Learning implicit multiple time windows in the Traveling Salesman Problem

Jorge Mortes, Martin Cousineau, Fabien Lehuédé, Jorge E. Mendoza, Maria I Restrepo

Learning implicit multiple time windows in the Traveling Salesman Problem

Jorge Mortes[a,b,*], Martin Cousineau[b], Fabien Lehuédé[a], Jorge E. Mendoza[b], María I. Restrepo[a]

[a] *IMT Atlantique, LS2N, UMR CNRS 6004, 4 Rue Alfred Kastler, Nantes F-44000, France*
[b] *HEC Montréal, 3000 Chemin de la Côte-Sainte-Catherine, Montréal H3T 2A7, Canada*

Abstract

Classically, researchers working in vehicle routing problems (VRPs) assume that the structure of the problem is known (i.e., objective function, constraints, parameters). However, recent studies have highlighted the gap between the routes offered by classical optimization algorithms and the routes followed by experienced drivers. As a result, researchers have turned their attention towards the acquisition and inclusion of drivers' knowledge to learn the order in which each customer is going to be served by the driver. In this study, we describe and solve a new problem called the multiple time window learning problem. In contrast to other VRP variants, the goal is to learn the time windows associated with each customer. Our approaches are based on the observation and exploitation of historical data with a new algorithm called the recall heuristic, and the exploration of new information based on the multi-armed bandit problem. Computational results based on real data extracted from a traffic sign dataset from the city of Montreal showed that our approaches can learn time windows and, as a result, propose routes similar to those created by experienced drivers, while still minimizing the routing costs.

Keywords: Multi-Armed Bandit, Traveling Salesman Problem with Multiple Time Windows; Reinsertion Algorithm.

1. Introduction

In the pandemic and post-pandemic world, retailers have seen a double-digit yearly growth in e-commerce (Dumanska et al., 2021). To be able to respond to the fast-increasing number of home deliveries while keeping their cost under control, retailers have been forced to look for alternative and creative ways to complete the last-mile.One such way is crowdsourcing platforms like Amazon Flex. In a nutshell, Amazon Flex is a platform where the company Amazon obtains independent drivers to perform delivery routes via a smartphone app where the drivers sign up and accept or decline routes. Additionally, those independent drivers are considered freelancers. One of the main problems that these companies that use crowdsourcing platforms have to deal with is the difference between the routes that the companies propose to the drivers and the routes that the drivers actually do (Samson and Sumi, 2019; Winkenbach et al., 2021).

These differences suggest that drivers have implicit knowledge (e.g., traffic conditions, availability of parking spots, school schedules, waste collection) which is not captured in most vehicle routing software. A large difference in duration between the routes that the drivers accept (and get paid for) and the ones that they perform, may generate dissatisfaction for the drivers and the companies: for the drivers because the offered route might have a lower duration (and hence a lower cost) than the real one, and for the companies because they assume a low performance coming from the drivers and they will not propose more routes for

*Corresponding author. E-mail address: jorge.mortes-alcaraz@imt-atlantique.fr

these drivers. Ultimately, these dissatisfactions might result in the use of more expensive courier services, due to the lack of drivers accepting the routes proposed in the crowdsourcing platform. In this line, acquiring drivers' knowledge through learning mechanisms leads to more efficient and realistic routes that are more likely to be used by the drivers without any intervention (i.e., the offered route better reflects the real costs) and without drastically increasing the companies' routing costs.

There exists previous work about the use of learning mechanisms in vehicle routing problems (VRP). Nevertheless, research has mainly focused on learning the sequence in which the customers are visited (Mandi et al., 2021; Wu et al., 2022). In this study, we assume that the knowledge that can be learned from the drivers goes further than sorting customers and might be related to the period of time when a specific neighborhood should be visited (i.e., time windows). Furthermore, by learning time windows, our framework is able to offer routes that could not be offered by learning customers' sequences from the drivers. However, the higher flexibility of our algorithms comes with the cost of a more complex routing problem. Another difference with previous work is that the algorithms presented in this study do not need historical data to start working with and efficiently validate or invalidate time windows while interacting with the driver and collecting new data.

Taking this into account, we can describe the main processes of a company to plan the routes and learn the drivers' knowledge as follows. At the beginning of day t, the company considers a subset of customers that must be served. Then, with the current knowledge of the customers' time windows, the company uses software to design a route that serves all the customers in a cost-effective way. We call this route the *software route*. After that, when considering the driver's awareness of the customers' real time windows, this route may be impractical or inefficient for the driver, and, due to their knowledge, the driver corrects the infeasibilities and performs a corrected route. We call this route the *driver's route*. Finally, at the end of day t, the company compares the *software route* to the *driver route* and applies a learning mechanism to try to acquire the drivers' knowledge. This learning procedure allows the company to propose better routes to the drivers in subsequent days. A flowchart of the company's process is shown in Figure 1.

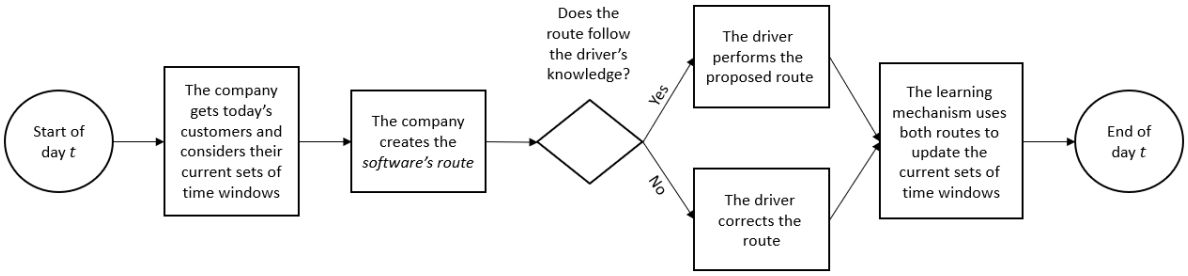


Figure 1: Flow chart outlining the different parts of the procedure on day t

In this study, we tackle the problem of learning real time windows from the drivers' knowledge, which corresponds to the end of the process before the day t ends. This process can be done simply by mimicking the drivers' historical data or by adding learning mechanisms. We first address this learning problem with the use of a heuristic that imitates the behavior of the drivers, which we called the recall heuristic (RH). After that, we add multi-armed bandit (MAB) algorithms (Lattimore and Szepesvári, 2020) as an exploration mechanism to learn customers' sets of time windows faster. MAB algorithms use a sophisticated procedure that balances the exploitation of historical data and the exploration of new data in search of valuable information about the correct time windows. Concerning the previous three events of the process (i.e., receiving the list of customers, creating the *software route*, and the route correction by the drivers) as we could not

find real data in the literature, we propose a simulation mechanism that uses a traveling salesman problem with multiple time windows (TSP-mTW) mathematical programming model to find the *software route* and an insertion algorithm to create the *driver's route* following the drivers' knowledge. This data simulation allows us to test the learning procedures.

The main contributions of this study are the following: we present the multiple time window learning problem (mTWLP), a new problem that has not been studied in the literature and that is relevant to real-life applications; we propose various approaches to solve this problem, in particular, the RH learning mechanism based on mimicking driver's historical data as well as various MAB algorithms, and we develop problem-specific optimization-based mechanisms to speed up and improve the scalability of the MAB algorithms.

The structure of the document is the following: Section 2 presents a literature review of the mechanisms to learn problem characteristics in VRP. Section 3, formally defines the mTWLP. Section 4 details the RH algorithm, and how to apply four MAB algorithms to the mTWLP. Section 5 details the computational experiments and experimental setup (i.e., validation process and instance creation). To end the document, Section 6 presents some conclusions.

2. Learning problem characteristics in VRP

Learning characteristics of a mathematical programming model (e.g., constraints, objective function, parameters) to offer realistic solutions that resemble the ones built by experts is a fast-growing field (den Hertog and Postek, 2016). Several methods have been proposed to approach the acquisition of these characteristics, including inverse optimization (Chan et al., 2021); machine learning (Kubat, 2017); data-driven optimization (Hewitt and Frejinger, 2020); and inverse reinforcement learning (Arora and Doshi, 2021). Inverse optimization uses optimal solutions as input to determine the parameters of the model that produced those input solutions. In the survey of Chan et al. (2021), the authors divide inverse optimization into two parts: classical inverse optimization, which learns the parameters of the model to produce the same solutions used as inputs (Shahmoradi and Lee, 2021; Bodur et al., 2022; Ghate, 2020), and data-driven inverse optimization, which allows infeasible solutions but penalizes infeasibilities with the use of loss functions (Aswani et al., 2018; Chan and Kaw, 2020). With the same objective in mind, but with different techniques, there are some works using machine learning methods (e.g., artificial neural networks) to learn the constraints of the model (Lombardi et al., 2017). Data-driven optimization is another technique that tries to learn and add *side constraints* to an original, previously defined, model (Hewitt and Frejinger, 2020). Lastly, inverse reinforcement learning transforms the problem into a Markov decision process defining pairs of state-actions to predict the reward function that makes adopted actions optimal (Fu et al., 2018).

Learning the driver's preferences in vehicle routing is one of the streams of research relying on the aforementioned methods. Chow and Recker (2012) propose an inverse optimization model to estimate the objective parameters and some constraints in the household activity pattern problem (HAPP). The HAPP aims to predict the optimal path of household members as they complete a prescribed agenda of out-of-home activities by analyzing their movement through time and space. The connection between the HAPP and VRP problems is that Recker (1995) modeled the former as a variant of the Pick-up and Delivery Problem with Time Windows. In Chow and Recker (2012), the authors suggest an approach to fit the HAPP model by jointly estimating the objective coefficients and goal arrival times using an inverse formulation. It is important to remark that this work predicts goal times at the activities by adding a cost to the deviation from the goal time. The authors also propose an inverse model to predict soft time windows. Some years later, You et al. (2016) adapted this HAPP inverse optimization model to fit an activity-based freight forecast model. In Chen et al. (2021), the authors use an inverse optimization model to learn the drivers' cost matrix in a capacitated VRP. In a similar way, the authors in Canoy and Guns (2019) and Mandi et al. (2021)

create a transition probability matrix using a weighted Markov counting approach and a neural network, respectively. Once the transition probability matrix is estimated, the authors solve a VRP problem by selecting the route that maximizes the sum of probabilities or, as the authors call it, *maximum likelihood routing*. Some examples of inverse reinforcement learning techniques can be found in Ziebart et al. (2008) and Snoswell et al. (2020), in which the authors model the driver's route prediction as a Markov decision process (i.e., the driver has to decide the path to follow at each intersection) and try to recover the reward function producing the decisions taken based on the maximum entropy principle (Jaynes, 1957). More recently, Dieter et al. (2023) proposed a framework that combines machine learning with optimization techniques to propose efficient routes that resemble the ones done by experienced drivers. To do so, the authors train a neural network to predict the customer sequence and, after, solve an optimization problem to create another route but limit the tour length deviation from the route predicted by the neural network.

Finally, the *Amazon Last Mile Routing Research Challenge* (Winkenbach et al., 2021) is a good example of the increased popularity of the topic of learning in transportation problems. In this challenge, Amazon made available a dataset containing more than 4,000 driver routes to research teams in order to learn drivers' preferences and recreate drivers' routes. In the challenge's final phase, different techniques were used in more than 30 technical reports addressing the issue of learning drivers' preferences. However, to the best of our knowledge, the issue of learning customers' time windows as a hard constraint has not been yet studied in the literature.

3. Problem description

Let $G = (V, A)$ be a graph, where $V = \{0, 1, \ldots, N\}$ is the set of vertices and $A = \{(i, j) \mid i, j \in V,\ i \neq j\}$ is the arc set. We divide the set of vertices as $V = \{0\} \cup C$, where the vertex 0 corresponds to the depot and the set C contains all the possible customers. A non-negative travel time matrix $D = [d_{ij}]$ is associated with every arc (i, j), $i \neq j$, where the element d_{ij} corresponds to the travel time between the vertices i and j. Each day $t \in \{0, 1, \ldots, T\}$ the company serves a set $C_t \subset C$ of customers $c \in C_t$ to serve. For the mTWLP, we refer to the last part of the company's procedure described in Section 1. This process corresponds to the end of the day when the driver comes back to the depot and applies the learning mechanism. At that moment the company compares both the *software route* that was proposed to the driver with the *driver's route*. We denote the *software route* $\hat{z}$ as the output of $f(C_t, \hat{y}_{C_t})$, where f is a minimum tour duration TSP-mTW algorithm used by the company to build the route, and $\hat{y}_{C_t}$ is the company's current knowledge about day t customer's set of time windows. On the other hand, we denote the *driver's route* $\bar{z}$ as the output of $g(C_t, y_{C_t}, \hat{z})$, where g is the procedure used by the driver to repair route $\hat{z}$, and y_{C_t} is the set of real time windows (known only to the driver) for the customers at day t. To compare both routes, the company has access to the following data: the order and time in which customers were expected to be visited (*software route*), and the order and time in which customers were actually visited (*driver's route*). We denote the learning procedure as $\pi(\hat{y}_{C_t}, \hat{z}, \bar{z})$.

Following the notation defined above, the mTWLP consists in proposing an online learning mechanism $\pi(\hat{y}_{C_t}, \hat{z}, \bar{z})$ for each customer that, using as input the information about the routes $\hat{z}$ and $\bar{z}$, is capable of proposing sets of time windows $\hat{y}_{C_t}$ containing time windows from the real sets of time windows (y_{C_t}). If the learning mechanism succeeds, the routes generated by algorithm f should converge to the ones that the driver corrects using algorithm g (i.e., the routes $\hat{z}$ will be more realistic and will satisfy the drivers), and the routes will be more efficient (i.e., as $\hat{y}_{C_t}$ will contain more precise information, algorithm f will propose routes with lower costs to the company).

For the remainder of the study, we divide a day with duration H into n different time periods of the same length H/n. We define P as the set that contains the n time periods. In this way, we assume that the set of

time windows of a specific customer y_c corresponds to a subset of P, in other words, $y_c \subset P$, $\forall c \in C$. To illustrate this in a more detailed way, let H be equal to 80 and n to 8. In that case, the time horizon is divided into 8 consecutive time periods of 10 time units (i.e., $P = \{[0,10),[10,20),\ldots,[70,80)\}$). A valid set of time windows for a customer c is, for example, $y_c = \{[10,20),[40,50),[50,60)\} \subset P$, while the set of time windows $y_c = \{[10,25),[38,50)\} \subset P$ is not valid. In the first case, we will abuse the notation simply by saying $y_c = \{[10,20),[40,60)\} \subset P$. Figure 2 illustrates this example.

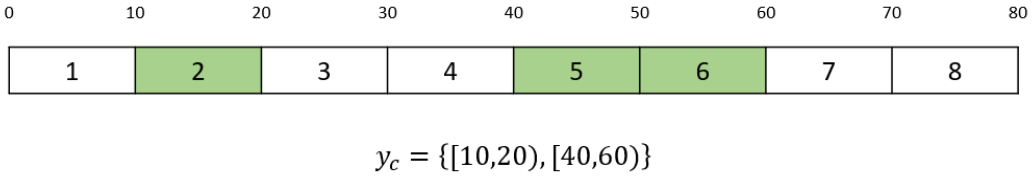


Figure 2: Example of a valid set of time windows for a customer if $H = 80$ and $n = 8$

4. Solution methods

This section discusses how we adapt learning algorithms to tackle the mTWLP. As we mentioned in Section 1, we are going to approach this problem in two different ways: by directly and solely utilizing the information provided by the *driver's route* and by enhancing the first approach using an exploration strategy that may enable to learn and predict the customers' real set of time windows faster. In Section 4.1 we present RH, an algorithm that purely exploits data from previous rounds, and in Section 4.2 we talk about how we adapt MAB algorithms to add exploration in the mTWLP.

4.1. Recall heuristic

One of the most straightforward ways to learn the drivers' knowledge about the customers' real set of time windows is to simply use historical data of past visiting times. To implement this principle in our process, we define the recall heuristic, denoted *RH*, which works as follows: each time a customer has to be served, RH proposes the minimal set of time windows that includes all the exact times at which the customer has been visited in the past, y_c. In other words, let Δ be the historical set of visit times including all the past visits to the customers, then $\Delta_c \subset \Delta$ is the set containing the historical visit times to customer c. Then, $y_c = \{p \in P \mid (\exists \delta \in \Delta_c \mid \delta \in p)\}$. In case there are no historical visit times for one specific customer (i.e., $\Delta_c = \varnothing$), the algorithm returns the entire time horizon.

A visual representation of this concept (following the same example as in Section 3) is shown in Figure 3: The time horizon $H = 80$ to serve one customer has been divided into $n = 8$ periods of 10 units of time each, and as the customer c has been visited in the past at times $\Delta_c = \{15,43,47,51\}$, RH proposes using the set of time windows $y_c = \{[10,20),[40,60)\} \subset P$ for this customer (highlighted in green in the figure).

It is important to remark that, even though RH does not need to be initialized with historical data, the second time that a customer is visited RH proposes only one time period. To avoid infeasibility problems (i.e., serving several customers with an identical time period on the same day), we consider a warm-up period. During the initialization period (i.e., $t < w$, where $w < T$, $w \in \mathbb{N}$ is the number of warm-up iterations) RH proposes a subset of the time periods ($\hat{P} \subset P$) for each of the customers. After the first w iterations, if there is still one customer c with no visits, the algorithm proposes y_c as the entire time horizon for the first time that c is visited, as explained above..

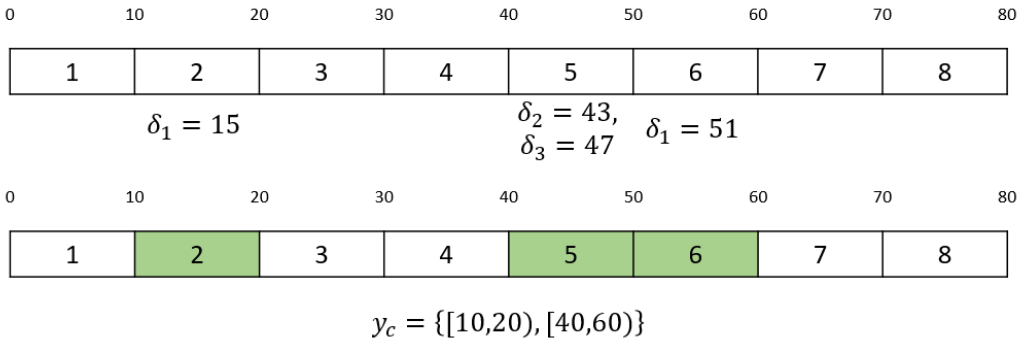


Figure 3: Example of using historical data to learn the real set of time windows

Algorithm 1 Recall heuristic

Input: Δ, P, $c \in C_t$, w, t
Output: y_c

1: **if** $t < w$ **then**
2: Select $\hat{P} \subset P$
3: $y_c \leftarrow \hat{P}$
4: **else**
5: **if** $\Delta_c \neq \varnothing$ **then**
6: $y_c \leftarrow \{p \in P \mid (\exists \delta \in \Delta_c \mid \delta \in p)\}$
7: **else**
8: $y_c \leftarrow P$
9: **end if**
10: **end if**

The pseudo-code of RH is shown in Algorithm 1. Line 1 checks if the number of iterations devoted to the warm-up procedure has been reached. If not, in lines 2 and 3 a subset $\hat{P} \subset P$ is selected and given to the customer. Otherwise, in lines 5 to 9, a set of time windows is given to the customer as detailed above.

Note, that, after the warm-up iterations, only time periods that are known to be correct are offered by this algorithm. Hence, the driver only changes a customer if there is a more efficient spot to serve it and, then, the learning pace is slow.

4.2. Multi-armed bandit algorithms

As mentioned in Section 1, an interesting strategy to learn the customers' set of time windows is to add an exploration mechanism to RH. The task of this mechanism is to predict which additional time windows might be promising for a specific customer and propose them along with the ones known to be correct thanks to the historical data. An exploration mechanism might help to identify time windows that are helpful to build an efficient route without waiting for the driver to visit it once (and change the route). In this work, we investigate the use of multi-armed bandits to learn mTW from the driver corrections to the software routes.

A MAB is a mechanism that models sequential decision problems. In each of the T iterations (i.e., days in the planning horizon), an agent must choose from a set of actions, also called arms. At the end of each iteration, the agent receives a reward that quantifies the performance of the chosen arm. In this way, the goal of a MAB is to gather enough information to maximize the overall reward after the T iterations.

In order to maximize the overall reward, at each iteration, the algorithm decides whether it exploits the current information about certain arms or it explores new arms to escape from a local optimum. Thus, one of the key challenges of MABs is to find an efficient mechanism to balance exploitation and exploration. As reinforcement learning mechanisms, MABs learn a model based on rewards. However, unlike reinforcement learning techniques, in MAB the state of the world is not affected by each taken decision.

Multi-armed bandit algorithms were introduced by Thompson (1933) in the context of a medical trial design problem, where the author presented a 2-armed bandit to decide between two rival medical treatments. Ever since, MABs have been used to solve a variety of problems. For instance, Villar et al. (2015) used them to solve another medical trial problem. Li et al. (2010) used a variation of a MAB algorithm to tackle the Personalized News Article Recommendation problem. These algorithms have also been used in economic problems such as dynamic assortment (Agrawal et al., 2019) or dynamic procurement (Badanidiyuru et al., 2018).

In this section, we explain four MAB algorithms detailing the key changes required to address the mTWLP. For the sake of simplicity, as we worked with two linear MAB algorithms and two combinatorial MAB algorithms, this subsection will be divided into two. In Section 4.2.1 we detail and introduce the adaptations to two linear MAB algorithms (i.e., LinUCB and LinTS), and in Section 4.2.2 we describe and modify two combinatorial MAB algorithms (i.e., CUCB and CTS).

4.2.1. Linear MAB algorithms

As their name suggests, linear MABs assume that the expected reward of each arm follows a linear function. We use the LinUCB (Li et al., 2010) and LinTS (Agrawal and Goyal, 2013) algorithms to tackle the mTWLP. In the articles, the authors assume that the expected reward of playing arm k (μ_t) is a linear combination of a vector $x_k \in \mathbb{R}^d$ describing the arm and context about the user (also called the features vector) with an unknown true parameter $\theta_t \in \mathbb{R}$ ($\mu_t = x_{kt}^\top \theta_t$). The authors try to estimate the unknown true parameters (i.e, $\hat{\mu}_t = x_{kt}^\top \hat{\theta}_t$), using different approaches. For a more detailed explanation of the parameters prediction, the reader is referred to the original articles.

In terms of the mTWLP, let P be a set of n periods of time and H be the time horizon, we define an arm $k \in K$ as a combination of periods of time from P (i.e., $k \subset P$). As we assume that we do not have context about the users, we define the features that describe one arm as a binary vector $x_k \in \{0,1\}^n$ characterized by

$$x_k^i = \begin{cases} 1, & \text{if } [p_i] \in k \\ 0, & \text{otherwise,} \end{cases} \tag{1}$$

where $[p_i]$ denotes the i-th time period in P. In the case that there was information about either the customer or the driver, extra features could be added to define this context (i.e., more entries in x_k). Following the same example as before (time horizon $H = 80$ divided into $n = 8$ periods of 10 time units), the feature vector assigned to the arm $k = \{[10,20),[40,60)\} \subset P$ is $x_k = (0,1,0,0,1,1,0,0)$. As each customer needs a non-empty set of time windows (and it has to be a subset of P), the algorithm will have to select at each iteration among $2^n - 1$ arms per customer. Hence, it is mandatory to search for the best arm efficiently as n increases.

There are different ways to define a reward in terms of the mTWLP using the information that the company has at the end of each day. We define the reward that we use in the linear MAB algorithms as:

$$r_t = -\left| t_c^{soft} - t_c^{dri} \right|, \tag{2}$$

where $t_c^{soft} \in \{0, 1, \ldots, H-1\}$ and $t_c^{dri} \in \{0, 1, \ldots, H-1\}$ represent respectively the proposed delivery time of customer c in the *software route* and the actual delivery time in the *driver's route*.

Hence, if the driver relocated one customer, that means that either the period of time where the customer was planned to be served is wrong or there exists another period of time that is more efficient. Thus, it is logical to penalize this set of time windows. On the other hand, if the delivery times of the *software route* and *driver's route* are the same, then it makes sense not to penalize the set of time windows since this time period of the arm is correct.

MAB algorithms are known for their mechanisms to find a trade-off between exploitation and exploration. However, as we detail in Section 5, in terms of the mTWLP, exploring incorrect arms over the iterations might have a big impact on the costs of the *software route*. Another well-known drawback of MAB algorithms is the so-called cold start. The cold start is produced due to the lack of historical information about the customers, and it is commonly addressed by a period of data collection. In order to address the cold start problem, we set a warm-up parameter $w < T$, $w \in \mathbb{N}$. During the first w iterations, in order to collect enough information, there is no limit in terms of exploration (i.e., the MAB is allowed to propose as many time periods as it wants per customer). Once the w warm-up iterations have finished, to prevent the costs of exploring many incorrect time periods, the algorithm is only allowed to explore up to one new time period (in addition to the ones known to be correct by previous visits). This will result in a high time slot learning pace during the warm-up period, which will be slowed down after the w iterations due to the limited exploration.

4.2.1.1. LinUCB. The upper confidence bound algorithm with linear rewards (LinUCB) was introduced in Li et al. (2010) and belongs to the family of UCB MAB algorithms. UCB algorithms are based on the *optimism principle under uncertainty*, which assumes an optimistic environment based on previous observations. In terms of the MAB, the optimism principle means calculating a value based on past historical data that overestimates each arm's unknown expected reward and selecting the arm with the best value at each iteration. This value is called the *upper confidence bound* and usually has two parts: an estimator $\hat{\mu}_t$ of the unknown expected reward μ_t and the *confidence width*, that estimates if the arm k still needs to be explored in order to gather more information. As might be anticipated, the estimator $\hat{\mu}_t$ converges to the real expected reward, whereas the confidence width of the arms gradually approaches zero every time these arms are selected. Thus, the UCB value, which is the sum of $\hat{\mu}_t$ and the confidence width, converges with high probability to μ_t. The first UCB algorithm was proposed by Lai (1987) and has been revisited multiple times until the current date. UCB is known for being effective in solving MAB problems. Figure 4 illustrates the optimism principle under uncertainty in UCB algorithms. In this example, despite having a lower expected reward, Arm 3 will be played over Arm 4, to try to gather more information about it.

In order to select the most promising arm at the t-th iteration (k_t) LinUCB selects the arm that maximizes the UCB using

$$k_t = \arg\max_k \left\{ x_{kt}^\top \hat{\theta}_t + \alpha \sqrt{x_{kt}^\top A_t^{-1} x_{kt}} \right\}, \tag{3}$$

where A_t is the matrix containing the information about whether the arm k has been played in the previous $t-1$ iterations and gets updated at each iteration following $A_{t+1} = A_t + x_{kt} x_{kt}^\top$.

In the mTWLP, there are up to $2^n - 1$ combinations of time periods. Because calculating the UCB of each set of time periods requires matrix multiplications, the enumeration of all the sets of time periods is untractable for large values of n. Hence, to get the most promising set of time periods, we propose an integer programming model that selects the set of time periods with the largest UCB. Note that Equation (3) has a non-linear term. Hence, we apply the square root linearization technique proposed in Asghari et al. (2022). It is important to highlight that, as the model is independent of the customer c and the iteration t,

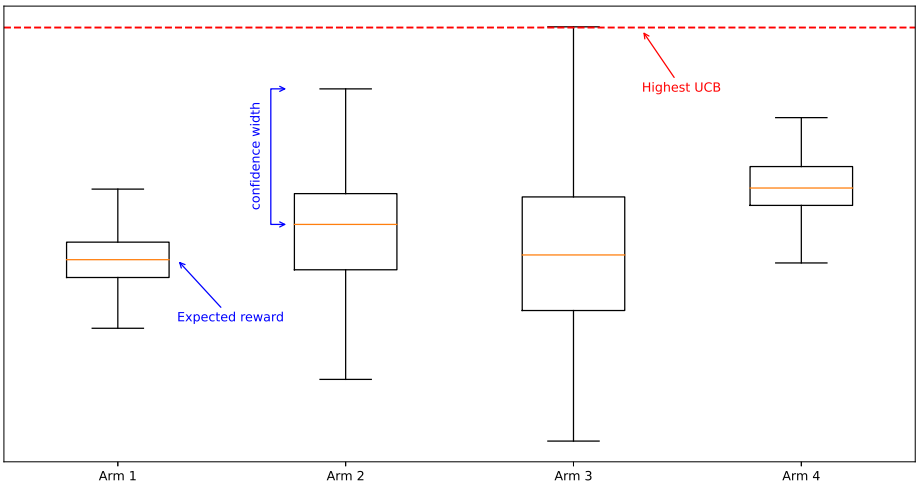


Figure 4: UCB algorithms illustration

with a slight abuse in notation, we will write x, $\hat{\theta}$, A, and $\mathcal{I}$, where $\mathcal{I}$ is a set with the indices of the time periods in which the customer has been visited in the past. The model that selects the most promising set of time periods follows:

$$\text{maximize} \sum_{i=1}^{n} x_i \hat{\theta}_i + \alpha \sqrt{B} \tag{4}$$

$$\text{subject to } B = \sum_{i=0}^{m-1} 2^i z_i + (u - 2^n + 1) z_m \tag{5}$$

$$\sum_{j=1}^{n} \sum_{i=1}^{n} (x_i A_{ij}) x_j \geqslant \sum_{i=0}^{m-1} 2^{2i} z_i + \sum_{i=0}^{m-2} \sum_{j>i}^{m-1} 2^{i+j+1} y_{ij} + (u - 2^m + 1) \sum_{i=0}^{m-1} 2^{i+1} y_{im} + (u - 2^n + 1)^2 z_m \tag{6}$$

$$y_{ij} \leqslant z_i \qquad \forall i, j \in \{0, \ldots, m\} \tag{7}$$

$$y_{ij} \leqslant z_j \qquad \forall i, j \in \{0, \ldots, m\} \tag{8}$$

$$y_{ij} \geqslant z_i + z_j - 1 \qquad \forall i, j \in \{0, \ldots, m\} \tag{9}$$

$$\sum_{i=1}^{n} x_i \geqslant 1 \tag{10}$$

$$x_i = 1 \qquad \forall i \in \mathcal{I} \tag{11}$$

$$x_i \in \{0, 1\} \qquad \forall i \in \{1, \ldots, n\} \tag{12}$$

$$z_i, y_{ij} \in \{0, 1\} \qquad \forall i, j \in \{0, \ldots, m\} \tag{13}$$

$$B \in \mathbb{N} \tag{14}$$

where y_{ij} and z_i are binary variables used to linearize the square root term, $u = \lceil \lambda_{max} n \rceil$ is an upper bound of the square root term ($\sqrt{x_{kt}^\top A_t^{-1} x_{kt}}$), and $m = \lfloor \log_2(u+1) \rfloor$. For more details about the upper bound

of $\sqrt{x_{kt}^{\top} A_t^{-1} x_{kt}}$, we refer the reader to the Appendix A. The objective function shown in Equation (4) corresponds to the UCB of each set of time periods. However, in order to perform the linearization, we replace the square root term with the integer variable B. Constraint (5) equalizes B to the development by binary variables of any natural number lower or equal than $u \in \mathbb{N}$ (i.e., $B \leqslant u$). The right-hand side (RHS) of Constraint (6) (that corresponds to the RHS of Constraint (5) squared after applying a linearization to approach the product of binary variables) is bounded by $x_{kt}^{\top} A_t^{-1} x_{kt}$. Thus, the integer variable $B \leqslant \sqrt{x_{kt}^{\top} A_t^{-1} x_{kt}}$. Nevertheless, as $B \in \mathbb{N}$, there might be an approximation error bounded in $[0, 1)$. Constraints (7), (8), and (9) are used to linearize the product of the binary variables z_i. Lastly, Constraint (10) assures the model to select a non-empty set of time periods, and Constraints (11) force the model to select the time periods that are known to be correct. For more details about the square root linearization, we refer the reader to Asghari et al. (2022).

This model will be used during the data collection or warm-up phase (i.e., first w iterations). Once this phase is finished, Constraint (15) is added to model (4)-(14). This constraint, together with Constraints (11) allows the bandit to explore at most only one new time period, apart from the time periods known to be correct.

$$\sum_{i=1}^{n} x_i \leqslant |\mathcal{I}| + 1 \tag{15}$$

The pseudo-code of the proposed LinUCB implementation is shown in Algorithm 2. In the first two lines, the algorithm initializes the matrices with the information of the explored time periods A and their rewards b. The algorithm's main loop, which completes the number of specified iterations, is located between lines 3 and 14. In line 4, the MAB collects the information about the expected reward estimator $\hat{\theta}_t$. Line 5 checks if the number of iterations devoted to the warm-up procedure has been reached. If not, the agent selects the set of time periods with the most promising UCB (including the time periods known to be correct) using the mathematical model presented above. Otherwise, Constraint (15) is added to the model, and the agent uses the extended model to select the best set of time periods. To finish the main loop, the algorithm receives a reward (line 10) for the chosen set of time periods and updates the matrices A and b (lines 11 and 12) with the new information.

Algorithm 2 LinUCB

Input: $\alpha > 0, \gamma > 0, w, T$

1: $A \leftarrow \gamma I_d$
2: $b \leftarrow 0_d$
3: **for** $t \in \{0, 1, \ldots, T-1\}$ **do**
4: $\quad \hat{\theta}_t \leftarrow A^{-1} b$
5: $\quad$ **if** $t < w$ **then**
6: $\quad\quad$ Choose the best arm k_t with model (4)-(14)
7: $\quad$ **else**
8: $\quad\quad$ Choose the best arm k_t that with the extended model (4)-(15)
9: $\quad$ **end if**
10: $\quad$ Compute reward $r_t = -\left| t_c^{soft} - t_c^{dri} \right|$
11: $\quad A \leftarrow A + x_{kt} x_{kt}^{\top}$
12: $\quad b \leftarrow b + x_{kt} r_t$
13: **end for**

4.2.1.2. LinTS. The Thompson Sampling algorithm with linear rewards (LinTS) was introduced in Agrawal and Goyal (2013) to solve the contextual bandit problem. LinTS belongs to the family of Thompson Sampling (TS) algorithms to solve the MAB problem, and this family is known for having the first algorithm proposed to approach the MAB problem in Thompson (1933). The idea behind TS is simple: assume that the expected reward of each arm μ_t follows a prior distribution and, at each iteration, the algorithm selects the arm according to the probability of being the best arm. A way of estimating μ_t is simply by sampling $\hat{\mu}_t$ from the posterior distribution. Hence, the exploration mechanism in TS algorithms is based on randomization: if there is almost no information about the arm k (i.e., the posterior distribution is not concentrated) there will be a large variance between samples, and the algorithm will probably explore until there is enough information about the arm k (i.e, the posterior distribution is enough concentrated around the true parameter). Hence, there will be less variance and the algorithm will explore less this arm. Figure 5 illustrates TS algorithms with an example: even though Arm 1 (depicted in blue) seems to have the best mean reward, as there is still uncertainty around Arm 2 (depicted in orange), the agent will select Arm 2 more often.

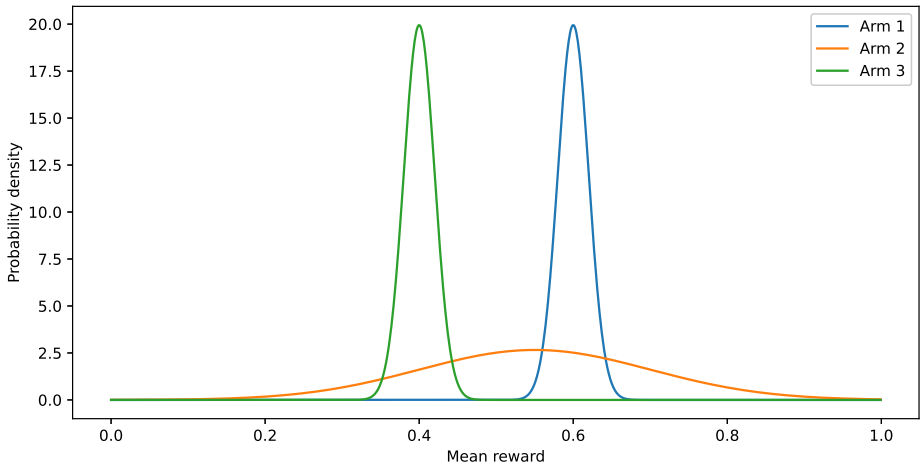


Figure 5: TS algorithms illustration

In Agrawal and Goyal (2013), the authors assume that the unknown true parameter (θ_t) of each arm follows a gaussian distribution $\mathcal{N}(\hat{\theta}_t, v^2 B^{-1})$, where $v = R\sqrt{9n \ln\left(\frac{T}{\epsilon}\right)}$, $R \geqslant 0$ being a bound of the difference between the expected and the real reward, and $\epsilon \in (0, 1]$ being the high probability bound of the regret. In terms of the mTWLP, at each iteration, LinTS will sample $\bar{\theta}_t$ for each time period. Then, unlike LinUCB, the selection of the most promising set of time periods avoids non-linear terms and this selection only consists of the first term of Equation (4). Hence, to choose the set of time periods to explore, the algorithm simply selects the time periods that have a positive value of $\bar{\theta}_t$ (in addition to the ones known to be correct). The algorithm follows this criterion during the data collection or warm-up phase (i.e., first w iterations), once the w first iterations have finished and the algorithm has enough information, the exploration mechanism will be limited to explore only the time slot with the highest positive $\bar{\theta}_t$, if there is any.

The pseudo-code of the proposed LinTS implementation is shown in Algorithm 3. In the first two lines, the algorithm initializes the matrices with the information of the explored time periods A and their rewards b. The algorithm's main loop, which completes the number of specified rounds, is located between lines 3 and 14. Line 4 collects the information about the expected reward estimator $\hat{\theta}_t$. In line 5, the algorithm samples the parameter $\bar{\theta}_t$ for all the time periods. Line 6 checks if the number of iterations devoted to the warm-up procedure has been reached. If not, line 7 selects all the time periods with a positive value of $\bar{\theta}_t$

and the time periods known to be correct. Otherwise, line 9 selects the time period with the largest positive value of $\bar{\theta}_t$ and the time periods known to be correct. To finish the main loop, the algorithm receives a reward (line 11) for the chosen set of time periods and updates the matrices A and b (lines 12 and 13) with the new information.

Algorithm 3 LinTS

Input: $R > 0, \epsilon \in (0, 1], T \in \mathbb{N}$

1: $A \leftarrow I_d$
2: $b \leftarrow 0_d$
3: **for** $t \in \{1, 2, \ldots, T\}$ **do**
4: $\quad \hat{\theta}_t \leftarrow A^{-1}b$
5: $\quad$ Sample $\bar{\theta}_t$ from $\mathcal{N}(\hat{\theta}_t, \nu^2 B^{-1})$
6: $\quad$ **if** $t < w$ **then**
7: $\quad\quad$ Choose the time periods that have a positive value of $\bar{\theta}_t$ and the correct time periods
8: $\quad$ **else**
9: $\quad\quad$ Choose the time period with the largest positive value of $\bar{\theta}_t$ and the correct time periods
10: $\quad$ **end if**
11: $\quad$ Compute reward $r_t = -\left|t_c^{soft} - t_c^{dri}\right|$
12: $\quad A \leftarrow A + x_{kt}x_{kt}^{\top}$
13: $\quad b \leftarrow b + x_{kt}r_t$
14: **end for**

4.2.2. Combinatorial MAB algorithms

A combinatorial MAB (CMAB) is considered a generalization of MAB, as the algorithm is allowed to play a combination of arms per iteration instead of only one. This combination of arms is called super arm and we denote it as $S \in \mathcal{S}$, where $\mathcal{S}$ is the set containing all the super arms. As a consequence, a CMAB can receive different rewards for each of the arms $k \in S_t$. In terms of the mTWLP, contrary to the Linear MAB, in the CMAB one arm $k \in K$ contains one time period in which customer c will be served and, as there are n arms, $\mathcal{S}$ may contain up to $2^n - 1$ super arms. In this study, one CMAB super arm is equivalent to one Linear MAB arm, as both are defined as combinations of time periods.

Another important aspect to highlight in CMAB algorithms is that usually the expected rewards are assumed to be bounded in $[0, 1]$. Hence, let $\mathcal{J}_t$ be the set of indexes of all $k \in S_t$ and $j_c \in \mathbb{N} \mid j_c \leqslant n$ be the index of the time period where the driver served customer c. Then we define the reward of each $k \in S_t$ at iteration t as

$$r_{kt} = 1 - \frac{|i_k - j_c|}{n}, \ \forall k \in S_t. \tag{16}$$

To illustrate this reward with an example, suppose that there are n different time periods and that the CMAB algorithm proposes to serve the customer in the first time period. If the driver keeps the customer in that position, the first time period receives a reward of $r_{1t} = 1 - \frac{|1-1|}{n} = 1$, while if the driver prefers to serve the customer in the n-th time period, the first time period receives as reward $r_{1t} = 1 - \frac{|1-n|}{n} = \frac{1}{n}$. Hence, if the driver decides to keep a customer, it makes sense to give the best reward to the selected time period. On the other hand, if the driver prefers to serve the customer in a different time period, that means that either this time period is incorrect or not efficient and it should be penalized.

In this study, these CMABs face the same challenges as the Linear MABs (i.e., the cold start and high cost when exploring the wrong time period). To approach these issues, we set a warm-up parameter

$w < T$, $w \in \mathbb{N}$. During the data collection or warm-up phase, as the expected rewards are assumed to be bounded in $[0, 1]$, to prevent the algorithms to select all the time periods at each iteration, the maximum time period selection to form S is limited to 10 (including the correct time periods). After the warm-up iterations, the algorithms are allowed to explore only one new time period at each iteration.

We decided to use CUCB (Chen et al., 2013) and CTS (Wang and Chen, 2018) as CMAB algorithms. It can be noted that CUCB belongs to the UCB family of MAB algorithms (as LinUCB) and CTS to the TS family of MAB algorithms (as LinTS), so they share the mechanism to get the exploitation-exploration trade-off.

4.2.2.1. CUCB. Similarly to the UCB MAB algorithm, CUCB calculates the UCB value of every arm $k \in K$ at each iteration t. To do so, Chen et al. (2013) proposed the following formula:

$$\bar{\mu}_k = \hat{\mu}_k + \sqrt{\frac{3 \ln t}{2\mathcal{T}_k}} \; \forall k \in K, \tag{17}$$

where $\hat{\mu}_k$ represents the reward average at iteration t, $\bar{\mu}_k$ is the UCB of arm k, and $\mathcal{T}_k$ is the number of times that arm k has been played. In terms of the mTWLP, in a similar way to LinUCB, the super arm selection is done by calculating the UCB of each arm. However, as the second term in Equation (17) only involves information about the current iteration and the number of times that a time period has been explored, the algorithm does not need to use the model (4)-(14).

The pseudo-code of our CUCB implementation is shown in Algorithm 4. The algorithm's main loop, which completes the number of specified rounds, is located in lines 1 to 11. In line 2, the algorithm calculates the UCB value of all the time periods. Line 3 checks if the number of iterations devoted to the warm-up procedure has been reached. If not, line 4 selects the time periods with the 10 largest value of $\bar{\mu}_k$, including the time periods known to be correct. Otherwise, line 6 selects the time period with the largest positive value of $\bar{\mu}_k$ and the time periods known to be correct. To finish the main loop, the algorithm receives a reward (line 8) for each time period in S_t and updates the reward averages $\hat{\mu}_k$ and the number of times that each time period has been explored $\mathcal{T}_k$ (lines 9 and 10).

Algorithm 4 CUCB

Input: $T \in \mathbb{N}$

1: **for** $t \in \{1, 2, \ldots, T\}$ **do**
2: Calculate UCB $\bar{\mu}_k \leftarrow \hat{\mu}_k + \sqrt{\frac{3 \ln t}{2\mathcal{T}_k}}$ for all K time periods
3: **if** $t < w$ **then**
4: Form S_t adding the time periods with the 10 largest value of $\bar{\mu}_k$ and the correct time periods
5: **else**
6: Form S_t adding the time period with the largest value of $\bar{\mu}_k$ and the correct time periods
7: **end if**
8: Compute rewards $r_{kt} = 1 - \frac{|i_k - j_c|}{n}, \; \forall k \in S_t$
9: $\hat{\mu}_k \leftarrow \frac{\hat{\mu}_k \mathcal{T} + r_{kt}}{\mathcal{T}+1}, \; \forall k \in S_t$
10: $\mathcal{T}_k \leftarrow \mathcal{T}_k + 1, \; \forall k \in S_t$
11: **end for**

4.2.2.2. CTS. As a TS algorithm, CTS assumes that the expected reward of each arm k follows a probability distribution and, at each iteration, the algorithm selects as a super arm the arms with the largest

probability of getting a reward. The authors in Wang and Chen (2018) assume that μ_k follows a beta distribution Beta(α_k, β_k). Once the rewards for each arm have been received (Equation (16)), α_k and β_k are uploaded following the formulas $\alpha_k = \alpha_k + r_k t$ and $\beta_k = \beta_k t + 1 - r_t$ to compute the posterior distribution. In terms of the mTWLP, in the same way as the LinTS algorithm, the super arm selection is done by sampling $\bar{\mu}_k$ for each time period.

The pseudo-code of our CTS implementation is shown in Algorithm 5. In the first two lines, the algorithm initializes the Beta distribution parameters. The algorithm's main loop, which completes the number of specified rounds, is located between lines 3 and 13. In line 4, the algorithm samples the parameter $\bar{\mu}_k$ for each time period $k \in K$. Line 5 checks if the number of iterations devoted to the warm-up procedure has been reached. If not, line 6 selects the time periods with the 10 largest value of $\bar{\mu}_k$, including the time periods known to be correct. Otherwise, line 8 selects the time period with the largest positive value of $\bar{\mu}_k$ and the time periods known to be correct. To finish the main loop, the algorithm receives a reward (line 10) for each time period in S_t and updates the parameters α_k and β_k (lines 11 and 12) with the new rewards.

Algorithm 5 CTS

Input: $T \in \mathbb{N}$

1: $\alpha_k \leftarrow 1$ for all K arms
2: $\beta_k \leftarrow 1$ for all K arms
3: **for** $t \in \{1, 2, \ldots, T\}$ **do**
4: Sample $\bar{\mu}_k$ from Beta(α_k, β_k) for all K arms
5: **if** $t < w$ **then**
6: Form S_t adding the time periods with the 10 largest value of $\bar{\mu}_k$ and the correct time periods
7: **else**
8: Form S_t adding the time period with the largest value of $\bar{\mu}_k$ and the correct time periods
9: **end if**
10: Compute rewards $r_{kt} = 1 - \frac{|i_k - j_c|}{n}, \ \forall k \in S_t$
11: $\alpha_k \leftarrow \alpha_k + r_k t$
12: $\beta_k \leftarrow \beta_k t + 1 - r_t$
13: **end for**

5. Computational experiments

In this section, we assess the capacity of our approaches to learn customers' real sets of time windows and provide more realistic routes to limit the costs for the company. We first describe the framework that allows testing our approaches (i.e., simulation algorithms, instances, and parameters). Then we compare the learning mechanisms presented in Section 4 (i.e., RH, LinUCB, LinTS, CUCB, and CTS). We tried to compare the learning mechanisms with a non-learning algorithm that solves a minimum-tour duration TSP with a fixed and non-evolving set of customer time windows (i.e., it does not learn after observing the driver's route). However, the routes proposed by this non-learning approach were so different from the ones using time windows that the simulated algorithms that we present below were not able to recover feasibility. Thus, having a learning mechanism helps to provide more realistic routes in the considered instances. Lastly, all the computational experiments have been conducted in a processor AMD Rome 7532 @ 2.40 GHz with 10 GB of RAM.

5.1. Simulation algorithms: generating the software and the driver's routes

As we defined in Section 3, at the beginning of day t the company considers the set of customers to serve C_t and, with the current knowledge of the customers' set of time windows designs the *software route* $\hat{z} = f(C_t, \hat{y}_{C_t})$. However, this route may be infeasible for the driver. Hence, the driver modifies the route (if necessary) and creates the *driver's route* $\bar{z} = g(C_t, y_{C_t}, \hat{z})$. In the remainder of this section, we describe the procedures used to simulate the construction of the *software route* and the procedure applied by the driver to repair the route and generate the *driver's route*.

To build the *software route*, we modify the integer linear programming model for the symmetric TSP-TW proposed by Kara and Derya (2015). This model is an adaptation of the non-linear TSP-TW model proposed by Baker (1983) and minimizes the tour duration. The way we adapt this model is by allowing each customer to have more than one time window (i.e., adapting it to solve TSP-mTW instances).

Let $t_c \in \mathbb{N} \cup \{0\}$ be the arrival time at a certain customer $c \in C$, with t_0 being the departure from the depot and t_{n+1} the arrival time at the depot. Let $p_{c\hat{c}} \in \{0,1\}$ be a non-consecutive precedence variable between the customers c and $\hat{c}$. The variable $y_{c\hat{c}}$ takes value 1 if the customer c is served before customer $\hat{c}$, and 0 otherwise. Let $\mathcal{L}_c$ be the time window set of indexes of the customer c, $z_{cl} \in \{0,1\}$ be a binary variable that gets value 1 if the time window with index l is used for the customer c and 0 otherwise, a_{cl} and b_{cl} be the earliest and latest that a customer c can be served in the time window with index l, and $M > 0$ being a sufficiently large constant. The model for building the *software route* follows:

$$\text{minimize } t_{n+1} - t_0 \tag{18}$$

$$\text{subject to } t_c - t_0 \geqslant d_{0c} \qquad \forall c \in C \tag{19}$$

$$t_{n+1} - t_c \geqslant d_{c,n+1} \qquad \forall c \in C \tag{20}$$

$$t_c \geqslant \sum_{l \in \mathcal{L}_c} a_{cl} z_{cl} \qquad \forall c \in C \tag{21}$$

$$t_c \leqslant \sum_{l \in \mathcal{L}_c} b_{cl} z_{cl} \qquad \forall c \in C \tag{22}$$

$$\sum_{l \in \mathcal{L}_c} z_{cl} = 1 \qquad \forall c \in C \tag{23}$$

$$t_c - t_{\hat{c}} + M p_{c\hat{c}} \geqslant d_{c\hat{c}} \qquad \forall c, \hat{c} \in C \cup \{0\} \tag{24}$$

$$t_{\hat{c}} - t_c - M p_{c\hat{c}} \geqslant d_{c\hat{c}} - M \qquad \forall c, \hat{c} \in C \cup \{0\} \tag{25}$$

$$t_c \geqslant 0 \qquad \forall c \in C \cup \{0, n+1\} \tag{26}$$

$$z_{cl} \in \{0,1\} \qquad \forall l \in \mathcal{L}_c, \forall c \in C \cup \{0, n+1\} \tag{27}$$

$$p_{c\hat{c}} \in \{0,1\} \qquad \forall c, \hat{c} \in C \cup \{0\} \tag{28}$$

The objective function shown in Equation (18) minimizes the tour duration. Constraints (19) and (20) ensure that the route starts and finishes at the depot. Constraints (21) and (22) bound the delivery time of customer c inside of the selected time window. Constraints (23) assure that only one time window per customer is used. Constraints (24) and (25) correspond to the linearization made by Kara and Derya (2015) and ensure that the delivery time difference between customers c and $\hat{c}$ is greater than the time between the two customers $d_{c\hat{c}}$.

In order to correct the *software route*, the *simulated driver* (SD) does the following: first, it checks if the route is feasible by verifying if the customers are served inside of their set of time windows. If the service time at a customer is unfeasible, SD removes this customer from the route and tries to reinsert it. This is

done with a reinsertion heuristic. Lastly, after checking for infeasibilities, SD tries to improve the route with a local search algorithm. This process is kept simple to simulate the decisions that a real driver would make to improve a route in a short time at the beginning of the day. In the following, we summarize the procedures for the aforementioned insertion and local search algorithms. However, for further details about the algorithms, we refer the reader to the original papers.

Let $\bar{C} \subset C_t$ be the set of customers to reinsert. In terms of the insertion algorithm, SD tries to reinsert every customer $c \in \bar{C}$ following a simplification of the reinsertion algorithm for the TSP-TW presented in Gendreau et al. (1998). The first step in our adaptation is to sort the customers $c \in \bar{C}$ in ascending order of the sum of the width of their time windows. Then, SD tries to insert every customer $c \in \bar{C}$ in every possible position of the route trying all of its time windows. Lastly, SD inserts c in the feasible position that generates the lowest increase in the duration of the tour (breaking ties using the lowest increase in total travel time). By feasible we mean that the insertion of the customer c in the partial route should not push other customers out of their time window. In the original paper, the authors allow a backtracking technique that removes customers from the route to increase feasibility. To better align with what real drivers would do, we do not allow backtracking, and we do not allow the customers already in the route to move between time windows.

To improve the solution found by the insertion algorithm, we use a simple local search based on the TSP-TW 2-exchange algorithm presented in Savelsbergh (1992). The 2-exchange is an algorithm that at each step considers two different edges from a route $v_{c\hat{c}}$ and $v_{\bar{c}\bar{c}}$, and exchanges them with $v_{c\bar{c}}$ and $v_{\hat{c}\bar{c}}$, reversing the orientation of all the other edges between the customers $\hat{c}$ and $\bar{c}$. Again, to better align with what real drivers would do, we adapt this local search to allow the 2-exchanges that involve reversing only one edge. Hence, following the example shown above, SD swaps the edges $v_{c\hat{c}}$ and $v_{\bar{c}\bar{c}}$ with $v_{c\bar{c}}$ and $v_{\hat{c}\bar{c}}$ only if in the original route those edges are linked by the edge $v_{\hat{c}\bar{c}}$. Lastly, as in the insertion algorithm, we do not allow the customers to move between time windows.

5.2. Test instances

To carry out our experiments, we used a dataset coming from the city of Montreal, Canada. This dataset (available through the Open Canada website) contains the location and the information about all the parking signs in the great area of Montreal. Under the assumption that a customer lives in front of the parking sign and that the driver has to follow the parking restriction to perform the delivery, this dataset can be used to create mTWLP instances.

To create the instances, we have restricted the dataset to the parking signs that: 1) the resulting parking time window intersected with $[9am, 5pm]$ is not void; 2) the parking signs affect during weekdays (i.e., Monday to Sunday, Monday to Friday, etc); 3) the parking signs are located in the neighborhoods of Le Plateau - Mont-Royal, Outremont, Rosemont - La Petite Patrie, or Ville-Marie. This results in 4,373 parking signs of 209 different types.

To create each instance we sample 100 parking signs from the restricted dataset. As all the resulting parking time windows always start and end either at o'clock or at half past, we divided the time horizon $[9am, 5pm]$ into 16 periods of 30 minutes each. In order to calculate the travel time matrix, the haversine formula (de Mendoza y Rios, 1795) has been used. We assume a uniform service time of 3 minutes, and the average speed of the vehicle is assumed to be 8.8 m/s (Saunier and Chabin, 2020). Lastly, the depot has been located in the industrial area of Saint Laurent.

Each run of the algorithms simulates a total of $T = 450$ days and, on each day t, the driver has to serve $|C_t| = 30$ randomly selected customers. In order to simulate fairly all the algorithms, the set of customers to serve on any of the 450 days (C_t) is the same for all methods. In addition, the warm-up parameter w equals to 50 iterations for all the methods.

5.3. Results

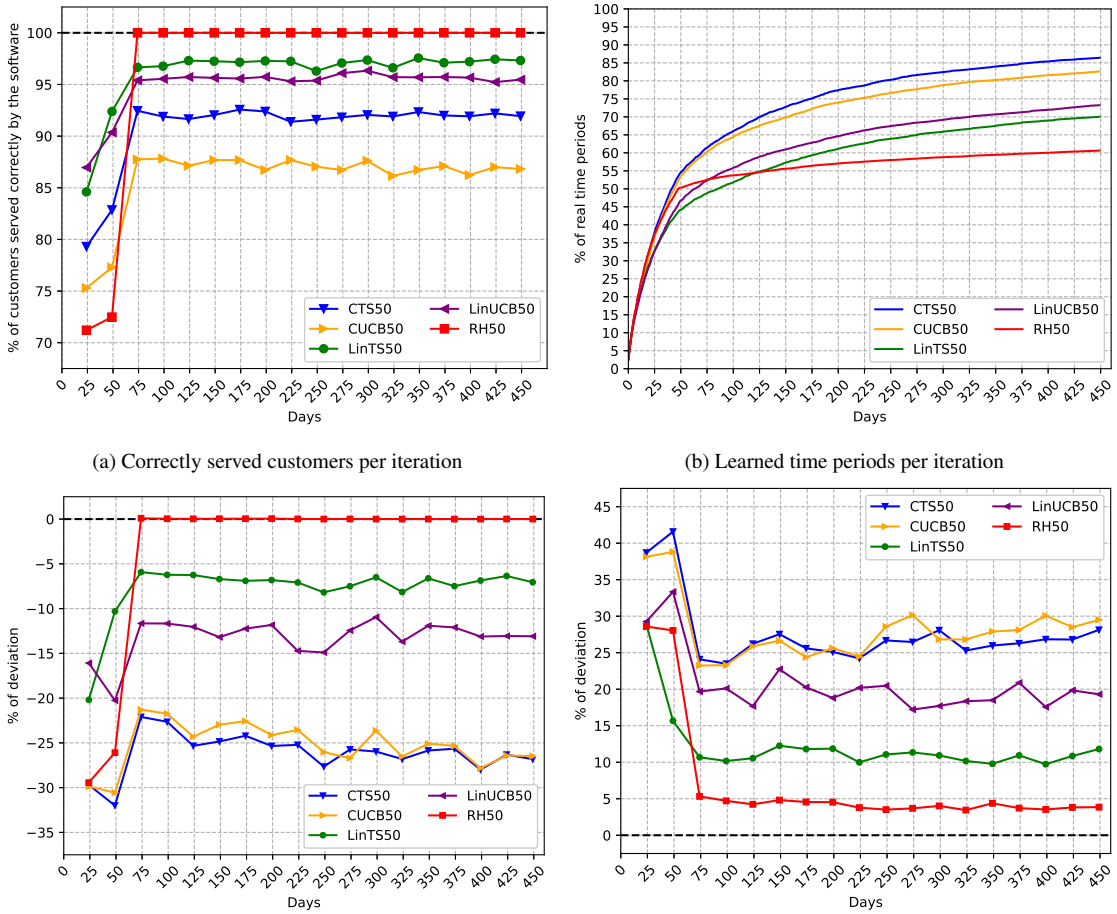
In this section, we discuss two experiments. The first aims to assess the value of the learning mechanisms. In other words, it tests whether the learning mechanisms are capable of acquiring a higher number of correct time slots and the impact that this learning has on the routing costs, and if the proposed routes are more likely to be used by the drivers without any intervention. The second aims to analyze the quality of the learned time windows. To do so, we let the MAB algorithms run normally for 100 iterations (50 data collection iterations and 50 limited exploration iterations) and then we turn off the exploration. By turning off the exploration we will be able to see if the learned time periods help the software to propose good routes. This experiment could represent a company that at some point wants to stop exploring and exploit the information that have acquired. For the remainder of the section, we denote all the algorithms by concatenating their name with the number of warm-up iterations (i.e., CTS50, CUCB50, LinTS50, LinUCB50, and RH50).

Figure 6 details the results of the first experiment. In Figure 6a we show the percentage of customers that the *software route* served inside of the real set of time windows (i.e., the higher the percentage, the fewer the number of customers that must be reaccommodated by the driver). In this chart, to get some smoothness, each point over the x-axis represents an average over the previous 25 days and over the whole set of instances, while the y-axis represents the percentage of customers served at a feasible time over the 30 daily customers. All the algorithms serve correctly above 85% of the customers after the 50 warm-up iterations. However, it can be seen that the combinatorial MABs are proposing more wrong time windows, as they might be exploring more new time slots than the linear MABs. Figure 6b compares the percentage of correct time periods that each algorithm learns over 450 days. In the chart, we can see that all the algorithms learn at a fast rate during the warm-up period. After that, as the exploration is restricted for the MAB algorithms and RH50 only learns what it observes from the drivers. Thus, the learning pace decreases for the remaining 400 days. As expected, the MAB algorithms learn more correct time periods than RH50 over the 450 iterations. Analyzing the figures 6a and 6b together we can conclude that the algorithms that explore more are able to learn more time slots, but also propose more incorrect time windows.

Figure 6c hecks the convergence of the *software route* tour duration towards the *driver's route* tour duration. Again in this chart, each point over the x-axis represents an average over the previous 25 days and over the entire set of instances, while the y-axis represents the deviation between *software route* and *driver's route* tour duration (i.e., $(dur_{soft} - dur_{dri})/dur_{dri}$, where dur corresponds with the tour duration of the route). We can see that the algorithms provided with learning mechanisms improve after the warm-up period. However, due to the higher exploration of the combinatorial MABs, the proposed routes have irrealistic costs while the linear MABs are able to propose routes that are closer to the driver route. Taking into account both figures 6a and 6c, we can conclude that the MAB algorithms keep exploring incorrect time windows (MAB algorithms never reach 100% in Figure 6a) and propose less realistic routes, while RH50 proposes feasible routes that the drivers rarely have to change. Thus, RH50 offers more realistic routes than CTS50, CUCB50, LinTS50, and LinUCB50, with the combinatorial MABs being the learning algorithms with the worst tour deviation.

Similarly, Figure 6d compares the *driver's route* tour duration with an oracle that knows the real set of time windows and calculates a near-optimal solution (i.e., $(dur_{soft} - dur_{ora})/dur_{ora}$). As in Figure 6c, the drivers have difficulties repairing routes designed by the combinatorial MABs, yielding low-performance routes. With respect to the linear MABs, the costs of routes seem to stop improving after the warm-up iterations, incurring overruns in comparison with the oracle. RH50 reaches the lowest tour duration deviation for the drivers. However, taking into account figures 6c and 6d, as RH50 reaches 0% of deviation in the first figure and is approaching the 5% of deviation in the latter figure, this means that the company

incurs almost no overruns. In conclusion, after observing Figure 6, we can state that having an exploration mechanism allows to learn a higher number of correct time slots (over 25% for CTS50). However, as that implies proposing more incorrect time slots, a higher learning rate does not translate into a significant value to the company, as the driver is creating more expensive routes after correcting the infeasibilities.



(a) Correctly served customers per iteration

(b) Learned time periods per iteration

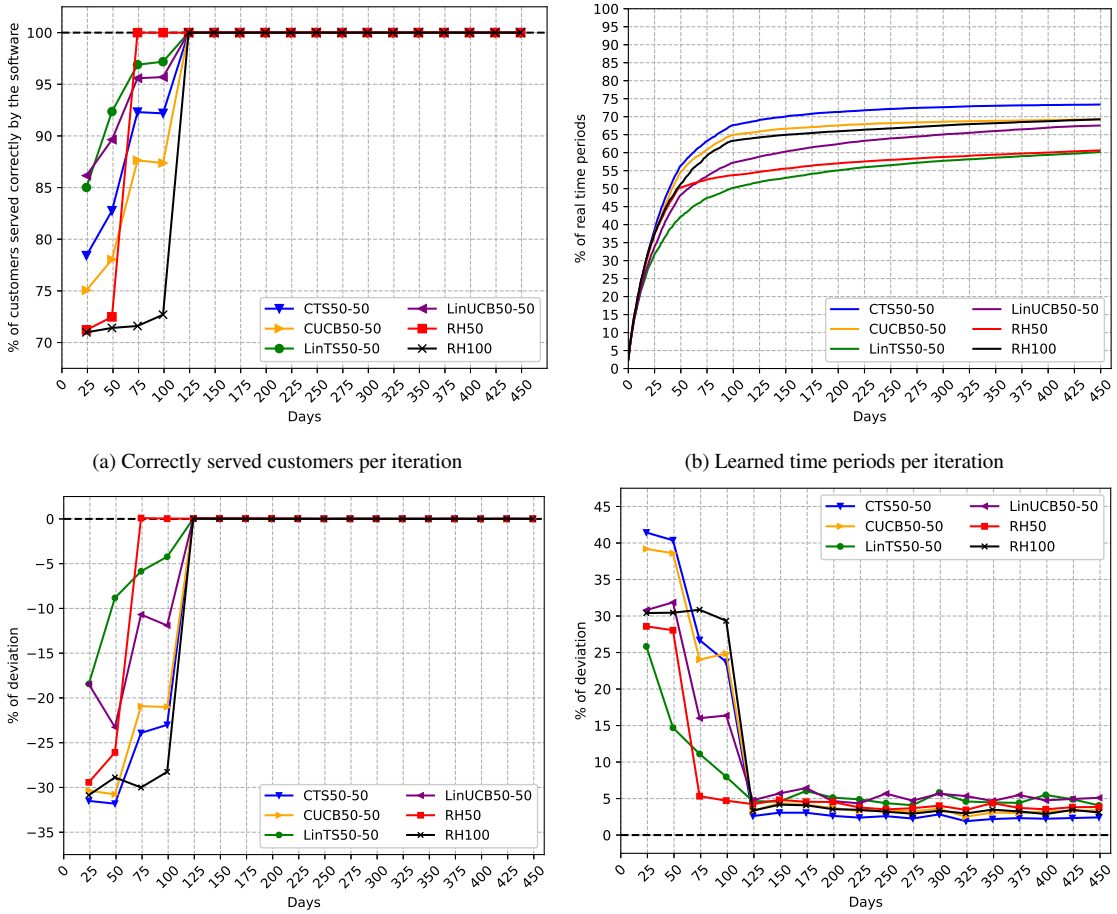
(c) Tour duration deviation between software and driver's route

(d) Tour duration deviation between driver and oracle's route

Figure 6: Learning mechanisms comparison with 50 warm-up iterations

As it is not clear whether having an exploration algorithm adds significant value, we present an additional experiment. First, we let all the algorithms warm up for 50 iterations. After that, we restrict the exploration for another 50 iterations, as defined in Section 4. Lastly, we turn off the exploration mechanisms for the remaining 350 days. Once the exploration mechanisms are turned off, the MAB algorithms propose as a set of time windows, the time periods known to be correct (i.e., the same behavior as RH). Hence, with this experiment, we compare if allowing the MAB algorithms to explore for a short period of time compensates for the costs in the long run in comparison with having no exploration, as RH. We denote the algorithms with 50 warm-up iterations and 50 exploration iterations by concatenating their name with "50-50" (i.e., CTS50-50, CUCB50-50, LinTS50-50, LinUCB50-50). We also add to the comparison RH50 and RH100, respectively RH with 50 and 100 warm-up iterations. The results of the second experiment are shown in Figure 7.

Figure 7a shows the percentage of customers that the *software route* served inside of the correct set



(a) Correctly served customers per iteration

(b) Learned time periods per iteration

(c) Tour duration deviation between software and driver's route

(d) Tour duration deviation between driver and oracle's route

Figure 7: Learning mechanisms comparison with 50 warm-up days and turning off the exploration after other 50 days

of time windows. As expected, during the first 100 iterations, all the MAB algorithms behave similarly to the equivalent ones in the first experiment, and, after turning off the exploration mechanisms, the MAB algorithms always propose feasible routes. In addition to that, RH100 has a similar behavior to RH50 in the first 50 iterations and remains almost constant until the 100 warm-up iterations are over. Figure 7b compares the percentage of correct time periods that each algorithm learns over 450 days. Again, we see that all the MAB algorithms but LinUCB50-50 still learn more time periods than RH50. Additionally, due to the larger warm-up periods, RH100 is able to learn more than the linear MABs, but not more than the combinatorial MABs. Figure 7c shows results on the same line: after the first 100 days, all the learning algorithms propose realistic routes that the driver barely has to change. Note that the MAB algorithms seem to be bounded between RH50 and RH100 during the second 50 days. Lastly, Figure 7d shows that, after turning off the exploration, the combinatorial MABs and RH100 are capable of generating slightly less expensive routes than RH50 but incurring higher costs in the days 50 to 100. Analyzing together figures 7b and 7d, we can conclude that the highest learning pace of combinatorial MAbs with respect to RH50 and RH100 yields less expensive routes once the exploration mechanism is turned off. That being said, the routes proposed by the combinatorial MABs during the first 100 iterations are less realistic than the ones proposed by the RH.

Therefore, it seems reasonable to conclude that adding learning mechanisms help to provide more realistic routes that the drivers accept and change less often (figures 6c and 7c) without incurring high costs for the company (figures 6d and 7d). Additionally, having an exploration mechanism for a short period of time leads to acquiring more knowledge from the driver yielding more realistic and efficient routes. However, this comes with the price of less realistic routes during the iterations where the MABs are allowed to explore.

6. Conclusions

Motivated by the relevant real-life problem of understanding the behavior of the drivers in the last-mile delivery, we presented the mTWLP, a problem that has not been studied in the literature before. The mTWLP defines the drivers' knowledge in terms of time windows that have to be learned. We addressed this problem in two different ways. First, we proposed RH, a new algorithm that learns the customers' real sets of time windows by mimicking drivers' historical data. Second, we adapted four MAB algorithms to add an exploration mechanism to RH. Furthermore, those two approaches are able to start learning without the need for historical data. Additionally, we also developed an integer programming model that selects the most promising arm without incurring the enumeration of all of them, which can be untractable if the set of arms is large.

The computational experiments showed that the implementation of a learning mechanism results in more effective and realistic routes that are more likely to be followed by the drivers without significantly raising the companies' routing costs. Furthermore, it was shown that RH solves the mTWLP, and the addition of exploration mechanisms leads, as expected, to learn a larger number of time periods. If the exploration of the algorithms is turned off after a short period of iterations, as a company might decide to do, the higher information acquired by the MABs yields lower costs than RH. However, this short exploration phase comes with higher costs. Therefore, a company would search for a trade-off between high costs in the short term (i.e., exploration phase) and lower costs in the long term (i.e., exploitation phase).

Future studies could explore whether a harder set of instances could better support the use of exploration. Furthermore, other types of learning procedures could be used to acquire time windows (e.g., inverse optimization, machine learning, or data-driven optimization). Lastly, it would be interesting for future studies to look at the dynamic version of the mTWLP, in which the sets of time windows change over time. It is reasonable to believe that some of these time windows change over time in practice (e.g., in summer versus winter). This dynamic version of the mTWLP should also better highlight the additional value of exploration since it would then be hard for RH to unlearn the previously correct but now incorrect time windows; which is done automatically with MABs due to exploration.

References

Agrawal, S., Avadhanula, V., Goyal, V., Zeevi, A., 2019. MNL-Bandit: A Dynamic Learning Approach to Assortment Selection. Operations Research 67, 1453–1485.

Agrawal, S., Goyal, N., 2013. Thompson sampling for contextual bandits with linear payoffs, in: Proceedings of the 30th International Conference on Machine Learning, pp. 127–135.

Arora, S., Doshi, P., 2021. A survey of inverse reinforcement learning: Challenges, methods and progress. Artificial Intelligence 297, 103500.

Asghari, M., Fathollahi-Fard, A.M., e hashem, S.M.J.M.A., Dulebenets, M.A., 2022. Transformation and Linearization Techniques in Optimization: A State-of-the-Art Survey. Mathematics 10.

Aswani, A., Shen, Z.J.M., Siddiq, A., 2018. Inverse optimization with noisy data. Operations Research 66, 870–892.

Badanidiyuru, A., Kleinberg, R., Slivkins, A., 2018. Bandits with Knapsacks. Journal of the Association for Computing Machinery 65, 1–55.

Baker, E.K., 1983. An Exact Algorithm for the Time-Constrained Traveling Salesman Problem. Operations Research 31, 938–945.
Bodur, M., Chan, T.C.Y., Zhu, I.Y., 2022. Inverse Mixed Integer Optimization: Polyhedral Insights and Trust Region Methods. INFORMS Journal on Computing 34, 1471–1488.
Canoy, R., Guns, T., 2019. Vehicle routing by learning from historical solutions, in: Principles and Practice of Constraint Programming, pp. 54–70.
Chan, T.C., Kaw, N., 2020. Inverse optimization for the recovery of constraint parameters. European Journal of Operational Research 282, 415–427.
Chan, T.C., Mahmood, R., Zhu, I.Y., 2021. Inverse Optimization: Theory and Applications. arXiv:2109.03920 [math.OC] .
Chen, L., Chen, Y., Langevin, A., 2021. An inverse optimization approach for a capacitated vehicle routing problem. European Journal of Operational Research 295, 1087–1098.
Chen, W., Wang, Y., Yuan, Y., 2013. Combinatorial multi-armed bandit: General framework and applications, in: Proceedings of the 30th International Conference on Machine Learning, pp. 151–159.
Chow, J.Y., Recker, W.W., 2012. Inverse optimization with endogenous arrival time constraints to calibrate the household activity pattern problem. Transportation Research Part B: Methodological 46, 463–479.
Dieter, P., Caron, M., Schryen, G., 2023. Integrating driver behavior into last-mile delivery routing: Combining machine learning and optimization in a hybrid decision support framework. European Journal of Operational Research .
Dumanska, I., Hrytsyna, L., Kharun, O., Matviiets, O., 2021. E-commerce and m-commerce as global trends of international trade caused by the covid-19 pandemic. WSEAS Transactions on Environment and Development .
Fu, J., Luo, K., Levine, S., 2018. Learning robust rewards with adverserial inverse reinforcement learning, in: International Conference on Learning Representations.
Gendreau, M., Hertz, A., Laporte, G., Stan, M., 1998. A Generalized Insertion Heuristic for the Traveling Salesman Problem with Time Windows. Operations Research 46, 330–335.
Ghate, A., 2020. Inverse optimization in semi-infinite linear programs. Operations Research Letters 48, 278–285.
den Hertog, D., Postek, K., 2016. Bridging the gap between predictive and prescriptive analytics – new optimization methodology needed. Eprints for the optimization community .
Hewitt, M., Frejinger, E., 2020. Data-driven optimization model customization. European Journal of Operational Research 287, 438–451.
Horn, R., Johnson, C., 1985. Matrix Analysis. Cambridge University Press.
Jaynes, E.T., 1957. Information theory and statistical mechanics. Physical Review 106, 620–630.
Kara, I., Derya, T., 2015. Formulations for minimizing tour duration of the traveling salesman problem with time windows, in: 4th World Conference on Business, Economics and Management (WCBEM-2015), pp. 1026–1034.
Kubat, M., 2017. An Introduction to Machine Learning. Springer Cham.
Lai, T.L., 1987. Adaptive Treatment Allocation and the Multi-Armed Bandit Problem. The Annals of Statistics 15, 1091–1114.
Lattimore, T., Szepesvári, C., 2020. Bandit Algorithms. Cambridge University Press.
Li, L., Chu, W., Langford, J., Schapire, R.E., 2010. A contextual-bandit approach to personalized news article recommendation, in: WWW '10: Proceedings of the 19th International Conference on World Wide Web, pp. 661–670.
Lombardi, M., Milano, M., Bartolini, A., 2017. Empirical decision model learning. Artificial Intelligence 244, 343–367.
Mandi, J., Canoy, R., Bucarey, V., Guns, T., 2021. Data driven vrp: A neural network model to learn hidden preferences for vrp, in: 27th International Conference on Principles and Practice of Constraint Programming (CP 2021), pp. 42:1–42:17.
Recker, W., 1995. The household activity pattern problem: General formulation and solution. Transportation Research Part B: Methodological 29, 61–77.
de Mendoza y Rios, J., 1795. Memoria sobre algunos metodos nuevos para calcular la longitud por las distancias lunares, y aplicacion de su teorica a la solucion de otros problemas de navegacion .
Samson, B.P.V., Sumi, Y., 2019. Exploring factors that influence connected drivers to (not) use or follow recommended optimal routes, in: Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems, p. 1–14.
Saunier, N., Chabin, V., 2020. Should i bike or should i drive? comparative analysis of travel speeds in montreal. Findings , 11900.
Savelsbergh, M.W.P., 1992. The Vehicle Routing Problem with Time Windows: Minimizing Route Duration. ORSA Journal on Computing 4, 146–154.
Shahmoradi, Z., Lee, T., 2021. Quantile Inverse Optimization: Improving Stability in Inverse Linear Programming. Operations Research 70, 2538–2562.
Snoswell, A.J., N., S.P.S., Ye, N., 2020. Revisiting maximum entropy inverse reinforcement learning: New perspectives and algorithms, in: 2020 IEEE Symposium Series on Computational Intelligence (SSCI), pp. 241–249.
Thompson, W.R., 1933. On the Likelihood that One Unknown Probability Exceeds Another in View of the Evidence of Two Samples. Biometrika 25, 285–294.
Villar, S.S., Bowden, J., Wason, J., 2015. Multi-armed Bandit Models for the Optimal Design of Clinical Trials: Benefits and Challenges. Statistical science : a review journal of the Institute of Mathematical Statistics 30, 199–215.

Wang, S., Chen, W., 2018. Thompson sampling for combinatorial semi-bandits, in: Proceedings of the 35th International Conference on Machine Learning, pp. 5114–5122.

Winkenbach, M., Parks, S., Noszek, J., 2021. Technical Proceedings of the Amazon Last Mile Routing Research Challenge. MIT Megacity Logistics Lab.

Wu, C., Song, Y., March, V., Duthie, E., 2022. Learning from Drivers to Tackle the Amazon Last Mile Routing Research Challenge. arXiv:2205.04001 [cs.AI] .

You, S.I., Chow, J.Y., Ritchie, S.G., 2016. Inverse vehicle routing for activity-based urban freight forecast modeling and city logistics. Transportmetrica A: Transport Science 12, 650–673.

Ziebart, B.D., Maas, A., Bagnell, J., Dey, A.K., 2008. Maximum entropy inverse reinforcement learning, in: 22nd AAAI Conference on Artificial Intelligence, p. 1433–1438.

Appendix A. Upper bound of $\sqrt{x_{kt}^\top A_t^{-1} x_{kt}}$

In order to get u (i.e., an upper bound of $\sqrt{x_{kt}^\top A_t^{-1} x_{kt}}$), first we have to prove that $A_t \in \mathbb{R}^{n\times n}$ is a symmetric and positive definite matrix. Let $C \in \mathbb{R}^{n\times n}$, we denote that C is positive definite by $C \succ 0$, and that C is positive semidefinite by $C \succeq 0$. Then, as $A_0 = \gamma I_n$, where $\gamma > 0$ and I_n is the identity matrix of size n, $A_0 \succ 0$ and symmetric. In addition to that, as $\forall \hat{x} \in \mathbb{R}^n$ we have that $\hat{x}\hat{x}^\top \succeq 0$ and symmetric and $x_k \in \{0,1\}^n$, it follows that $x_{kt}x_{kt}^\top \succeq 0$ and symmetric $\forall t < T$. Consequently, as $A_{t+1} = A_t + x_{kt}x_{kt}^\top$, A_{t+1} $\forall t < T$, $A_0 \succ 0$ and symmetric and $x_{kt}x_{kt}^\top \succeq 0$ $\forall t < T$ and symmetric, we have that $A_{t+1} \succ 0$ and symmetric $\forall t \in T$. By last, as $A_{t+1} \succ 0$ and symmetric, it follows that $A_{t+1}^{-1} \succ 0$ and symmetric. Hence, to get the upper bound of $\sqrt{x_{kt}^\top A_t^{-1} x_{kt}}$, as $A_{t+1}^{-1} \succ 0$ and symmetric, and $x \neq \mathbf{0}^n$, where $\mathbf{0}^n$ is a vector of size n having all entries equal to 0, the Rayleigh quotient (Horn and Johnson, 1985) ensures that

$$x_{kt}^\top A_t^{-1} x_{kt} \leqslant \lambda_{max} x_{kt}^\top x_{kt},$$

where λ_{max} is the largest eigenvalue of A_t^{-1}. As the arm that maximizes the UCB is uncertain, we bound $x_{kt}^\top x_{kt} \leqslant \mathbf{1}^{n\top}\mathbf{1}^n = n$, where $\mathbf{1}^n$ is a vector of size n having all entries equal to 1. In this way, the following inequality holds:

$$x_{kt}^\top A_t^{-1} x_{kt} \leqslant \lambda_{max} n. \tag{A.1}$$

Lastly, as $u \in \mathbb{N}$, we define the upper bound of $x_{kt}^\top A_t^{-1} x_{kt}$ as

$$u = \lceil \lambda_{max} n \rceil \tag{A.2}$$