparsity parameter K, we prune $\mathcal{G}$ to create a sparser graph $K\mathcal{G}$ by only keeping the K-lowest cost incoming and outgoing arcs for every node in $\mathcal{G}$. We name this flow graph pruning technique `K-neighborhood`, and its steps are detailed in Algorithm 1.

Algorithm 1: `K-neighborhood`

Input: The flow-graph $\mathcal{G}$ and a sparsity parameter K.
Output: A sparser flow-graph $K\mathcal{G}$ by pruning $\mathcal{G}$.
begin
  Initialize graph $K\mathcal{G}$ with the same nodes as $\mathcal{G}$ and without any arcs;
  for *all node* n *in* $\mathcal{G}$ **do**
    select the K incoming arcs to node n in $\mathcal{G}$ with lowest cost $C(\cdot, n)$ and add them to $K\mathcal{G}$;
    select the K outgoing arcs out of node n in $\mathcal{G}$ with lowest cost $C(n, \cdot)$ and add them to $K\mathcal{G}$;
  end
end

The new formulation associated with the graph $K\mathcal{G}$ (for fixed K) has $O(|\mathcal{C}| + |\mathcal{K}|)$ decision variables and constraints instead of $O(|\mathcal{C}|^2 + |\mathcal{C}||\mathcal{K}|)$ for $\mathcal{G}$, which allows us to solve problems at a much larger scale. In practice,for the real-world problems we have tried, we noticed that $K = 20$ usually provides near optimal solutions, and $K = 50$ optimal solutions. We have also empirically found that the choice of K should only depend on the balance between supply and demand. When demand is lower than supply, we have seen in Section 3.4 that taxis tend to be idle and the closest taxi is likely to pick-up a customer in an optimal solution. Low values of K are then enough to get near-optimal solutions, as the closest taxis will have generally the lowest values of $C(\cdot, \cdot)$. As demand grows and matches or exceeds supply, we have empirically found that the best solutions correspond to higher values of K, up to $K = 50$. Furthermore, the size of the city and the total number of taxis and customers typically do not influence the choice of K: the choices for one given taxi are typically local, in the taxi's neighborhood, and are not influenced by the total size of the city.

When time windows are small, results from Section 3.4 indicate that `maxflow` provides near-optimal solutions. Additionally, using `K-neighborhood` with $K = 50$, problems with thousands of taxis and customers are solved in seconds using `maxflow`. When time windows are larger, typically 3-6 minutes, we have shown that the offline taxi routing problem becomes much harder, and that we need to use `MIOoptimal` to get good solutions. Unfortunately, when using `MIOoptimal` and `K-neighborhood` for our large scale applications, the problem is typically intractable for $K \geq 4$ and low values of K reduce the quality of the solutions. These observations motivate the ideas in the next section.

4.2 The Backbone Algorithm

In order to make `MIOoptimal` tractable for large instances of the taxi routing problem, we need to remove a lot of arcs from $\mathcal{G}$. We cannot set a value of K that is too low, as `K-neighborhood` would remove too many arcs that participate in optimal solutions and correspondingly decrease the quality of the solution. Nonetheless, even within a limited neighborhood around a taxi's position, there are customers that are better than others, given the positions of other taxis. If we identify these potentially good arcs of $\mathcal{G}$, we could reduce the number of arcs even more and make `MIOoptimal` tractable.

In Theorem 1 we have shown that for fixed pick-up times the `maxflow` algorithm solves the problem optimally. Furthermore, the `maxflow` algorithm scales for very large problems. If we randomly select a pick-up time within each time window and solve the fixed-pickup time problem with `maxflow`

in the graph $K\mathcal{G}$, we get a solution that is feasible for the problem with time windows, as seen in Section 3.2. If we resolve several times the fixed pick-up time problem with the tractable `maxflow` and random pick-up time within the time windows, and collect all the optimal arcs across the different solutions, we obtain a set of arcs that are likely to be optimal. This set of arcs represents a very sparse sub-graph of $\mathcal{G}$, a "backbone" for our optimization problem, on which we can use `MIOoptimal` to compute an optimal solution within the backbone network, which is near-optimal for the original graph $\mathcal{G}$. This is the `backbone` algorithm, described formally in Algorithm 2.

Algorithm 2: `backbone`

Input: The flow-graph $\mathcal{G}$; a sparsity parameter K such that `maxflow` is tractable on $K\mathcal{G}$; a limit E_{max} on the number of arcs such that `MIOoptimal` stays tractable.
Output: A backbone flow-graph $B\mathcal{G}$ that is a sparser version of $K\mathcal{G}$ with a maximum of E_{max} arcs.
begin
  Step 1: Initialize the backbone graph $B\mathcal{G}$ by removing all the arcs in $\mathcal{G}$;
  while *$B\mathcal{G}$ has less than E_{max} arcs* **do**
    Step 2: **for** *each customer $c \in \mathcal{C}$* **do**
      generate a uniformly random pick-up time $t_c \in [t_c^{min}, t_c^{max}]$;
    end
    Step 3: use `maxflow` on $K\mathcal{G}$ with the fixed pick-up times t_c;
    Step 4: add all the optimal arcs of the computed solution to $B\mathcal{G}$;
  end
end

We typically choose $K = 20$ for the large scale instances of this paper: this choice of K creates a sparse graph while rarely sacrificing optimality. This algorithm gives good results in practice, especially if the time windows are small (less that 2 minutes in our applications). Steps 2-3 of Algorithm 2 can be executed in parallel, which allows us to assign most of the available computational time to solve the mixed-integer optimization problem on the backbone $B\mathcal{G}$. For wider time windows, the `maxflow` solutions with random pickup times create too many arcs and therefore `MIOoptimal` is not tractable at the largest scale. This motivates the need to improve the `backbone` algorithm, which we do next.

4.3 The Local Backbone Algorithm

When using the re-optimization strategy presented in Section 3.1, we solve the offline taxi routing problem with all the available future demand information at every time-step of length Δt, typically 30 seconds. The offline problem at time t is very similar to the next problem at time $t + \Delta t$ as we only add and remove a few requests. Therefore, a good solution to the offline problem at time t can be used to construct a good solution to the problem at time $t + \Delta t$. More specifically, we adapt the previous solution by removing the customers that have just been served at time t and adding the new requests of time $t + \Delta t$ as "rejected" to make the solution feasible for the new problem (we do not know yet if we can accept them). We can then use this solution as a warm-start for the new problem.

In Steps 2-3 of the `backbone` algorithm in Section 4.2, the fixed pick-up time is selected uniformly randomly within the customers time windows. The idea of the local backbone algorithm is to update the customers time windows so that the solution s at time t is feasible with these pick-up times.

For each customer c served in s, we define $[t^{min}_{c,s}, t^{max}_{c,s}]$ to be the interval of possible pick-up times t_c such that s is still feasible. In other words, all taxis can still serve the same sequence of customers as prescribed by solution s, while respecting the pick-up time t_c for customer c. We have $[t^{min}_{c,s}, t^{max}_{c,s}] \subset [t^{min}_c, t^{max}_c]$. We compute $t^{min}_{c,s}$ and $t^{max}_{c,s}$ next. Suppose that in solution s, a taxi has to pick-up customers c^- , c and c^+, in this order. Then

$$t^{min}_{c,s} = \max\left(t^{min}_c, t^{min}_{c^-,s} + T_{c^-,c}\right), \tag{16}$$

$$t^{max}_{c,s} = \min\left(t^{max}_c, t^{max}_{c^+,s} - T_{c,c^+}\right). \tag{17}$$

Equation (16) states that the minimal pick-up time $t^{min}_{c,s}$ for customer c either corresponds to t^{min}_c, the beginning of its time window, or to the earliest possible time to pick-up customer c^- plus the travel-time between c^- and c : $t^{min}_{c^-,s} + T_{c^-,c}$. Equivalently Equation (17) defines $t^{max}_{c,s}$ to either be equal to t^{max}_c or to the latest possible time to pick-up c^+ minus the travel-time between c and c^+, whichever is the earliest.

Additionally, if c^{first} and c^{last} are the first and last customers to be picked-up by taxi k, there are no propagating constraints on their earliest and latest pick-up times, respectively, which leads to:

$$t^{min}_{c^{first},s} = \max\left(t^{min}_c, t^{init}_k + T_{k,c}\right), \tag{18}$$

$$t^{max}_{c^{last},s} = t^{max}_c. \tag{19}$$

Using (16) and (18), $t^{min}_{c,s}$ can be computed for each customer c by forward induction on each taxi's sequence of customers. Similarly, $t^{max}_{c,s}$ can be computed by backward induction using Equations (17) and (19). These forward

and backward computations are similar to the *Lazy* and *Eager Scheduling Algorithms* introduced in [4] to build solutions for the dial-a-ride problem, and are linear in the number of pick-ups in the route.

The `local-backbone` algorithm. We use these new time windows as a guide for our exploration process: instead of selecting random pick-up times within $[t_c^{min}, t_c^{max}]$ in the `backbone` algorithm, we select them within $[t_{c,s}^{min}, t_{c,s}^{max}]$. All the arcs generated by `maxflow` will therefore be in a "neighborhood" of solution s, allowing us to improve on the solution while building on the quality of s to limit the search space. This process can be bootstrapped to improve on itself iteratively: we name `local-backbone` this variant of `backbone`, as described in Algorithm 3.

Algorithm 3: `local-backbone`

Input: The flow-graph $\mathcal{G}$; a sparsity parameter K such that `maxflow` is tractable on $K\mathcal{G}$; a limit E_{max} on the number of arcs such that `MIOoptimal` stays tractable; a starting solution s, that can be empty if none is known.
Output: A solution s' for the offline taxi routing problem that improves upon solution s. **begin**
  while *time is available* **do**
    compute the values $[t_{c,s}^{min}, t_{c,s}^{max}]$ for each customer c using the solution s;
    create an empty "backbone" graph $B\mathcal{G}$ by removing the arcs of $\mathcal{G}$;
    add all the arcs of s to $B\mathcal{G}$;
    while $B\mathcal{G}$ *has less than* E_{max} *arcs* **do**
      for *each customer* $c \in \mathcal{C}$ **do**
        generate a uniformly random pick-up time in $[t_{c,s}^{min}, t_{c,s}^{max}]$;
      end
      use `maxflow` on $K\mathcal{G}$ with the fixed pick-up times t_c;
      add all the optimal arcs of the computed solution to $B\mathcal{G}$;
    end
    use `MIOoptimal` to solve the offline taxi routing problem on $B\mathcal{G}$, with s as a warm-start;
    update s to be this new solution s' ;
  end
end

`local-backbone` is an algorithm that combines the advantages of a local-

improvement and global optimization. It aims to avoid local-minima by using an MIO solver, and usually provides near-optimal solutions. Its main strength is when the problem is hard to solve or when we have tight constraints on computational time: the difficulty of the problem can be limited by using a very sparse graph $B\mathcal{G}$, and compensate for the corresponding decrease in the solution quality by doing more iterations of `local-backbone` to keep improving the solution as with any local-improvement method. We empirically found that the starting solution does not significantly influence the quality of the convergence: we could not find unsatisfactory local minima in our applications. Furthermore, we adapted the algorithm to get out of any local optimum, as described in the Remark below.

Remark 1. *If we modify slightly* `local-backbone` *to also add some uniformly random pick-up time (not local) in addition to the local ones, we obtain an algorithm that converges to the optimum. Indeed, at each iteration of Algorithm 3, the non-local pick-up times have a positive probability of being compatible with the optimal solution. If they are,* `maxflow` *will add the arcs of the optimal solution to $B\mathcal{G}$, and the optimal solver will find the optimal solution.*

For large scale online routing problems, we found that `local-backbone` leads to stronger solutions than `backbone`, as it is able to make the most out of a warm-start when using re-optimization. Furthermore, we have found in our experiments that `local-backbone` outperforms all the other methods we tried by a large margin. We next present computational results and compare `local-backbone` to other offline solvers.

4.4 Taxi Routing in NYC

When studying large-scale vehicle routing problems, synthetic data is not enough to represent complex real-world demand and networks. Therefore, we reconstructed the exact Manhattan routing network in New York City, and used real demand data from NYC Yellow Cabs to build accurate real-world online taxi-routing problems.

Using OpenStreetMap data, presented in [14], we extracted the complete routing network of the island of Manhattan, as represented in Figure 4. This large network, with 4324 intersections and 9518 directed arcs, was chosen because taxis and on-demand ride-sharing vehicles are an extremely popular mean of transportation in this city, with more than 500,000 trips everyday. Interestingly, a large fraction of the rides stay within Manhattan from origin to destination: taxi demand data in [23] shows that around 80% of the rides that have a pick-up location in Manhattan stay within Manhattan.

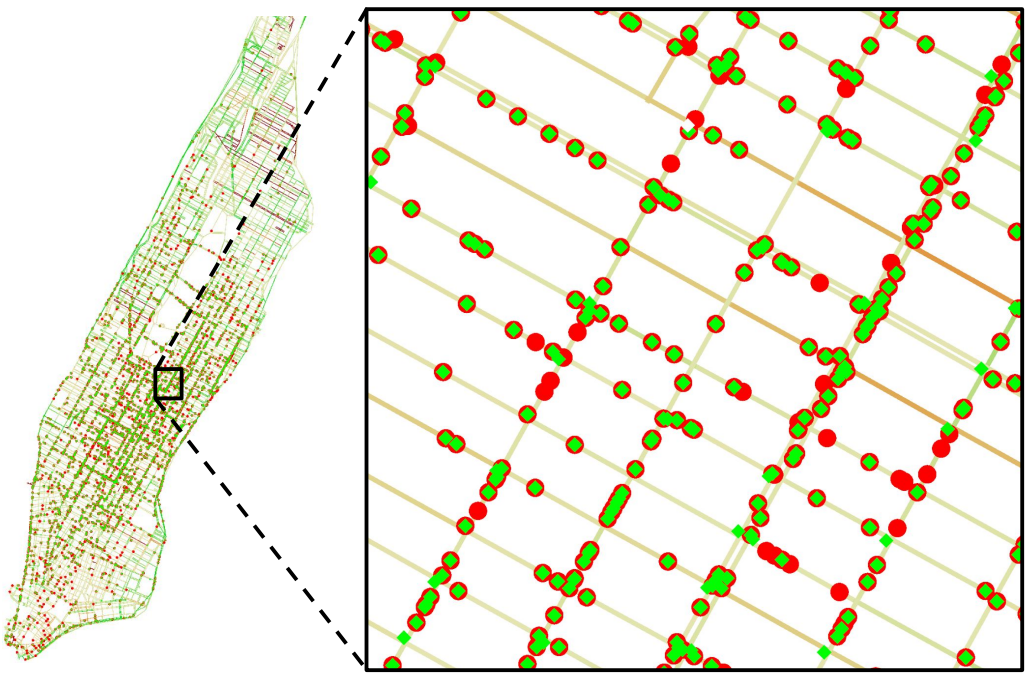

Figure 4: A taxi-routing simulation on the Manhattan routing network in NYC. On the left, we show a general view of the routing network, with 4324 intersections and 9518 directed arcs, built using OpenStreetMap data. On the right, we zoom on a detail of the online taxi routing simulation. Each red circle is a taxi, and the green squares represent customers, either waiting or being transported by a taxi. There were more than 26,000 customers and 4,000 taxis for 1.5 hours of simulation.

The New York City Taxi and Limousine Commission has released a large taxi-trip dataset that is freely available online, see [23]. We have access to more than a billion trips, all the Yellow and Green Cabs trips for the years 2009-2016. The available information includes their pick-up and drop-off points and times, the fare and tip paid, the number of passengers and more. The volume of the demand is large, with generally more than 400,000 trips a day for Yellow Cabs alone, more than 12 million per month. We focus in this paper on the yellow-cab rides of Friday 04/15/2016 12-1:30pm from Manhattan to Manhattan, which represent exactly 26,109 customers after removing data errors. This date was chosen purely arbitrarily, though we selected a time of high demand. We adapt the trips to our routing network by projecting the origin and destination of the customers on the nearest intersection. The fare paid by each customer c is used to create the profit parameters $R_{c',c}$, and we set the beginning of the pick-up time window t_c^{min} to be the real pick-up time of the customers. The values of t_c^{max}, $t_c^{request}$, and t_c^{conf} are chosen for each simulation to represent the situations we want to model.

Simulated taxis are added to the network, and we change the number of taxis while keeping the demand constant to control the balance between supply and demand. We typically need a lot less taxis than the real number of Yellow Cabs to serve the same demand, because of our optimized solutions, more centralized control and future planning. Furthermore, we used the very same yellow-cab taxi data to estimate the travel-time on all arcs of the routing network, running the algorithm described in [6] on the same demand data we used to create the rides. Therefore, these travel-times match the congestion and traffic patterns of the same precise day and time: Friday 04/15/2016 12-1:30pm. Under the assumption that taxis use the fastest route, which is verified in practice for ride-sharing companies as drivers paths are suggested and monitored in real-time by the driver's phone application, we can compute the travel-time $T_{c',c}$ for each arc of the graph $\mathcal{G}$. We also subtract a cost of \$5 per hour of driving so that $R_{c',c}$ represents the profit.

We created a micro-simulation software able to simulate, optimize and visualize online vehicle routing on real-world networks. This software has provided us a much finer control and better speed than existing software like MATsim. It also enables us to easily interact with any free or commercial solver like CBC or Gurobi, through the Julia for Mathematical Programming (JuMP) interface. Figure 4 shows an example of such a simulation in Manhattan.

Algorithm	Increase in profit, compared to `greedy`			
	$K = 2$	$K = 4$	$K = 8$	$K = 20$
`MIOoptimal`	-5.05%	**5.41%**	3.75%	2.28%
`local-backbone`	-5.74%	4.74%	8.27%	**9.13%**
`2-OPT`	**4.03%**			

Table 2: Comparing offline algorithms on a big routing problem. These results are averaged on five distinct simulations for an offline taxi routing problem in Manhattan with 2700 taxis and more than 6000 customers. The computational time is limited to 5 minutes for each algorithm. To make the problem tractable for the optimization-based methods, we apply `K-neighborhood` to shrink the flow graph $\mathcal{G}$. We show the results for different values of K, highlighting the most favorable case for each method. Our version of `2-OPT` does not use $\mathcal{G}$ and is therefore not sensitive to K.

4.5 Offline Results for Large Scale Taxi Routing

We have introduced new algorithms to scale the offline MIO formulation of taxi routing to real-world demand scenarios. The flow-graph shrinking heuristic `K-neighborhood` implements a trade-off between the tractability of the solution methods and the quality of the solutions. And the `local-backbone` algorithm is our most scalable optimization-based algorithm. In the high-demand scenario with time windows, we want to compare our algorithms to the baseline `greedy` and `2-OPT`. These algorithms are meant to be used in a re-optimization setting when solving the online problem. We study in this section an offline taxi routing problem in NYC that represents a typical iteration of re-optimization for the online problem.

We create an offline scenario, using the online taxi-routing problem presented in Section 4.4. We assign each customer a 5 minutes time window and a random request time that is on average 15 minutes prior to their first desired pick-up time, generated randomly uniformly between 0 and 30 minutes. This 15 minutes prior time was chosen to represent a situation with some reasonable prior information available, and we will study in Section 5.3 the influence of this prior request time on the quality of the solutions. We add 2700 taxis on the routing network, at random locations following the distribution of customer pick-up location in Manhattan at this time of the day. This number is chosen to represent situations with slightly exceeding demand, for which optimization algorithms are useful. In this context, the greedy heuristic is able to serve 80% of the demand on time.

We consider the offline taxi routing problem corresponding to one step of the re-optimization process: at 12:30pm, with all the customers c such that

$t_c^{request} \leq$ 12:30pm. This gives roughly 6000-6500 customers, depending on the random values chosen for $t_c^{request}$. We generate five such random problems by re-generating the request times and the taxi initial positions and average the profits generated by our different algorithms, with a computational limit of 5 minutes for each. The actual time available to solve this problem in practice is 15 seconds, but we give the algorithms more time to compensate for the fact that we do not provide a warm-start and to be able to compare the optimization power of each algorithm. In the next section, we will show how to limit the computational time. The numerical results are presented in Table 2.

The optimization based algorithms `MIOoptimal` and `local-backbone` perform better than the nearest-taxi baseline `greedy` and the state-of-the-art local improvement algorithm `2-OPT`: our algorithms scale to real-world taxi routing. `MIOoptimal` managed to find the provable optimal solution within 5 minutes only for $K = 2$. Note that this solution is worse than `greedy`, as $K = 2$ is too small and `greedy` does not operate on the pruned flow graph, and thus gives a better solution. It did not give an optimal solution in the other cases, but generally yielded a good feasible solution. For $K = 2$ and $K = 4$, `MIOoptimal` performs slightly better than `local-backbone`, because of the loss due to the backbone structure. But for larger values of K, `local-backbone` continues to improve and provides higher quality solutions, whereas `MIOoptimal` becomes intractable and fails to find better solutions in the allowed time.

`local-backbone` manages to do better than all the algorithms we tried on offline taxi routing. More than the extra 5% of profit this algorithm generates, we have demonstrated that mixed integer formulations can be used in practice for large scale vehicle routing by leveraging the "locality" of the decisions. On the other hand, we have only solved one particular iteration of the re-optimization strategy for one particular offline routing problem. We show in the next section how our algorithm performs in the full online setting, and what situations are more favorable for optimization.

5 Online Taxi Routing in NYC

In Section 3.1, we have introduced a re-optimization strategy to solve online taxi routing. This iterative algorithm requires to be able to solve large scale offline taxi routing problems within a limit of 15 seconds, which is the limit we have chosen for our applications. We have demonstrated in the previous section that `local-backbone` can be used to get near-optimal solutions to these large offline problems in a tractable way, though not yet respecting

this strong time limit. We now show how to respect the 15 seconds limit in practice, and compare our optimization-based algorithms to other online strategies. These algorithms are tested on the New York City taxi routing problem defined in Section 4.4 to gain insights on how the increasing connectivity, central control and knowledge of the future demand can be used to better optimize online routing decisions.

5.1 Re-optimization and Warm-Starts

Re-optimization involves re-solving the offline taxi routing problem with all the known future customers periodically, at every time-step of length $\Delta t^{update} = 30$ seconds. This frequent re-optimization can be leveraged to reduce the computational time needed at each iteration. We present here our approach to re-optimization in a large-scale real-time setting.

Accelerating Re-Optimization. In the re-optimization strategy, the solution of the offline problem at one iteration can be used to provide the solver with an initial solution feasible for the next iteration. We have discussed in Section 4.3 that `local-backbone` and `2-OPT` can improve on any provided initial solutions; our relatively high re-optimization frequency provides good warm-start at each step, which leads to better results when a limited time is available.

Moreover, the previous solution is not the only thing we can build on from the past iterations. Our `local-backbone` algorithm uses the flow graph $\mathcal{KG}$ to represent the problem to solve. Unfortunately, it takes time to construct the graph $\mathcal{G}$ at each iteration, to prune it with `K-neighborhood` as presented in Section 4.1, and to convert the resulting problem into a sparse matrix to give to a commercial solver. It actually takes us 10 to 40 seconds to go through these preliminary steps for a problem of the scale of taxi routing in Manhattan. Thankfully, the graph $K\mathcal{G}$ is not too different from one iteration to the next. As new customers appear, we perform an online update on $K\mathcal{G}$, adding new arcs and removing the obsolete ones. This online update is particularly useful because we never have to construct and store the full graph $\mathcal{G}$. To make such an update possible, we keep track of the cost $C(c', c)$ (as introduced in Section 4.1) of each arc of the graph $K\mathcal{G}$, and we use a heap data structure that allows us to efficiently keep and update the K-best arcs when new requests come or old requests become obsolete. Thus, we update the pruned flow graph $K\mathcal{G}$ in-place at each iteration, without reconstructing and pruning the full graph $\mathcal{G}$. This in-place update of the graph and of the corresponding sparse matrix that we send to the solver, is what we call a *formulation warm-start*. In practice, formulation warm-start

allows us to create $K\mathcal{G}$ in one to two seconds when the formulation of the previous iteration is available, instead of half a minute at each iteration.

Parallelization is also useful in practice, particularly to accelerate `local-backbone` and when using a solver to perform a branch-and-bound on the MIO formulation. Indeed, the exploration phase of the backbone algorithm can be computed in parallel, as discussed in Section 4.2.

The Online Re-Optimization Strategy The online re-optimization strategy periodically re-optimizes its assignments of future customers to taxis, sends the taxi routing decisions to the vehicles, receives the vehicles status, and processes the customer requests. We list all the steps of one iteration of our implementation of re-optimization:

1. Gather the new taxi actions since the last update, and all the new customer requests.

2. Compute the new pruned flow graph $K\mathcal{G}$: we update the one from the previous iteration to the new situation. More specifically, we add the new requests and we remove the completed picked-ups and the rejected customers. This is done while maintaining the K-sparsity property of $K\mathcal{G}$. This step corresponds to the "formulation warm-start" discussed above.

3. Update the offline solution of the previous iteration to make it feasible for the new formulation. Specifically, mark the new customers as being rejected and remove the decisions that have already been implemented.

4. Solve the optimization problem with `local-backbone`, using the formulation and the warm-start constructed in Steps 2 and 3. A solution must be provided in less than $\Delta t^{update} = 30$ seconds, but we use 15 seconds to keep a security margin and leave time to broadcast the actions to the fleet.

5. If we have reached the confirmation time t_c^{conf} of a customer c, look at the customer status in the current solution. If the customer is rejected, communicate this information to the customer, or offer her to wait for another confirmation time. In the examples of this paper, we will reject the customer. If the customer is accepted, make sure that she will be accepted in all future iterations. A simple way to do so is to add the constraint $p_c = 1$ to the MIO formulation presented in Section 3.2. This does not break the network-flow structure of the problem and makes sure that customer c is picked-up in all feasible solutions.

6. Send the taxis all the routing actions that occur before the next update. Specifically, we dispatch a taxi to a customer pick-up location so that it reaches the customer at the earliest pick-up time compatible with the new solution, which is $t_{c,s}^{min}$ as defined in Section 4.3. Taxis are not aware of the full offline schedule, as it can change in the next re-optimization iterations.

7. Idle taxis are instructed to wait at their current position. Note that other behaviors could be chosen instead, for example using forecast demand to route the idle taxis, or just let them move as they want. We do not study these choices in this paper.

5.2 Online Solution Methods

To evaluate the performance of our re-optimization strategy with `local-backbone`, we created a set of reference large-scale online algorithms that will serve as a baseline to evaluate our work.

Pure online algorithm. Our simplest algorithm, `pure-online`, does not use the customer prior request information, pretending that $t_c^{request} = t_c^{min}\ \forall c \in \mathcal{C}$. At time t_c^{min}, this algorithm will send the nearest available taxi to the customer. The taxi needs to be able to pick-up c before t_c^{max}, but does not have to be idle at t_c^{min}. This myopic algorithm is not too different from real taxi behavior, that will look for a customer in their neighborhood, or from ride-sharing for-hire vehicles, that will be matched with nearby requests. Therefore, `pure-online` will be used as a baseline to outline the extra efficiency other algorithms gain from more optimization and prior knowledge of the demand.

Planning with no re-optimization. `no-reopt` is a greedy algorithm that uses prior request knowledge to plan ahead and find better solutions, but does not re-optimize. We maintain a list of future assignments for each taxi. When a new customer requests a ride for a specific pick-up time window, we check if we can insert it in the lists of customers assigned to each taxi. If it is not possible, we reject the customer. If we can, we assign it to a taxi chosen such that this new assignment maximizes the total profit, using the efficient insertion algorithm described in Appendix A.1. Therefore, `no-reopt` takes into account the future positions of the taxis when making these decisions, though the decision cannot be changed once a customer is assigned to a taxi. This is different from the re-optimization process described in Section 5.1,

that can re-assign a customer to another taxi as new information about the future is revealed, and only decides the final action when it is time for pick-up.

Optimization-based updates. The `backbone` online algorithm is the re-optimization process described in Section 5.1. This algorithm uses `local-backbone` to perform the updates and is limited to 15 seconds of computation per iteration.

Heuristic-based updates. The `2-OPT` online algorithm is the adaptation of its offline counterpart to the online setting. We use a re-optimization process similar to the one presented in Section 5.1, removing the flow-graph computations and replacing the offline solution method `local-backbone` by `2-OPT`. We use the warm-start solution from the previous iteration, and we limit the algorithm to 15 seconds of computation. This algorithm uses all available prior information, allows for re-optimization and performs typically well in practice.

5.3 Experiments and Results

We apply our online algorithms to the taxi routing problem presented in Section 4.4. The confirmation time t_c^{conf} for each customer c is chosen to be a maximum of 3 minutes after the request time $t_c^{request}$. To study the impact of prior customer knowledge, we vary the customer request time. Let $T^{request}$ be the desired average time of prior request. We assign each customer c with a random request time $t_c^{request}$ drawn uniformly within the interval $[t_c^{min} - 2T^{request}, t_c^{min}]$. The randomness of the request times is important: for example, if each customer c were to request a ride at the non-random time $t_c^{request} = t_c^{min} - T^{request}$, the request times would be ordered by pick-up times, which is not real-world behavior. The customer time window length is the same for each customer: we assign each customer with a time window of length T^{wait}, with $t_c^{max} = t_c^{min} + T^{wait}$. To control the supply-demand balance, we vary the number of taxis while keeping the customers constant.

As discussed in Section 4.4, our algorithms are implemented in the Julia language, with a special care for computational speed and visualizations. Their parameters were all optimized to get the best results. We created a framework allowing us to test the different online strategies in the same environment, making sure that we only share the requests information in real time. All simulations are run on identical machines, using 2 CPUs and 8GB of memory. Each simulation presented in this section was done over a time period of 1.5 hours, as we simulated vehicle routing in Manhattan for the

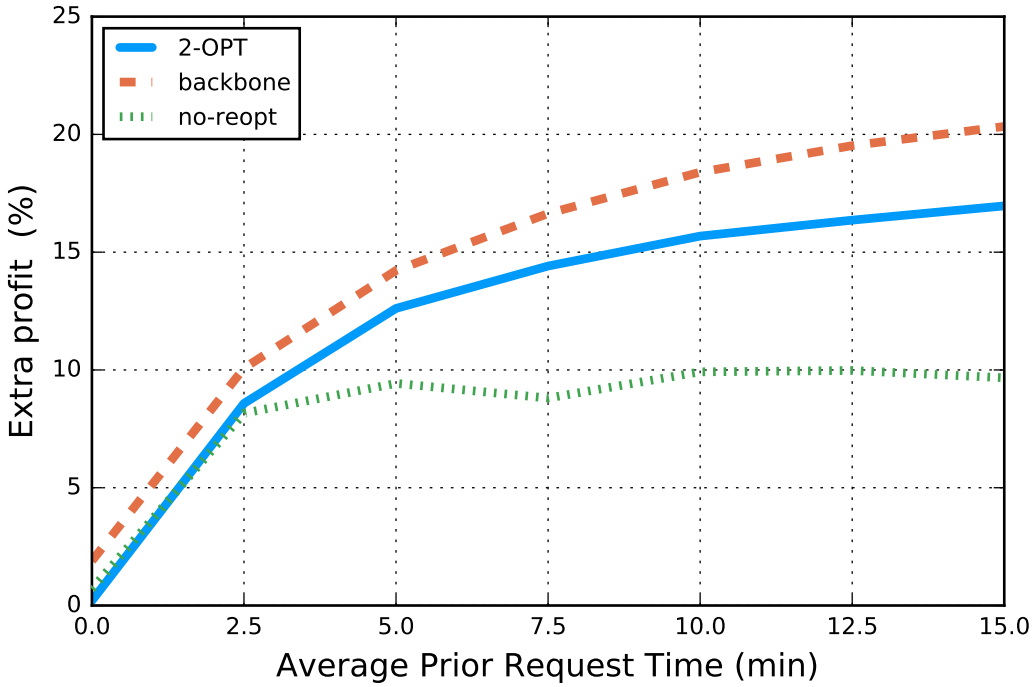

Figure 5: Varying the mean prior request time. Increase in profit of each online taxi routing algorithm compared to `pure-online`. We vary the mean prior request time $T^{request}$ from 0 (pure online situation) to 15 minutes. Each customer is assigned to a $T^{wait} = 5$ minutes time window. We control 4000 taxis, which corresponds to a high demand scenario as 80% of the demand is served in the best case.

real yellow cab demand of Friday 04/15/2016 12:00-1:30pm. Figure 4 is an example of visualization created by our simulation software, during an online simulation. These visualizations have proved to be extremely helpful to understand the algorithms behavior, to compare their results and to develop a good intuition of the problem, and ultimately to design the `backbone` online algorithm.

Figure 5 shows how prior information influences the different online algorithms, in a high demand scenario with 5 minutes time windows. `backbone` performs significantly better than `2-OPT`, and the similarity of the two curves confirms the similarity of the two re-optimization approaches. The extra percents of profit, from 1% to 3.5% between these two methods are significant in practice, as they represent hundreds of additional customers that have been served thanks to optimization. The sharp increase of profit for the first few additional minutes of prior request time at the beginning of the curve is

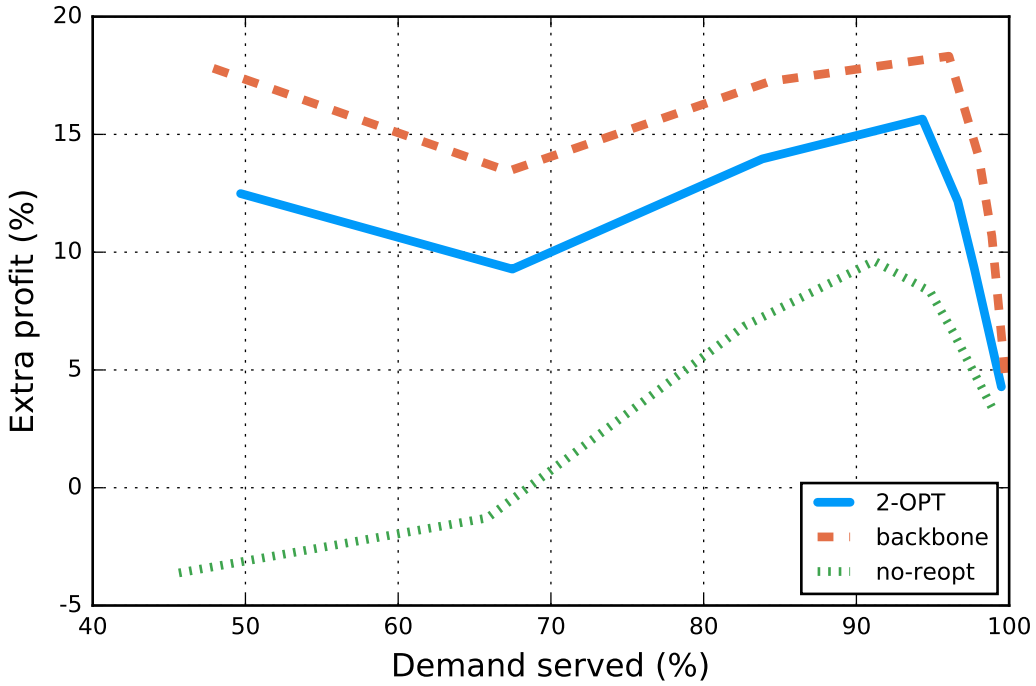

Figure 6: Varying the supply-demand balance. Increase in profit of each online taxi routing algorithm compared to `pure-online`. We vary the number of taxis from 2000 to 10000, and represent on the x-axis the corresponding fraction of customers served by the different algorithms. The time windows have a length of $T^{wait} = 5$ minutes and the mean prior request time is $T^{request} = 15$ minutes.

experienced by all online methods using prior information. It is explained by the additional time available to dispatch taxis to customers that are further away, and that `pure-online` cannot pick-up because the 5 minutes time window is too short. Nonetheless, `no-reopt` plateaus when more information is available, and cannot use the increasing prior request time to make better decisions. This is typically the situation in which re-optimization is important: due to high demand, all the `no-reopt` taxis are assigned to customers and we cannot accept new ones. On the other hand, re-optimization allows the option to reorganize the assignment of customers to taxis in order to be able to pick-up more customers, more efficiently. The surprising finding is that not a lot of prior information is needed in order to make better decisions: asking customers to request for a ride 10 minutes beforehand already allows for an 18% increase in profits.

Figure 6 shows how the balance between supply and demand influences

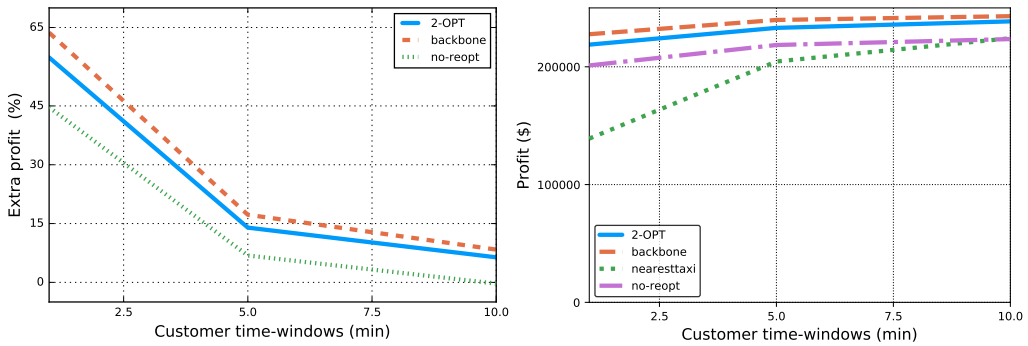

Figure 7: Varying the time windows size. Increase in profit of each online taxi routing algorithm compared to `pure-online` (left) and in absolute profit values (right). We vary the size T^{wait} of the customer time windows: $T^{wait} =$ 1 minute, $T^{wait} = 5$ minutes and $T^{wait} = 10$ minutes. There are 4000 taxis, which corresponds to a high demand scenario, and the mean prior request time is $T^{request} = 15$ minutes.

the results of our algorithms. We have showed in Section 3.4 that optimization-based algorithms and `2-OPT` have a strong edge on their greedy counterpart when demand was high. These results confirm this observation in an online setting: when the served demand is below 95%, we do not have enough taxis to serve the high demand. Thus `2-OPT` and `backbone` perform significantly better than the greedy algorithms `no-reopt` and `pure-online`, and `backbone` clearly outperforms `2-OPT`. We have found that problems with more demand than available taxis (in this case with less taxis) are generally harder to solve by offline solution methods in the re-optimization process. Given our limited computational time, this difficulty reduces the quality of the solutions found in the allowed time. This phenomenon is illustrated by the loss of performance at around 70% of served demand: this problem was the hardest to solve and there was not enough time given to the solution methods to find near-optimal solutions. When demand is low, and the served demand is close to 100%, taxis are generally mostly idle, and a greedy algorithm like `no-reopt` performs almost as well as the optimization-based algorithm. This confirms the insight we gained in Section 3.4 that problems with low demand are easy to solve and do not require re-optimization.

Figure 7 shows the impact of the time window length on the quality of the different solution methods. We also represented the profit values on the right to give a sense of scale. The sharp decrease in relative profit of the online algorithms in comparison to `pure-online` is actually due to an increase in quality of the `pure-online` solutions, which masks the fact that all strategies give better results with larger time windows. For $T^{wait} = 1$

minute, `pure-online` does not manage to pick-up customers when there is no free taxi in the close vicinity, and performs really poorly. Interestingly Figure 7 illustrates that `no-reopt` is no better than `pure-online` for very large time windows, which makes sense as the two greedy heuristics are almost equivalent in this setting. The prior information accessed by `no-reopt` is not useful when the time windows are long enough. As a consequence, the extra 10% of profit obtained by `backbone` for $T^{wait} = 10$ minutes is only due to the edge of re-optimization over greedy algorithms, as revealed in Section 3.4. Even with large time windows, re-optimization methods are significantly better than `pure-online` when some prior request information is available. Moreover, even if `pure-online` manages to use the large time windows to pick-up more customers, these pick-ups are generally later than the three other algorithms, giving them a strong edge in practice for customer satisfaction. In general, large time windows are better represented using soft time windows constraints, penalizing the delay.

We have compared all our results to the `pure-online` profit, as it is representative of typical taxi system greedy strategies. This empirical study shows that using optimization based strategies on today's relevant large scale vehicle transportation systems can have a serious impact on their performance, particularly in the daily situations of peak demand. Furthermore, our experiments suggest that these systems should give incentives to customers to request their trips few minutes in advance. Customers flexibility in pick-up time should also be used as much as possible, and time windows could be personalized for each customer, with an incentive to accept a larger one.

6 Conclusions

6.1 Extensions

Using historical data, it is possible to accurately forecast the demand in large scale settings and use it to route idle taxis to areas of popular demand. We did not use this in our application, but such an extension can improve the system efficiency, especially when there is a large cyclical demand in far-away locations like airports. Another way to use historical and real-time data is to provide online estimate of the travel times. In our applications in NYC, we have estimated the travel-times from data, under the assumption that they are stationary for the time of the experiment. In practice, travel-times can be re-estimated at each step of the re-optimization process.

We used the assumption of full control of the vehicles, as we expect that vehicle control will become increasingly centralized in the future. However,

the re-optimization framework can be adapted to be more of a recommendation system, suggesting customers to drivers, and updating the planning at each iteration given the vehicles actual moves. More generally, this framework is also suitable to other real-time vehicle routing applications. As our algorithm use a mixed-integer optimization at the core, we could add extra operational constraints to represent situations as diverse as cargo ship routing, on-demand private jets, bus renting, electric vehicles, self-driving taxis, car-pooling and more.

6.2 Impact

Our contributions surpass the scope of this paper in two ways:

First, the core ideas of our main algorithms `K-neighborhood`, `backbone` and `local-backbone` are not specific to taxi routing and can be applied to other large scale decision problems of vehicle routing and operations research. The core idea of a "backbone" is that some decision variables do not vary too much across almost all near-optimal solutions, and that identifying them can significantly accelerate the optimization process. This idea can be applied in a variety of situations, and is much more general than taxi routing. For example, [30] presents a backbone algorithm for the TSP, though formulated as a greedy heuristic. The part of a backbone algorithm that depends on the application is how to generate "good" and varied feasible solutions in a cheap way: we use `maxflow` for our taxi routing application. `local-backbone` goes even one step further: if it is too expensive to construct the problem backbone, one can do it iteratively, at each step constructing a local backbone around the current best solution to improve on it. This general algorithm has the advantage of combining global-optimization to avoid local extrema and local-improvement for tractability.

Additionally, the software we have built and released (see [20]) is able to simulate and visualize online and offline vehicle routing problems with synthetic or real-world routing data, using real or generated demand data. Being able to simulate real-world vehicle routing, our framework and algorithms can solve problems that are relevant to the industry. For example, the insights we get about the value of future information can be of immediate practical interest for current urban transportation companies.

Acknowledgement

We would like to thank the associate editor and three reviewers for their numerous helpful suggestions and comments. Research funded in part by

ONR grants N00014-12-1-0999 and N00014-16-1-2786. Geographical data for New York City is copyrighted to OpenStreetMap contributors and available from [24].

A Insertions and 2-OPT heuristics

We present in this appendix the details of our offline greedy and local improvement heuristics. We use these algorithms as a baseline to evaluate the effectiveness of our optimization-based algorithms.

A.1 Insertions and Greedy Heuristic

Given a solution s to the offline taxi routing problem and a customer c that is rejected in this solution, an *insertion* of c into s is the process of finding a taxi that is able to pick-up c without modifying the rest of the solution. For example, if a taxi in s is supposed to serve customers $c_1, c_2, \ldots, c_n$ in this order, inserting customer c at position k corresponds to modify s so that the taxi serves customer $c_1, \ldots, c_{k-1}, c, c_k, \ldots, c_n$, under the condition that the solution is still feasible.

The most important thing when inserting a customer is to be able to check the feasibility of the insertion, given the pick-up time window constraints. It is possible to do this in a very efficient way: given a solution s and a customer c, let $[t_{c,s}^{min}, t_{c,s}^{max}]$ be the interval of possible pick-up times t_c such that s is still feasible. These times are defined in Section 4.3 and can be computed quickly using forward induction. We can use them to quickly check the feasibility of inserting a customer. If we want to insert customer c within taxi k's schedule, we can compute the feasible time windows using the induction equations (16)-(19). For example, to insert customer c between c_{k-1} and c_k, we first compute its values $t_{c,s}^{min}$ and $t_{c,s}^{max}$ using equations (16) and (17):

$$t_{c,s}^{min} = \max(t_c^{min}, t_{c_k,s}^{max} - T_{c,c_k}) \tag{20}$$

$$t_{c,s}^{max} = \min(t_c^{max}, t_{c_{k-1},s}^{min} + T_{c_{k-1},c}) \tag{21}$$

and the insertion is only feasible if the pick-up time window is non-empty, i.e, $t_{c,s}^{min} \leq t_{c,s}^{max}$.

For each possible insertion, we can compute the difference in profit ΔR in the new solution after insertion. For example, when inserting c between c_{k-1} and c_k, we have:

$$\Delta R = R_{c_{k-1},c} + R_{c,c_k} - R_{c_{k-1},c_k} \tag{22}$$

We can now use these insertions in an iterative way to describe the `greedy` heuristic introduced in Section 3.2:

1. Create an "empty" solution s, in which all customers are rejected and all taxis idle.

2. Order all the customers by minimum pick-up time t_c^{min}, and apply the next steps to each one, sequentially.

3. Given a customer c to insert, try to insert it in each taxi using the feasibility rules described in this section with the values $[t_{c,s}^{min}, t_{c,s}^{max}]$ of the other customers.

4. If no inserting position is feasible, reject the customer.

5. If inserting the customer is feasible, select the taxi and the position that yield the highest difference in profit ΔR, and insert the customer.

6. Update the values $[t_{c,s}^{min}, t_{c,s}^{max}]$ for all the customers that are assigned to the taxi chosen for the insertion, using equations (16)-(19).

Inserting the customers by order of t_c^{min} performs typically really well, and is very close to the nearest-taxi strategy, as each customer will be inserted at the end of a taxi's schedule, usually the taxi that is the closest to the customer.

A.2 Local-Improvement and 2-OPT

Let s be a solution to the offline taxi routing problem, a local improvement is a solution s' that is in a "neighborhood" of s, such that the total profit of s' is higher than the profit of s. A simple yet powerful definition of such a neighborhood is the *2-OPT* neighborhood. We perform a swap between two nearby taxis, exchanging their assigned customers. For example if taxi 1 is picking up customers c_1^1, c_2^1, c_3^1 and taxi 2 is picking-up customers c_1^2, c_2^2, c_3^2, swapping customer c_2^1 and c_2^2 (together with the subsequent customers) could result in assigning c_1^1, c_2^2, c_3^2 to taxi 1 and c_1^2, c_2^1, c_3^1 to taxi 2. Formally, we execute the following algorithm:

1. Given a solution s, choose a customer c that is already assigned to taxi k, let $c_1^k, \ldots, c_n^k$ be the customers assigned to k whose pick-up times are after customer c. Let also c_{-1}^k be the customer coming immediately before c in taxi k's schedule.

2. Select another taxi k'. Let customer c' be the first customer of k' such that $t^{min}_{c',s} + T_{c',c} \leq t^{max}_{c,s}$. In other words, c' is the first customer assigned to k' such that k' can serve all its customer preceding c', followed by $c', c, c^k_1, \ldots, c^k_n$.

3. Let $c^{k'}_1, \ldots, c^{k'}_n$ be the customers assigned to k' whose pick-up times are after customer c' in solution s. Remove these customers from k', and assign $c, c^k_1, \ldots, c^k_n$ to k' after c' instead.

4. Find the first customer $c^{k'}_i$ of the sequence $c^{k'}_1, \ldots, c^{k'}_n$ such that $t^{min}_{c^k_{-1},s} + T_{c^k_{-1},c^{k'}_i} \leq t^{max}_{c^{k'}_i,s}$. In other words, find the longest sub-sequence $c^{k'}_i, \ldots, c^{k'}_n$ such that all these customers can be inserted at the end of taxi k's schedule, immediately after customer c^k_{-1}, while respecting the pick-up time windows.

5. Assign customers $c^{k'}_i, \ldots, c^{k'}_n$ to taxi k. And reject the customers $c^{k'}_1, \ldots, c^{k'}_{i-1}$ that we could not insert.

6. At this point of the swap, taxi k schedule is now $\ldots, c^k_{-1}, c^{k'}_i, \ldots, c^{k'}_n$ and taxi k' schedule is now $\ldots, c', c, c^k_1, \ldots, c^k_n$. Customers $c^{k'}_1, \ldots, c^{k'}_{i-1}$ are rejected.

7. Use the insertion algorithm described in Section A.1 to try to insert all the customers that were rejected in s into k and k' schedules, the only two taxis that we have modified.

8. Also use the insertion algorithm to try to insert the newly rejected customers $c^{k'}_1, \ldots, c^{k'}_{i-1}$ in all taxis schedule.

9. We have built our final solution s'. Compute its profit and compare it with the previous one.

This construction of a new solution may seem elaborate, because of its need to respect the time windows feasibility. However, it is in practice very fast as it only modifies a small sub-part of the solution. Steps 1. and 2. are the two most important, as we choose the two taxis and customers on which we will perform the swap. To make it tractable on a large scale such as our application in Manhattan in Section 5.3, we use the costs described in Section 4.3 to smartly choose good potential swaps. In practice, we were able to perform 10,000 swaps per minute in the large scale online taxi problem in NYC introduced in Section 4.4.

We use these swaps to perform a local-improvement descent, only accepting a 2-OPT swap when the profit is improved, as described here:

1. Begin with a solution s as given by `greedy`.
2. Perform a 2-OPT swap on s. If the profit is improved, update s to be this new solution.
3. If there is time left, go back to Step 2.

We call this offline algorithm `2-OPT`. Note that all solutions s in this algorithm share the invariant that no customer rejected in s can be inserted in s. Indeed, `greedy` respect this invariant, and steps 7. and 8. make sure that we try all new insertion possibilities at each swap. On small instances of taxi routing (less than a few hundred customers), we have noticed that `2-OPT` tends to converge very fast to a locally optimal solution. In large cities with thousands of customers, we usually do not have enough time to reach a locally optimal solution. The algorithm is slowed down by the high dimensionality of the routing problem, though it manages to significantly improve the solutions quality. This is a sign that more complex local-improvement algorithms, like `3-OPT` modifying 3 taxi's schedules at a time, could not really help with large scale problems, as we do not even have enough time to sufficiently explore the `2-OPT` neighborhood. The same applies for more complex global-local algorithms like *Tabu-search*.

References

[1] Roberto Baldacci, Aristide Mingozzi, and Roberto Roberti. "Recent exact algorithms for solving the vehicle routing problem under capacity and time window constraints". In: *European Journal of Operational Research* 218.1 (2012), pp. 1–6.

[2] Russell Bent and Pascal Van Hentenryck. "Scenario-Based Planning for Partially Dynamic Vehicle Routing with Stochastic Customers". In: *Operations Research* 52.6 (2004), pp. 977–987.

[3] Russell Bent and Pascal Van Hentenryck. "Waiting and relocation strategies in online stochastic vehicle routing". In: *IJCAI International Joint Conference on Artificial Intelligence* (2007), pp. 1816–1821.

[4] Gerardo Berbeglia, Jean-François Cordeau, and Gilbert Laporte. "A Hybrid Tabu Search and Constraint Programming Algorithm for the Dynamic Dial-a-Ride Problem". In: *Journal on Computing* 24.3 (2012), pp. 343–355.

[5] Gerardo Berbeglia, Jean-François Cordeau, and Gilbert Laporte. "Dynamic pickup and delivery problems". In: *European Journal of Operational Research* 202 (2010), pp. 8–15.

[6] Dimitris Bertsimas et al. “Traffic Estimation in the Age of Big Data”. In: *Submitted* (2016).

[7] Jeff Bezanson et al. “Julia: A Fresh Approach to Numerical Computing”. In: (Nov. 2014). arXiv: `1411.1607 [cs.MS]`.

[8] Olli Bräysy and Michel Gendreau. “Vehicle Routing Problem with Time Windows, Part I: Route Construction and Local Search Algorithms”. In: *Transportation Science* 39.1 (2005), pp. 104–118.

[9] Zhi-Long Chen and Hang Xu. “Dynamic Column Generation for Dynamic Vehicle Routing with Time Windows”. In: *Transportation Science* 40.1 (2006), pp. 74–88.

[10] G. A. Croes. “A Method for Solving Traveling-Salesman Problems”. In: *Operations Research* 6.6 (1958), pp. 791–812.

[11] Guy Desaulniers et al. “Exact Algorithms for Electric Vehicle-Routing Problems with Time Windows”. In: *Operations Research* 64.6 (2016), pp. 1388–1405.

[12] Michel Gendreau, Alain Hertz, and Gilbert Laporte. “A Tabu Search Heuristic for the Vehicle Routing Problem”. In: *Management Science* 40.10 (1994), pp. 1276–1290.

[13] Gabriel Gutiérrez-Jarpa et al. “A branch-and-price algorithm for the Vehicle Routing Problem with Deliveries, Selective Pickups and Time Windows”. In: *European Journal of Operational Research* 206.2 (2010), pp. 341–349.

[14] Mordechai Haklay and Patrick Weber. “OpenStreetMap: User-Generated Street Maps”. In: *IEEE Pervasive Computing* 7.4 (Oct. 2008), pp. 12–18.

[15] Hideki Hashimoto et al. “The vehicle routing problem with flexible time windows and traveling times”. In: *Discrete Applied Mathematics* 154.16 (2006), pp. 2271–2290.

[16] Mark Horn. “Fleet scheduling and dispatching for demand-responsive passenger services”. In: *Transportation Research Part C: Emerging Technologies* 10.1 (2002), pp. 35–63.

[17] Patrick Jaillet and Michael Wagner. “Generalized Online Routing: New Competitive Ratios, Resource Augmentation, and Asymptotic Analyses”. In: *Operations Research* 56.3 (2008), pp. 745–757.

[18] Miles Lubin and Iain Dunning. “Computing in operations research using Julia”. In: *INFORMS Journal on Computing* 27.2 (2015), pp. 238–248.

[19] Shuo Ma, Yu Zheng, and Ouri Wolfson. "Real-Time City-Scale Taxi Ridesharing". In: *IEEE Transactions on Knowledge and Data Engineering* 27.7 (2015), pp. 1782–1795.

[20] Sebastien Martin. *TaxiSimulation Julia Package*. `https://github.com/sebmart/TaxiSimulation`. [Online; accessed 23-January-2018]. 2017.

[21] Fei Miao et al. "Taxi Dispatch With Real-Time Sensing Data in Metropolitan Areas: A Receding Horizon Control Approach". In: *IEEE Transactions on Automation Science and Engineering* 13.2 (2016).

[22] Snežana Mitrović-Minić, Ramesh Krishnamurti, and Gilbert Laporte. "Double-horizon based heuristics for the dynamic pickup and delivery problem with time windows". In: *Transportation Research Part B: Methodological* 38.8 (2004), pp. 669–685.

[23] NYC. *New York City Taxi & Limousine Commission - Trip Record Data*. `http://www.nyc.gov/html/tlc/html/about/trip_record_data.shtml`. Accessed: 2017-02-12. 2017.

[24] OSM. *OpenStreetMap Project Database*. `http://planet.openstreetmap.org`. 2017.

[25] Masayo Ota et al. "A scalable approach for data-driven taxi ride-sharing simulation". In: *Proceedings - 2015 IEEE International Conference on Big Data, IEEE Big Data 2015*. Dec. 2015, pp. 888–897.

[26] Masayo Ota et al. "STaRS: Simulating Taxi Ride Sharing at Scale". In: *IEEE Transactions on Big Data* (2016), pp. 1–1.

[27] Victor Pillac et al. "A Review of Dynamic Vehicle Routing Problems". In: *Cirrelt-2011-62* (2011), pp. –28.

[28] Jean-Yves Potvin and Jean-Marc Rousseau. "An Exchange Heuristic for Routeing Problems with Time Windows". In: *The Journal of the Operational Research Society* 46.12 (1995), pp. 1433–1446.

[29] Paolo Santi et al. "Quantifying the benefits of vehicle pooling with shareability networks." In: *Proceedings of the National Academy of Sciences* 111.37 (2014), pp. 13290–4.

[30] Johannes Schneider et al. "Searching for backbones -an efficient parallel algorithm for the traveling salesman problem". In: *Computer Physics Communications* 96 (1996), pp. 173–188.

[31] K.I. Wong and Michael Bell. "The Optimal Dispatching of Taxis under Congestion : a Rolling Horizon Approach". In: *Advanced Transportation* 40.2 (2005), pp. 203–220.

[32] Zhihai Xiang, Chengbin Chu, and Haoxun Chen. “A fast heuristic for solving a large-scale static dial-a-ride problem under complex constraints”. In: *European Journal of Operational Research* 174.2 (2006), pp. 1117–1139.

[33] Ci Yang. “Data-Driven Modeling of Taxi Trip Demand and Supply in New York City”. PhD thesis. 2015.

[34] Jian Yang, Patrick Jaillet, and Hani Mahmassani. “Real-Time Multivehicle Truckload Pickup and Delivery Problems”. In: *Transportation Science* 38 (2004), pp. 135–148.

[35] Rick Zhang, Federico Rossi, and Marco Pavone. “Routing Autonomous Vehicles in Congested Transportation Networks: Structural Properties and Coordination Algorithms”. In: *Proceedings of Robotics: Science and Systems* (2016).