IISE Transactions

ISSN: (Print) (Online) Journal homepage: https://www.tandfonline.com/loi/uiie21

Selection of auto-carrier loading policy in automobile shipping

Sajeeb Kumar Kirtonia, Yanshuo Sun & Zhi-Long Chen

To cite this article: Sajeeb Kumar Kirtonia, Yanshuo Sun & Zhi-Long Chen (06 Dec 2023): Selection of auto-carrier loading policy in automobile shipping, IISE Transactions, DOI: 10.1080/24725854.2023.2291469

To link to this article: https://doi.org/10.1080/24725854.2023.2291469

IISE TRANSACTIONS
https://doi.org/10.1080/24725854.2023.2291469

Selection of auto-carrier loading policy in automobile shipping

Sajeeb Kumar Kirtonia[a], Yanshuo Sun[a], and Zhi-Long Chen[b]

[a]Department of Industrial and Manufacturing Engineering, FAMU-FSU College of Engineering, Florida State University, Tallahassee, FL, USA; [b]Robert H. Smith School of Business, University of Maryland, College Park, MD, USA

ABSTRACT

Auto-carriers are widely used to ship automobiles by land from origins to destinations. To enable the compact storage of multiple automobiles, auto-carriers are specially designed such that automobiles can only be loaded and unloaded through a common exit of an auto-carrier, which complicates the automobile loading and unloading operations. This study is motivated by the lack of consensus in the automobile shipping literature regarding whether reloading operations should or should not be prohibited while auto-carriers are en-route. The impact of a loading policy on auto-carrier shipping is not well understood in the literature. We thus examine two types of loading policies (namely reloading prohibited versus allowed), and design network-based optimization methods for each resulting policy variant. We then conduct extensive numerical experiments based on the data from the Southeast region of the USA to investigate the impact of a loading policy on automobile shipping operations through a trade-off analysis between solution quality and computational burden. We find that two proposed policy variants when reloading is allowed can achieve a desirable compromise between cost efficiency and computational effort. A full-scale analysis involving 10 auto-carriers with various capacities further confirms that with these policy variants, substantial cost savings are achieved with reasonable computation effort. The research findings from this article are expected to inform the choice of an appropriate auto-carrier loading policy for automobile transportation companies.

ARTICLE HISTORY
Received 25 April 2022
Accepted 21 November 2023

KEYWORDS
Automobile shipping; loading optimization; space-state network; tradeoff analysis

1. Introduction

Each year, more than 10,000,000 light vehicle units (automobiles and light trucks) are sold in the United States (U.S.). For instance, approximately 15,000,000 light vehicles were sold in 2021 (Statista, 2022), despite the COVID-19 pandemic. Those finished vehicles are shipped from automobile manufacturers or importers to automobile dealerships by in-house or third-party logistics providers. Besides the distribution of new vehicles, preowned vehicles are transported throughout the U.S. as people relocate from one state to another for better employment opportunities or more attractive living conditions. Those individual automobile shippers rely on automobile shipping companies if they do not drive or tow their own vehicles long distance. Unlike finished vehicles, which tend to have clustered origins (e.g., automotive assembly plant), preowned vehicles usually have distinct origins and destinations (e.g., previous or new home location). For economic reasons, automobiles with similar destinations and delivery timelines are consolidated and carried by auto-carriers. A general class of decision-making problems concerned with shipping automobiles from origins to destinations by auto-carriers is known as the automobile shipping optimization problem. This problem distinguishes itself from other distribution problems (Sun *et al.*, 2021), mainly because of the close interrelation between auto-carrier routing and loading decisions, arising from the special trailer configuration of an auto-carrier (Agbegha *et al.*, 1998).

An auto-carrier usually has a single exit, which is used for both automobile loading and unloading. As there are various automobile types (such as compact cars and minivans), automobiles must be assigned in a limited number of ways to meet various loading restrictions. For instance, a small auto-carrier slot cannot hold a large automobile (referred to as single-car constraints); two automobiles exceeding a size threshold cannot be in a certain pair of slots (referred to as pairwise constraints). The single-exit auto-carrier configuration and its associated loading constraints imply that when an automobile is ready to be delivered, other automobiles obstructing the unloading path of the destined automobile must be temporarily unloaded and loaded back afterwards. This operation is called a *reload*. As reload operations are very time-consuming, expensive, and may involve safety risks, reloading should be minimized from the perspective of loading/unloading only. Nonetheless, prohibiting reloading can have a negative impact on auto-carrier routing decisions. Conversely, allowing reloading may yield significant auto-carrier travel cost savings. Although this interrelation between loading/unloading and

CONTACT Yanshuo Sun y.sun@eng.famu.fsu.edu
Supplemental data for this article can be accessed online at https://doi.org/10.1080/24725854.2023.2291469.

routing is not difficult to describe, it is challenging to incorporate this interrelation in an optimization model, because of the additional computation effort needed, which is clearly dependent on the extent to which reloading is allowed. This creates a stark contrast between two streams of studies in the literature: studies where reloading is allowed while auto-carrier routes are fixed, versus studies where reloading is prohibited while routes are optimizable (see more detailed reviews in Section 2). The joint optimization of intercorrelated auto-carrier loading/unloading and routing decisions requires a full understanding of the cost efficiency (measured by total cost, including number of reloads and route cost) and computational burden (measured by total computation time), and how they vary with the degree to which auto-carrier reloading is allowed. Unfortunately, there are no such studies in the current literature. Therefore, this article seeks to enrich the automobile shipping optimization literature by investigating what specific auto-carrier loading policy can strike a desirable balance between solution quality and solution time. Another research gap this article fills is that in almost all existing auto-carrier loading optimization studies, including the most recent work by Bonassa *et al.* (2023), it was assumed that all pickups occur at the beginning of an auto-carrier route. In this study, we examine a general pickup and delivery route where any stop of the route can be a pickup and/or delivery location. This is very common in shipping preowned vehicles for private customers which generally have different pickup and delivery locations that can be spatially distributed in a random manner.

We propose a unified solution framework based on what we call space-state graphs to solve the loading optimization problem when auto-carrier routes with pickup and delivery are given. The space-state graph is inspired by the observation that we essentially seek to optimize how the automobile-to-slot assignment (represented as loading state) should vary along a predetermined auto-carrier route, i.e., over space. With this graph, the auto-carrier loading optimization problem for any given auto-carrier route is converted to some network flow problem that can be solved efficiently. Under each loading policy, we customize how the space-state graph is generated and how the optimal loading/unloading plan is determined. Next, we conduct numerical studies using a range of problem instances for an anonymized automobile transportation company in Southeast U.S. and identify the desirable loading policy based on the trade-off between solution quality and time. We also explore how a few key inputs influence the performance of a selected loading policy. Based on the findings from our numerical studies, we conclude that in automobile shipping, allowing reloading while considering only a subset of all feasible loading states and state transitions is preferable because it strikes a desirable balance between cost efficiency and computational burden.

This article makes the following contributions:

1. We prove that finding even a feasible loading plan for the loading optimization problem defined by Agbegha *et al.* (1998) is strongly NP-complete, when the numbers of cars and slots are arbitrarily large. This means that it is impossible to solve this problem optimally in a time that is a polynomial function of the numbers of cars and slots, unless $P = NP$.
2. The loading optimization framework that we develop based on space-state graphs is new and differs from the existing ones in the literature (e.g., Agbegha *et al.* 1998). Through comparison we find better loading plans (with fewer reloads) for routes in problem set #3 in Agbegha *et al.* (1998), as documented in the online Appendix B. Our approach is also a unified method that works for any given loading policies. Consequentially, this approach provides an efficient and fair evaluation of alternative loading policies.
3. We fill the literature gap that no studies have explored how the extent to which reloading is allowed affects the trade-off between the total automobile shipping cost and the additional computational time needed for auto-carrier operators. We are thus the first to derive several key managerial insights into the choice and impact of auto-carrier loading policy. For instance, we find that a proposed loading policy that limits the automobile reshuffling operations (namely Policy 1b, presented in Section 4.2) can yield the optimal loading solution in virtually all randomly generated instances, while requiring less than 25% of the computation time needed by a brute-force policy that guarantees loading optimality (i.e., Policy 1a). In addition, we find that with minimum computation time, a greatly simplified version of Policy 1b (i.e., Policy 1c) can significantly improve the loading solution quality relative to Policy 0, which prohibits reloading. Therefore, those key insights into the trade-off of computation time and cost efficiency can inform the choice of an appropriate auto-carrier loading policy when the joint optimization of auto-carrier routing and loading is pursued in a future study.

The rest of this article is organized as follows. Section 2 reviews the existing literature on loading optimization involved in automobile and non-automobile shipping problems. Section 3 defines the auto-carrier loading problem and presents a computational complexity analysis. Section 4 describes the space-state network-based solution approach for the loading optimization problem. Section 5 describes numerical studies and reports major research findings. Section 6 concludes the article with a summary of managerial insights and future research directions.

2. Literature review

2.1. Auto-carrier loading optimization problem

Agbegha *et al.* (1998) were the first to define and study the auto-carrier loading optimization problem. They developed a quadratic assignment formulation to optimize how automobiles should be assigned to various auto-carrier slots. In the predetermined distribution route, all automobiles are

picked up simultaneously at one location and dropped off possibly at multiple locations, implying the one-to-many distribution pattern. One shortcoming of this seminal work (Agbegha *et al.*, 1998) was that it required that reloaded automobiles must be loaded back to the original slots. Therefore, the automobile-to-slot assignment was optimized only once at the beginning of the predetermined route. The optimization model, with the objective of minimizing the total number of reloads, was solved with a heuristic based on a branch-and-bound procedure. Their numerical studies indicated that it took a few seconds on average to find an optimized loading plan for a given route, although their algorithm failed to solve some of their test instances. A few other studies (Lin, 2010; Chen, 2016) have tried to improve on Agbegha *et al.* (1998), for instance, by allowing reloaded automobiles to be loaded back to any feasible slots after temporary unloading. A similar formulation, namely quadratic assignment, was used in Lin (2010) and was solved directly by a nonlinear programming solver. One clear research gap is that in all available auto-carrier loading studies, only one-to-many distribution routes were considered while the general case with multiple pickup locations and multiple delivery locations (i.e., a many-to-many demand pattern), which is common in the shipping of preowned vehicles, was not considered.

While the above reviewed studies were focused on loading optimization under the assumption that auto-carrier routes were given, a few other studies optimized auto-carrier routes considering various levels of auto-carrier loading complexities. Dell'Amico *et al.* (2015) were among the earliest researchers to optimize auto-carrier routes while employing a sophisticated procedure to check whether certain loading restrictions were satisfied for a given route. Unlike Agbegha *et al.* (1998), Dell'Amico *et al.* (2015) did not model an auto-carrier trailer as a collection of discrete slots; they instead considered a continuous loading plane with length limits, which was considered earlier by Tadei *et al.* (2002). The same way of modeling loading capacity was also used in Tadumadze and Emde (2021). No reloading was needed in Dell'Amico *et al.* (2015), as it imposed the restriction that all automobiles must be loaded and unloaded in a Last-In-First-Out (LIFO) fashion. Wang *et al.* (2018) mainly optimized how automobiles should be assigned to auto-carriers and how auto-carriers should be routed considering a so-called downward compatible loading constraint. Under this constraint, a large auto-carrier slot can hold any automobiles whereas a small slot can store only a small car. Essentially, this was called single-car constraints in Agbegha *et al.* (1998). Wang *et al.* (2018) considered such loading restrictions when assigning automobiles to auto-carriers, but did not optimize loading decisions or explore the possibility of reloading. In a follow-up study, Chen and Wang (2020) focused on auto-carrier loading optimization while simplifying routing optimization by approximating the actual transportation cost of a route using a simple cost estimation. They considered the same downward compatibility loading constraint, but did not consider reloading. Juárez Pérez *et al.* (2019) adopted a two-phase heuristic (routing first, loading second) for the automobile shipping problem. In the second phase of the heuristic, they checked the loading feasibility of a route generated in the first phase. They assumed the LIFO loading policy, which implied reloading prohibition. In all auto-carrier routing studies including the most recent one by Bonassa *et al.* (2023), it was assumed that automobiles were picked up at the same location, with the exception that Chen and Wang (2020) allowed multiple pickup locations prior to any drop-offs. Clearly, all such routes are only a special case of a general pickup and delivery route, which is considered in this study. For a more comprehensive review of the auto-carrier routing and loading optimization studies, see Sun *et al.* (2021).

The above review also indicates that in the existing studies, when auto-carrier routes were fixed, reloading was allowed in optimizing auto-carrier loading decisions; when routing optimization was involved, reloading was not allowed. It can be expected that when both routing optimization and reloading are allowed, the computational burden can increase significantly. For instance, for a given auto-carrier route, if reloading is allowed, it may take only a second to optimize the loading plan (Agbegha *et al.* 1998). However, when there are N auto-carrier routes to evaluate for the purpose of routing optimization, the total computation time spent on solving the loading subproblem becomes N seconds. Clearly, in a practical context, N should be at least in the order of tens of thousands (Dell'Amico *et al.*, 2015). Despite the significant computational efforts, allowing reloading is expected to yield substantial savings in routing cost, as will be demonstrated in Section 5 of this article. Thus, it is necessary to fully understand how the degree to which reloading is allowed affects the trade-off between solution quality and computational burden, which has not been explored in any existing study in the literature. In practice, a full understanding of such a trade-off is essential in choosing a proper auto-carrier loading policy.

2.2. Loading policies in non-automobile transportation

In transporting non-automobile commodities, different carrier configurations presented other loading and unloading challenges. For instance, Petersen and Madsen (2009) considered a special truck with multiple independent rows, each of which was modeled as a LIFO stack. After an item was assigned to one of the rows, it became inaccessible until it was at the top of its assigned stack. Battarra *et al.* (2010) studied a special Traveling Salesman Problem (TSP) with additional handling operations. There were two types of commodities, namely commodities of type *a* for delivery only (e.g., functioning bikes) and commodities of type *b* for picking up only (e.g., defective bikes). Each customer for visiting had both types of commodities. As Battarra *et al.* (2010) concluded it was too complex to optimize the TSP tour and vehicle loading decisions, so they introduced three handling policies to simplify the optimization problem. In policies 1 and 2, all commodities of type *b* were stored at the rear or front of a truck, respectively; the third policy was a hybrid one. Those handling policies were compared

through extensive numerical experiments in Battarra *et al.* (2010). Veenstra *et al.* (2017) also studied the TSP with handling operations and reported that handling costs were reduced significantly while a small increase in vehicle distance was incurred.

In the above reviewed studies, one truck had a single or multiple independent stacks for loading. In some other cases, the loading space of a truck can be modeled as a continuous two-dimensional (2D) or three-dimensional (3D) space. For instance, Lee *et al.* (2013) considered height and width of steel slabs in packing. Bortfeldt and Yi (2020) assumed that cargos were rectangular shaped, and a truck had a 3D space with fixed dimensions for cargo storage. In Bortfeldt and Yi (2020), cargos of each customer can be freely unloaded without moving cargos of other customers, because the LIFO loading policy was enforced. Reil *et al.* (2018) considered two separate compartments for linehaul and backhaul with similar 3D loading constraints. Bukchin and Sarin (2004) modeled a truck with several different compartments, each of which was allowed to contain a single product type.

By contrast, as indicated by Agbegha *et al.* (1998), an auto-carrier is characterized by a set of discrete slots and complex precedence relations, and hence, cannot be adequately modeled with a single or multiple independent LIFO stacks. Nor can it be modeled as a 2D or 3D space. That explains why Agbegha *et al.* (1998) introduced a loading network to model the special trailer configuration of an auto-carrier. In addition, the unique single-car and pairwise loading restrictions must be met when automobiles are loaded, which are not present in shipping non-automobile cargos.

While in the transportation of non-automobile cargos, various loading policies were defined and compared in the literature, the distinct configuration of an auto-carrier implies that the findings or managerial insights for non-automobile shipping may not be transferrable to the case of automobile shipping.

2.3. Research gaps

The literature review yields three research gaps as follows. First, although the auto-carrier loading optimization was studied by quite a few researchers, the computational complexity of this problem has not been analyzed. Second, in all existing studies, except for Wang *et al.* (2018), a restrictive route structure where all pickups occur at the beginning of an auto-carrier route was considered. A more general structure where pickup and/or delivery can occur at any stop of the route has not been studied. The third literature gap is that none of the existing automobile shipping studies have explored how the trade-off between solution quality and computational burden varies with the extent to which automobile reloading is permitted.

3. Problem statement and complexity analysis

We define the auto-carrier loading optimization problem for a given route precisely and prove that even a very special case of this problem is strongly NP-complete when the number of the auto-carrier slots is arbitrary, which means that unless $P = NP$, any exact algorithm for this problem has a worst-case running time exponential in the problem input length, including the number of slots.

3.1. Description of the auto-carrier loading optimization problem

Given an auto-carrier and a pickup and delivery route to be executed by this auto-carrier, the loading optimization problem is to determine how the automobile orders (or simply orders) covered by the given route should be loaded and unloaded at each stop of the route to minimize the total loading cost (e.g., measured by the number of reloads over the entire route), subject to various loading constraints associated with the auto-carrier. We consider the most general version of the problem where different orders may or may not share the same pickup and/or delivery locations (or auto-carrier stops), which means each auto-carrier stop could be a pickup and/or delivery location for one or multiple orders.

Let K denote the set of orders to be picked up and delivered by the auto-carrier by visiting the given route. The auto-carrier has a set of available slots I at the beginning of the route and is associated with loading restrictions. Let R be the set of stops covered by the given auto-carrier route, where the stops are denoted as 0, 1, 2, …, $|R|$ in the sequence visited by the auto-carrier. Except for stop 0 representing the auto-carrier depot, each stop r from 1 to $|R|$ could be a pickup location, or a delivery location, or both. We thus define two sets of orders for each stop r as follows: a set of orders for pickup, denoted as $K_r^P \subseteq K$, and a set of orders for delivery, denoted as $K_r^D \subseteq K$. At least one of these two sets is nonempty. Picking up or dropping off orders at a stop may incur a loading cost because some orders on the auto-carrier may have to be unloaded first to allow other orders to be loaded or unloaded, and loaded back to the auto-carrier. The loading optimization problem is to find a loading solution at each stop of the given route such that the total loading cost incurred along the route is mimimized without violating the loading constraints.

Although auto-carriers have various trailer types and capacities, we follow Agbegha *et al.* (1998) and model the trailer configuration of an auto-carrier with a tree-like loading structure, illustrated in Figure 1, where each circle represents a slot, and each arc from slot i to slot j defines a precedence relation, namely slot i must be empty before an automobile in slot j can be unloaded through slot i. When loading cars to the slots of the auto-carrier, two types of loading constraints must be satisfied, namely (i) single-car constraint: certain cars cannot be assigned to certain slots; (ii) pairwise constraints: there are pairs of slots such that, for each pair of slots, if a specific car is assigned to one of these slots, some other specific cars may not be assigned to the other slot. For instance, slots 3 and 4 of the six-slot auto-carrier in Figure 1 form such a pair. Both sets of constraints are considered by Agbegha *et al.* (1998) and some

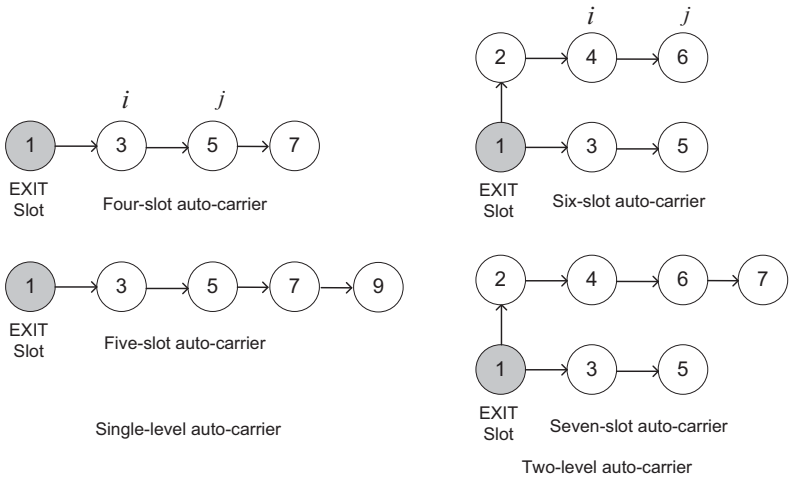

Figure 1. Loading structures of auto-carrier trailers.

follow-up studies, e.g., Lin (2010) and Chen (2016). The single-car constraint is considered by Wang *et al.* (2018).

We note that all the loading optimization problems studied in the literature, as reviewed in Section 2, are special cases of our problem. For example, in the problems studied by Agbegha *et al.* (1998) and Dell'Amico *et al.* (2015), all orders are picked up at a common pickup location, and in the problem considered by Wang *et al.* (2018), all the orders are picked up first from several warehouses before being delivered to dealers. Furthermore, in the problem studied by Agbegha *et al.* (1998), it is assumed that when a car is temporarily unloaded and loaded back, it must go back to its original slot. This assumption makes reloading more convenient, but limits the solution space for reloading, and hence, may increase loading cost. By contrast, in our problem, we do not make such an assumption, and hence, orders temporarily unloaded can be loaded back to any slots on the auto-carrier as long as the loading constraints are not violated.

Furthermore, our problem has a more complex structure and is likely to have a larger scale than these existing problems because: (i) in our problem, any stop of the given route can have pickups and/or deliveries, which makes our problem dynamic, whereas these existing problems are static (i.e., all decisions are made only once at the beginning of a given route); and (ii) the number of orders involved in these existing problems is in the same magnitude as the number of slots, whereas in our problem the number of orders involved can be much larger than the number of slots.

3.2. Complexity analysis of the auto-carrier loading optimization problem

Although quite a few studies have considered the loading optimization problem, the computational complexity of these problems is not fully analyzed. Agbegha *et al.* (1998) show that in the absence of loading constraints (i.e., any car can be assigned to any slot of the auto-carrier), then their problem can be solved optimally by a simple car-to-slot assignment rule, which can be constructed in polynomial time of the number of slots. However, we are not aware of any existing results on the complexity of loading optimization problems when loading constraints are present. In this section, we clarify the complexity of our problem by first analyzing the complexity of the problem studied by Agbegha *et al.* (1998) where there are loading constraints, as described in Section 3.1.

As discussed above, in the problem studied by Agbegha *et al.* (1998), as well as the problems studied by several other papers reviewed in Section 2.1, all the pickups are carried out prior to any deliveries, i.e., all pickups take place at the first stop or first few stops of the given auto-carrier route, and no orders are picked up at the subsequent stops of the route. Denote the number of auto-carrier slots as m. Clearly, in these problems, the number of orders is no greater than m, and the number of stops in the given route is no greater than $2m$. When the number of slots m is fixed (hence the number of cars and number of stops on the given route are also fixed), these problems can be solved in constant time because enumerating all feasible solutions takes at most $O(m!)$ time, which is fixed (i.e., a constant time). However, when the number of slots m is arbitrary, we show in the following theorem that the problem studied by Agbegha *et al.* (1998) becomes intractable.

Theorem 1: *When the number of cars to be loaded and the number of slots available are arbitrary, finding even a feasible loading plan for the loading problem studied by Agbegha et al. (1998) is strongly NP-complete.*

Proof: We show this by a reduction from a 3-dimensional matching (3DM) problem, which is known to be strongly NP-complete (Garey and Johnson, 1979). 3DM can be described as follows: Given three disjoint sets W, X, and Y, each with q elements, i.e., $W = \{w_1, ..., w_q\}$, $X = \{x_1, ..., x_q\}$, and $Y = \{y_1, ..., y_q\}$, and a set of triples $M \subseteq W \times X \times Y$, i.e., each element in M is a triple (w_i, x_j, y_k) where $w_i \in W$, $x_j \in X$ and $y_k \in Y$. The question asks whether M contains a matching, i.e., a subset $\bar{M} \subseteq M$ such that $|\bar{M}| = q$ and no two elements of $\bar{M}$ agree on any coordinate.

Given any instance of 3DM, for each w_i, for $i = 1, ..., q$, define the set of tuples, $T_i = \{(x_j, y_k) | (w_i, x_j, y_k) \in M\}$, the

set of x-coordinates covered by T_i, i.e., $X_i = \{x_j \in X|$ there is some $y_k \in Y$ such that $(x_j, y_k) \in T_i\}$, and the set of y-coordinates covered by T_i, i.e., $Y_i = \{y_k \in Y|$ there is some $x_j \in X$ such that $(x_j, y_k) \in T_i\}$. Furthermore, for each pair of (i,j) with $(w_i, x_j, y_l) \in M$ for some $y_l \in Y$, we define the set of feasible y-coordinates, $Y_{ij} = \{y_k \in Y|(w_i, x_j, y_k) \in M\}$. By these definitions, we can observe that $T_i = \{(x_j, y_k)|x_j \in X_i$ and $y_k \in Y_{ij}\}$.

We construct a corresponding instance of the loading problem as follows: There are $m = 2q$ cars to be loaded into $2q$ slots of an auto-carrier. Divide the cars into q pairs such that the two cars in each pair have to follow some loading constraints to be defined below. Index the q pairs of cars as $(w_{11}, w_{12}), ..., (w_{q1}, w_{q2})$, such that each pair of cars (w_{i1}, w_{i2}) corresponds to the element w_i in W, for $i = 1, ..., q$. Index the $2q$ slots as $(x_1, ..., x_q; y_1, ..., y_q)$, such that the first q slots correspond to the elements in X and the last q slots correspond to the elements in Y. We construct the following loading constraints that must be satisfied in any feasible solution:

1. Single-car constraints: For each $i = 1, ..., q$, the single-car constraints for the two cars (w_{i1}, w_{i2}) are defined as follows: car w_{i1} can only be loaded to slots in X_i, and cannot be loaded to any slot in $(X \backslash X_i) \cup Y$; and car w_{i2} can only be loaded to slots in Y_i, and cannot be loaded to any slot in $(Y \backslash Y_i) \cup X$.
2. Pairwise constraints: For each $i = 1, ..., q$, there are pairwise constraints for the two cars (w_{i1}, w_{i2}) as follows: for $h = 1, ..., q$, if car w_{i1} is loaded to a slot $x_h \in X_i$, then car w_{i2} can only be loaded to slots in Y_{ih} and cannot be loaded to any slot in $Y_i \backslash Y_{ih}$.

Observation:

We first observe that for the constructed instance, in any feasible solution that satisfies the above defined loading constraints, each pair of two cars (w_{i1}, w_{i2}), for $i = 1, ..., q$, must be loaded to the two slots corresponding to a tuple in T_i, respectively. This implies that for every $i = 1, ..., q$, there is a one-to-one correspondence between every feasible loading plan for the two cars (w_{i1}, w_{i2}) and a specific triple (w_i, x_j, y_k) in M.

Clearly, the above instance of the loading problem can be constructed in polynomial time.

We show in the following that there is a solution to the constructed instance of the loading problem if and only if there is a match $\bar{M} \subseteq M$ such that $|\bar{M}| = q$ and no two elements of $\bar{M}$ agree on any coordinate.

"If part": If there is a such a match $\bar{M} \subseteq M$, then $\bar{M} = \{(w_i, x_{[i]}, y_{\langle i \rangle})|i = 1, ..., q\}$, where each $x_{[i]}$ corresponds to a distinct element in X, and each $y_{\langle i \rangle}$ corresponds to a distinct element in Y, i.e., $\{x_{[i]}|i = 1, ..., q\} = X$ and $\{y_{\langle i \rangle}|i = 1, ..., q\} = Y$. We can constructure a feasible loading plan for the constructed instance of the loading problem as follows: load car w_{i1} to slot $x_{[i]}$ and load car w_{i2} to slot $y_{\langle i \rangle}$, for $i = 1, ..., q$. Since $\bar{M} \subseteq M$, we have: $(x_{[i]}, y_{\langle i \rangle}) \in T_i$, for $i = 1, ..., q$. Thus, by the Observation we made earlier, this is a feasible loading plan.

"Only if part": If there is a feasible solution to the constructed instance of the loading problem, then in any given feasible solution, for each pair of two cars (w_{i1}, w_{i2}), for $i = 1, ..., q$, by constraint 1, car w_{i1} must be assigned to a slot in X_i, and we denote this slot as $x_{[i]}$, and similarly, by constraint 2, car w_{i2} must be assigned to a slot in $Y_{i[i]}$ and we denote this slot as $y_{\langle i \rangle}$. The Observation we made earlier implies that $(x_{[i]}, y_{\langle i \rangle}) \in T_i$, for $i = 1, ..., q$. Furthermore, since there are $2q$ cars and $2q$ slots, each slot is occupied by exactly one car in the given solution. Thus, $\{x_{[1]}, ..., x_{[q]}\} = X$, and $\{y_{\langle 1 \rangle}, ..., y_{\langle q \rangle}\} = Y$. Therefore, $\{(w_i, x_{[i]}, y_{\langle i \rangle})|i = 1, ..., q\}$ is a match for the 3DM instance such that $|\{(w_i, x_{[i]}, y_{\langle i \rangle})|i = 1, ..., q\}| = q$ and no two elements in it agree on any coordinate.

We note that Theorem 1 means that the time it takes for any algorithm to find an optimal (or even a feasible) solution for the problem studied by Agbegha *et al.* (1998) must be an exponential function of the length of the problem input, including m, unless $P = NP$. This also implies that it is not possible to solve this problem to optimality in time that is a polynomial function of m, unless $P = NP$. Since our problem is more general than the problem studied by Agbegha *et al.* (1998), these same conclusions apply to our problem as well. It should be noted that although in theory, the complexity of our problem is similar to that of the problem studied by Agbegha *et al.* (1998), due to the more complex structure and more dynamic nature of our problem, as discussed in Section 3.1, our problem is more difficult to solve.

One of the algorithms (i.e., Policy 1a) we present in Section 4 solves our problem to optimality with a running time that is exponential in the number of auto-carrier slots m, but polynomial in the number of cars and number of stops on the given auto-carrier route. Thus, by Theorem 1, this is the best possible exact algorithm one can have for our problem from a theoretical point of view.

4. Space-state network-based approach

4.1. A motivating example for the proposed approach

The essence of the auto-carrier loading optimization is to decide how the loading state of an auto-carrier changes along the auto-carrier route (i.e., over space), where a *loading state* specifies what automobiles are on the auto-carrier and what specific auto-carrier slots these automobiles occupy. An illustrative instance of the auto-carrier loading problem is presented to motivate our space-state network-based solution approach to be presented in Section 4.2. A three-slot auto-carrier's route (0, 1, 2, 3) is given in Figure 2. Note that for simplification purposes, a distribution route is illustrated here. This auto-carrier will pick up three automobiles (one compact car and two minivans) at the common origin (denoted as stop 0) and visit customers 1, 2, and 3 (denoted as stops 1, 2, 3, respectively) sequentially. Customer 3 orders a compact car 3, whereas customers 1 and 2 order minivans 1 and 2, respectively. The exit slot of the auto-carrier (located at the rear of the auto-carrier) can hold a compact car, but not a minivan, due to the size limit.

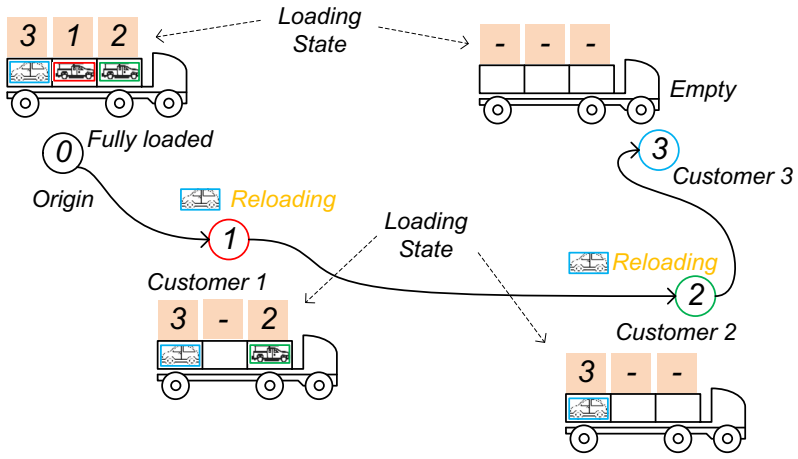

Figure 2. Example of loading/reloading decisions.

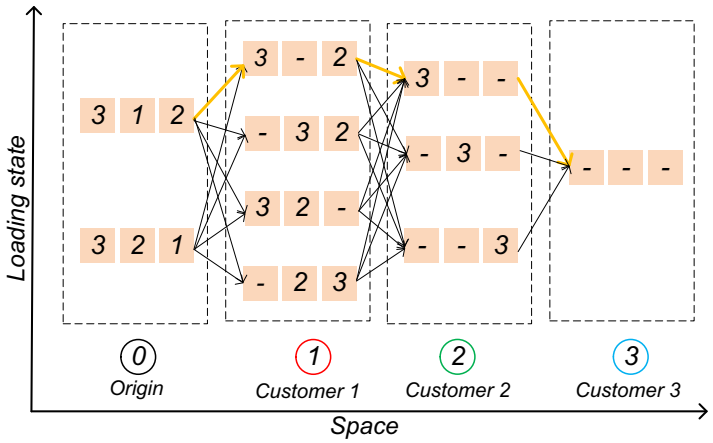

Figure 3. Space-state network for the example given in Figure 1.

The other two slots can hold any automobiles. Figure 2 highlights a specific loading plan, which consists of a specific loading state at each stop of the route. Figure 3 shows what we call the *space-state network*, which includes all possible loading plans for the given auto-carrier route, where a loading plan consists of one specific loading state at each stop of the route. Since the exit slot cannot hold minivans, the compact car ordered by customer 3 must be loaded to the exit slot to ensure the auto-carrier serves all three customers. Thus, there are two possible loading states after the auto-carrier leaves the origin, i.e., [3,1,2] and [3,2,1], as shown in Figure 3. Likewise, all possible loading states of the auto-carrier after visiting each customer can be found. At the end of the auto-carrier route, all slots are vacant. Figure 3 essentially shows how the loading states evolve over space, which is why the network in Figure 3 is referred to as a space-state network.

To illustrate, we describe the loading plan shown in Figure 2 using a sequence of loading states generated in Figure 3. This loading plan starts with the loading state [3, 1, 2] at the origin. When the auto-carrier visits customer 1 next, the compact car must be unloaded and loaded back to facilitate the unloading of the minivan ordered by customer 1. The resulting loading state becomes [3, -, 2], where the center slot is empty and marked as "-". Before the minivan ordered by customer 2 is delivered, the compact car should be unloaded and reloaded again. Thus, the loading state after visiting customer 2 becomes [3, -, -]. Finally, at customer 3, the compact car is unloaded and the loading state becomes [-, -, -]. Thus, a total of two reloads are needed when the illustrated loading plan is adopted for this given auto-carrier route under Policy 1. By contrast, this auto-carrier route becomes infeasible under Policy 0, because this policy prohibits reloading, whereas any loading plan shown in Figure 3 requires at least two reloads.

The network in Figure 3 does not yet show how costly a transition from one loading state to another is. The cost of change from state [3,1,2] to [3,-,2], for example, can be computed as follows. To transition from [3,1,2] to [3,-,2], an unload of car 3, an unload of minivan 1, and a reload of car 3 are needed sequentially. Since the unloading and reloading of car 3 are extra handling operations, which are only needed to enable the delivery of the minivan 1 to customer 1, the transition cost from [3,1,2] to [3,-,2] is evaluated as one reload. A formal procedure for computing the transitional cost is given in Section 4.2.3.

4.2. Space-state network-based solution approach

4.2.1. Formulation of auto-carrier loading constraints

To better understand the single-car and pairwise loading constraints involved in the loading optimization problem, we first formulate them mathematically. For an automobile order $k \in K$, the set of all feasible slots is denoted as $I_k \subseteq I$. Let x_{ki} be a binary variable, which has a value of one if automobile k is assigned to slot $i \in I_k$. Then, the loading constraints are formulated as follows:

$$\sum_{i \in I_k} x_{ki} = 1, \forall k \in K \tag{1}$$

$$\sum_{k \in K} x_{ki} \leq 1, \forall i \in I \tag{2}$$

$$\sum_{k \in K} b_k x_{ki} + \sum_{\bar{k} \in K} b_{\bar{k}} x_{\bar{k}j} \leq B_{ij}, \forall (i,j) \in P_I, i < j, k \neq \bar{k} \tag{3}$$

$$x_{ki} \in \{0,1\}, \forall k \in K, i \in I_k \tag{4}$$

Constraint (1) requires each automobile to be loaded exactly once. Constraint (2) restricts each auto-carrier slot to accept at most one automobile. Constraint (3) is used to incorporate pairwise constraints for a pair of interrelated slots (i,j), where P_I is the set of all such interrelated slot pairs; b_k represents the equivalent size of automobile k; B_{ij} is the size limit associated with pair (i,j).

Although Agbegha *et al.* (1998) provided pairwise constraints for the example auto-carriers in Figure 1, they did not formulate those constraints mathematically. We show that constraint (3) can be used to model the pairwise constraints in Agbegha *et al.* (1998). For the seven-slot auto-carrier in Figure 1 and three types of automobiles for loading (namely Types 1, 2, and 3, or T1, T2, and T3 for brevity), Agbegha *et al.* (1998) introduced the following pairwise constraint: if an automobile of Type 2 or 3 is assigned to slot 3, another automobile of Type 3 cannot be assigned to slot 4; likewise, if a T2 or T3 automobile is assigned to slot 4, a T3 automobile cannot be assigned to slot 3. Essentially, if one of the two pairing slots (3 and 4) holds a T3 automobile, the other slot can only hold a T1 automobile. If a T3 automobile is not involved, slots 3 and 4 can accept other combinations of automobiles, such as two T2 automobiles. Similar pairwise constraints are defined for another pair of slots, namely slots 5 and 6. To model such restrictions, we define $P_I = \{(3,4),(5,6)\}$. The parameter b_k is a proxy for the size of automobile k. If automobile k is of Type a, we set b_k to be a, where $a \in \{1,2,3\}$. The limit B_{ij} can be set to a value of four, namely $B_{34} = 4$ and $B_{56} = 4$. Once those parameters are specified as above, constraint (3) ensures that the pairwise loading constraints described by Agbegha *et al.* (1998) are incorporated. Constraint (4) defines x_{ki} to be binary.

Constraints (1)-(4) are checked when generating loading states (i.e., value of x_{ki}) at each auto-carrier stop under any loading policy, which is described in detail in Section 4.2.2.

It is worth noting that Mahmoudi and Zhou (2016) considered passenger carrying states in a ridesharing optimization study. However, those passenger carrying states are different from the auto-carrier loading states in two notable ways. First, in Mahmoudi and Zhou (2016), there are no single-passenger or pairwise constraints in generating passenger carrying states. Second, reloading or reshuffling is irrelevant in Mahmoudi and Zhou (2016), whereas in making auto-carrier loading decisions, reloading is essential and must be considered explicitly.

4.2.2. Space-state graph generation

Following the problem definition given in Section 3, for a given auto-carrier route, K denotes the set of orders to be picked up and delivered by an auto-carrier with a set of available slots I. We use R to denote the set of stops covered by the given auto-carrier route, where the stops are denoted as 0, 1, 2, … , $|R|$ in the sequence visited by the auto-carrier. We use K_r^P and K_r^D to denote the set of orders to be picked up and the set of orders to be dropped off at stop r, respectively. Immediately after visiting stop r, let K_r be the set of orders that are on the auto-carrier. Given a loading policy, a corresponding space-state graph can be generated accordingly. Regardless of which specific policy is used, a space-state graph can be generated using the general framework described below and where some steps can be customized for different policies, to be discussed later. In the procedures described below, N_r denotes the set of feasible loading states (interchangeably, vertices) generated for the orders in K_r right after visiting stop r.

Step 0: Create a null loading state at stop 0, denoted as a vector [-, -, … , -] with dimension $|I|$, where each "-" represents an empty slot of the auto-carrier. This null loading state is designated as the source vertex of the space-state graph.

Step r : At stop r, for $r = 1, ..., |R| - 1$, as orders in K_r^P and K_r^D are for pickup and drop-off, respectively, set $K_r = K_{r-1} \cup K_r^P \backslash K_r^D$. Generate a set of feasible loading states N_r for the orders in K_r by a procedure determined by the given policy.

Use a procedure determined by the given policy to add a directed edge to connect each vertex $\pi \in N_{r-1}$ at stop $r-1$ with all or a subset of the vertices in N_r at stop r.

Step $|R|$: At stop $|R|$, create a null loading state, denoted as a vector [-, -, … , -] with dimension $|I|$. This null loading state is intended to be the sink vertex of the space-state graph. Add a directed edge between each vertex $\pi \in N_{|R|-1}$ and the sink vertex.

Now we describe the specific loading policies we consider and how the above general framework for space-state graph generation can be customized for each policy. There are in general two types of loading policies, namely reloading prohibited (Policy 0) and allowed (Policy 1). When reloading is allowed, we consider three variants of Policy 1, namely Policies 1a, 1b, and 1c, which differ in the specific procedures involved in Step r of the above framework for space-state graph generation. As described below, Policy 1a generates all feasible vertices or loading states N_r for each stop r and all feasible edges, and hence, guarantees an optimal solution for the loading problem, whereas Policies 1b

and 1c generate a subset of feasible vertices and a subset of feasible edges only.

For Policy 1a, the following brute-force algorithm is employed in Step r to generate vertex set N_r and corresponding edges. We start with a single existing loading state [-, -, …, -] with dimension $|I|$, and a non-empty set of automobiles K_r. While $K_r \neq \emptyset$, insert an automobile from K_r to each unoccupied auto-carrier slot of every existing loading state to generate new loading states, considering the loading constraints (1)-(4). All resulting states replace existing loading states, and a new automobile from K_r is considered next. When $K_r = \emptyset$, all existing loading states together form set N_r. After generating N_r, add a directed edge to connect each vertex $\pi \in N_{r-1}$ at stop $r-1$ with all the states in N_r. Essentially, the above procedure generates every feasible solution that satisfies the constraints (1)-(4).

To illustrate, consider an example of an auto-carrier with four slots available to pickup and deliver orders for four customers following a given route (2+,3+,4+,1+,3-,2-,1-,4-), where "+" indicates pickup and "-" means drop-off. Suppose that the single-car constraint restricts that automobiles 3 and 4 can only occupy the middle two slots. Figure 4 shows how the space-state graph looks like under Policy 1a for this example, as well as space-state graphs generated under other policies. Note that in this illustrative route, at Stop r, one of K_r^P and K_r^D is a singleton set; the other is empty.

Although the brute-force algorithm used in Policy 1a can ensure solution optimality, the underlying graph may be extremely dense and thus too time-consuming to build. Our Policy 1b is thus created to generate a less dense space-state graph, which can yield a near optimal solution, but requires significantly less computation time. For Policy 1b, the specific procedures used in Step r of the space-state graph generation framework are as follows:

First, generate a set of feasible loading states $\bar{N}_r$ for the orders in $K_{r-1} \backslash K_r^D$ by the following *removal procedure*:

> For each loading state of stop $r-1$, denoted as $\pi \in N_{r-1}$, remove every $\overleftarrow{k}$ in K_r^D from π and reshuffle any reloaded orders when they are loaded back as follows: keep the ones not reloaded in the same slots as in π, and reshuffle the ones that are reloaded in any feasible way using the remaining slots. Reshuffling ensures that reloaded orders may be loaded back to the slots different from the slots they originally occupy in π, which could avoid or reduce the need for reloading at subsequent auto-carrier stops. Let $\bar{N}_r^\pi$ be the set of all the states generated. The states in $\bar{N}_r^\pi$ are called induced loading states from state π. The set of the loading states generated for stop r is thus $\bar{N}_r = \cup_{\pi \in N_{r-1}} \bar{N}_r^\pi$.

Note that when $K_r^D = \emptyset$, $\bar{N}_r = N_{r-1}$.

Second, generate a set of feasible loading states N_r for the orders in $K_{r-1} \backslash K_r^D \cup K_r^P$ by the following *removal procedure*:

> For each loading state $\pi \in \bar{N}_r$, try to insert every order $\overrightarrow{k}$ in K_r^P into each feasible slot in π considering the loading constraints ((2) and (3)). Let N_r^π be the set of the resulting

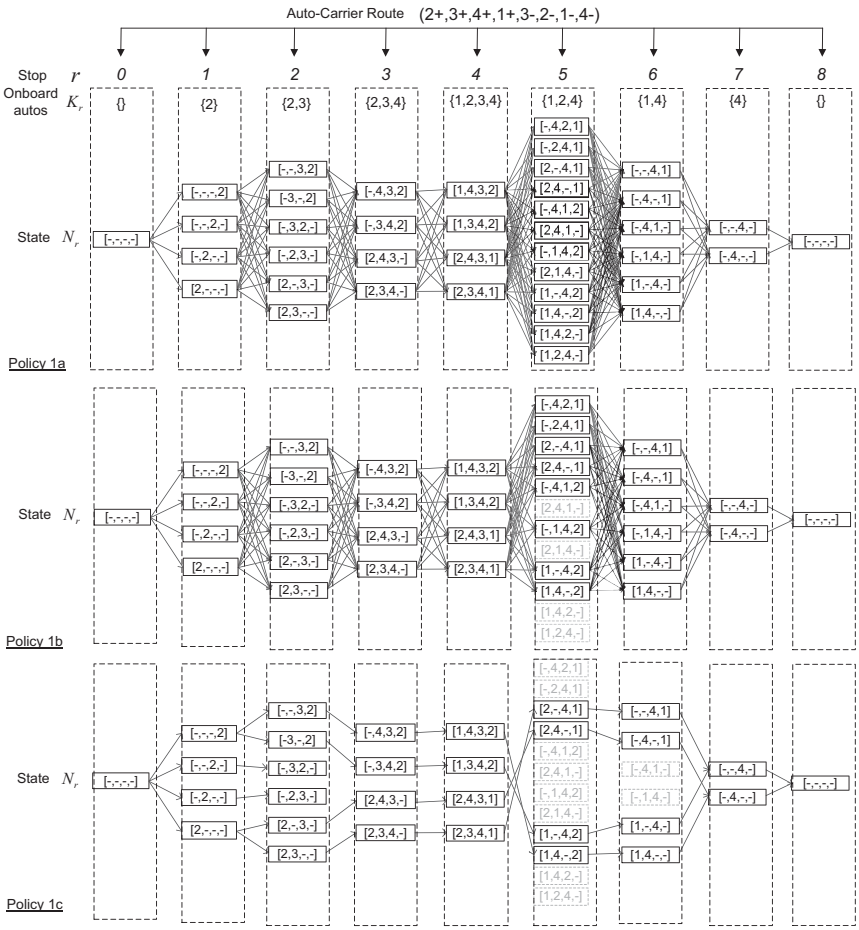

Figure 4. Space-state graphs generated under Policies 1a, 1b, and 1c.

loading states for orders in K_r. Those loading states in N_r^π are also referred to as "states induced by loading state π." The set of the loading states generated for stop r is thus $N_r = \cup_{\pi \in N_{r-1}} N_r^\pi$.

Note that when $K_r^P = \emptyset$, $N_r = \bar{N}_r$.

As reloading is permitted, it is possible to transition from one state to any other state, a directed edge is added to connect each vertex $\pi \in N_{r-1}$ at stop $r-1$ with each of the vertices in N_r at stop r.

The key difference between Policies 1a and 1b lies in how orders are rearranged after an order is dropped off. Under Policy 1a, all orders must be reshuffled regardless of whether they are reloaded ones or not, whereas under Policy 1b, only reloaded orders are reshuffled with those not for reloading kept intact. This difference explains why certain states are not generated at stop 5 under Policy 1b in Figure 4. Note that automobiles 1 and 2 are not reloaded, and thus are not involved in reshuffling and must remain in their original slots.

Policy 1c is a simplified version of Policy 1b such that it generates a sparser space-state graph. The procedures for Step r under Policy 1c differ from those under Policy 1b in the following two aspects. First, in the above-described removal procedure in the case of dropping off an order at stop r, no reshuffling is conducted for the reloaded orders (i.e., reloaded orders are loaded back to their original slots). Second, after generating the vertices at stop r, fewer edges are added as follows: a directed edge is added to connect each vertex $\pi \in N_{r-1}$ at stop $r-1$ with each of the vertices in N_r^π at stop r (i.e., connect a vertex in N_{r-1} to each of its induced vertex only, rather than all vertices in N_r).

After a space-state graph is generated for a candidate loading policy, the cost of each edge is specified as the number of reloads involved, which is described in detail in Section 4.2.3. In the space-state graph there could be many paths between the source and sink vertices, each of which represents how the loading state evolves over space, i.e., gives a loading and unloading solution. As the loading/unloading cost is measured by the number of reloads, the path between the source and sink vertices with the minimum number of reloads should be selected. Thus, the auto-carrier loading optimization problem under Policy 1 (including 1a, 1b and 1c) is converted to the shortest path problem in the state-space graph, which can be solved very efficiently.

Under Policy 0, reloading is prohibited, which means the space-state graph generation procedure designed for Policy 1a can be modified (i.e., by avoiding the edges with reloads) to accommodate this restriction. Nonetheless, a much simpler heuristic can be used to solve the loading/unloading optimization problem under Policy 0. Note that under Policy 0, the objective is not to find an optimal path with the minimum number of reloads. Instead, the goal is to find at least a feasible path from the source vertex to the sink vertex whose cost is 0 (no reloads). The heuristic is described as follows:

At each stop r, check whether the unloading path is clear for every $\bar{k}$ in K_r^D, i.e., no orders outside of K_r^D are in the unloading path. Remove $\bar{k}$ only when the path is clear. For all permutations of orders in K_r^P, insert an order in K_r^P to each possible innermost considering applicable loading constraints. If the final stop can be reached without incurring any infeasibility, a feasible path is found.

As the auto-carrier route shown in Figure 4 is infeasible under Policy 0. Figure 5 shows the generated space-state graph for another auto-carrier route (different from the one in Figure 4) using the above heuristic. We can observe from this example that although a new automobile for loading can be inserted into multiple slots, it is advantageous to insert it to the inner most slot under Policy 0. This is because under Policy 0, all auto-carrier slots that are currently occupied and those beyond an occupied slot are unavailable. For instance, if an automobile (say order 1) is loaded to the exit slot of a four-slot carrier, i.e., resulting in a loading state of [1,-,-,-], then no additional automobiles can be loaded until order 1 is delivered. Always loading an automobile to the inner-most slot thus maximizes the chance for additional automobiles to be loaded feasibly later. As in Figure 4 for illustrative purposes, we assume that either K_r^P or K_r^D is a singleton set; the other is empty.

4.2.3. Specification of edge costs

For an edge $(\pi, \bar{\pi})$ connecting state $\pi \in N_{r-1}$ at stop $r-1$ to state $\bar{\pi} \in N_r$ at stop $r \in \{1, \ldots, |R|\}$, we need to compute the cost of this edge, namely the number of reloads needed to transition from π to $\bar{\pi}$. Essentially, we need to evaluate how the relative sequence of those onboard orders has changed from stop $r-1$ to stop r.

We start with the case of a single-level auto-carrier. We denote the sequence of onboard orders in state π with a tuple T_π. For instance, state [1,2,-,4] yields tuple (1,2,4). Similarly, another tuple $T_{\bar{\pi}}$ is generated from state $\bar{\pi}$. To compute the number of reloads needed to transition from π to $\bar{\pi}$, we compare the two tuples T_π and $T_{\bar{\pi}}$ order by order, however, in reverse sequence (namely from the innermost slot to the exit slot). If two orders in the same position of T_π and $T_{\bar{\pi}}$ are identical, then these orders do not need to be reloaded. If a difference is observed starting from a position, such as j, all unexamined orders in T_π including the order in position j are saved in a set K_π, representing all vehicles to be unloaded; similarly, all unexamined orders in $T_{\bar{\pi}}$ including the order in position j are saved in a set $K_{\bar{\pi}}$, representing all vehicles to be loaded. Then, the intersection of K_π and $K_{\bar{\pi}}$ represents all reloaded orders in π and $\bar{\pi}$, respectively. The edge cost is thus $|K_\pi \cap K_{\bar{\pi}}|$.

Edge cost computation is more complex for a two-level auto-carrier, due to two reasons. First, two levels share the same exit slot. Second, a vehicle may move across levels, thus creating an additional reload. For instance, Figure 6 shows a state change. If the above described reload computation method is applied to each level separately, one would arrive at the wrong conclusion that no reloads are needed. Nonetheless, the physical configuration of a two-level auto-carrier determines that one reload is required as an order moves from the lower level to the upper level. This is because if an order is moved from the lower level to the upper level, the exit slot must be emptied so that the loading

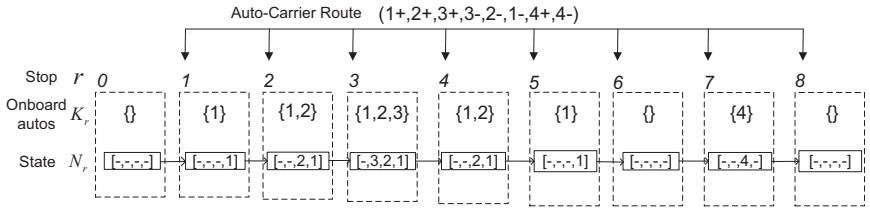

Figure 5. Space-state graph generation under Policy 0.

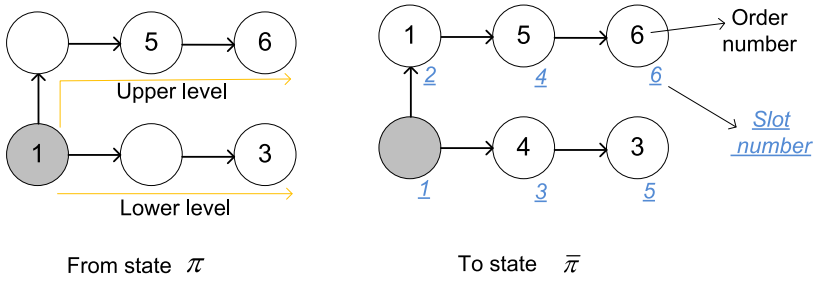

Figure 6. State change for a two-level auto-carrier.

platform of slot 2 (see Figure 1 for slot naming convention) is lowered to allow order loading to the upper level. In the case illustrated in Figure 6, order 1 must be first unloaded and then loaded to slot 2, thus creating one reload. Similarly, one reload is needed, if an order is moved from the upper level to the lower level.

We next describe how to compute the edge cost for a two-level auto-carrier. We use $T_\pi = (T_\pi^{up}, T_\pi^{lo})$ to represent the relative sequence of onboard orders in state π, where T_π^{up} is the tuple for the upper level and T_π^{lo} is for the lower level. Since the two levels intersect at the exit slot, if the exit slot is occupied by some order, then this order is included in both T_π^{up} and T_π^{lo}. In the same way, we use $T_{\bar{\pi}} = (T_{\bar{\pi}}^{up}, T_{\bar{\pi}}^{lo})$ to represent the relative sequence of orders in state $\bar{\pi}$. Then, we start with the upper level and compare two tuples T_π^{up} and $T_{\bar{\pi}}^{up}$ order by order in reverse sequence. If starting from position j, an order number difference is observed or the order under examination moves across levels, all unexamined orders in T_π^{up} including the order in position j are saved in set K_π^{up}, representing vehicles for unloading from the upper level. All unexamined orders in $T_{\bar{\pi}}^{up}$ including the order in position j are saved in set $K_{\bar{\pi}}^{up}$, representing vehicles for loading to the upper level. In the same way, we obtain K_π^{lo} and $K_{\bar{\pi}}^{lo}$, representing unloaded orders from and loaded orders to the lower level. The unions $K_\pi^{up} \cup K_\pi^{lo}$ and $K_{\bar{\pi}}^{up} \cup K_{\bar{\pi}}^{lo}$ thus represent reloaded orders in π and $\bar{\pi}$, respectively. The edge cost is thus $\left|(K_\pi^{up} \cup K_\pi^{lo}) \cap (K_{\bar{\pi}}^{up} \cup K_{\bar{\pi}}^{lo})\right|$.

4.2.4. A labeling approach for concurrent graph generation and edge cost computation

The above presented loading optimization method for Policy 1 (including 1a, 1b and 1c) first builds the space-state graph and then specifies the edge cost, before finding the optimal loading path. It has a few deficiencies. First, the built space-state graph can be very dense, especially under Policies 1a and 1b, because under these policies, at each stop $r = 1, \ldots, |R| - 1$, a directed edge is added to connect each vertex in N_{r-1} with each vertex in N_r, creating many edges between the two stops. Second, many loading paths with the same optimal cost may be found. Next, we intend to address the above deficiencies by proposing a labeling procedure where we build the space-state graph and search for the optimal path concurrently. The distinction of the labeling procedure is to optimally select a single vertex in N_{r-1} to connect with a given vertex in N_r, rather than connecting all vertices in N_{r-1} to the given vertex in N_r. Here, the label of a vertex π is defined as the minimum number of reloads needed to reach this vertex from the source vertex, and denoted as L_π.

The labeling procedure is described as follows. At Stop 0, attach a label with a value of 0 i.e., $L_\pi = 0$, to the source vertex and all generated vertices in N_1. At Stop $r = 1, \ldots, |R|$, an initial label of value ∞ is attached to each generated vertex in N_r. Section 4.2.2 provides more details on how vertices in N_r are generated. For each vertex $\bar{\pi} \in N_r$, consider all vertices in N_{r-1} in a non-decreasing order by their label values. Depending on the label of each vertex $\pi \in N_{r-1}$, there are two cases:

1. If the label of vertex π is smaller than the label of vertex $\bar{\pi}$, i.e., $L_\pi < L_{\bar{\pi}}$, calculate the cost of edge $(\pi, \bar{\pi})$, denoted as $C(\pi, \bar{\pi})$, using the procedure described in Section 4.2.3. If $L_\pi + C(\pi, \bar{\pi}) < L_{\bar{\pi}}$, then update the label of vertex $\bar{\pi}$ as $L_{\bar{\pi}} = L_\pi + C(\pi, \bar{\pi})$.
2. If $L_\pi \geq L_{\bar{\pi}}$, then the label of $\bar{\pi}$ cannot be improved by a path through vertex π or any remaining vertices with equally or larger label values. Hence, the remaining sorted vertices in N_{r-1} do not need to be considered, and any edge between π and $\bar{\pi}$, as well as any edge between any remaining vertices in N_{r-1} and $\bar{\pi}$ do not need to be considered.

In the above process, after vertex $\bar{\pi} \in N_r$ is considered but before considering the next vertex in N_r, add the edge $(\pi^*, \bar{\pi})$ to the space-state graph, where π^* is the vertex in

N_{r-1} that gives the final value of the label for $\bar{\pi}$, i.e., $L_{\bar{\pi}} = L_{\pi^*} + C(\pi^*, \bar{\pi})$.

To summarize the labeling procedure, at each stop r, for each vertex $\bar{\pi} \in N_r$, its label is updated if going through vertex π gives a shorter path from the source vertex to $\bar{\pi}$, i.e., if $L_\pi + C(\pi, \bar{\pi}) < L_{\bar{\pi}}$. Otherwise, going through π or any remaining vertices in N_{r-1} will not create a shorter path, and hence, those vertices should be skipped. The labeling procedure creates exactly one direct edge that connects an optimally selected vertex in N_{r-1} and $\bar{\pi}$. Furthermore, after this process is completed, it creates a single path from the source vertex to the sink vertex, which is the shortest path with the total cost of the path equal to the label of the sink vertex.

5. Computational experiments

We first lay out the framework for evaluating and comparing different types of loading policies. It is inadequate to only investigate how different loading policies perform in the context of the loading optimization problem for a given vehicle route for the following reason. Suppose that we are given a vehicle route, and it turns out that this given route is infeasible under Policy 0, and this can be checked nearly instantaneously. However, for this same route, it takes 5 seconds to find that the minimum number of reloads is two under Policy 1a. In such a case, it is difficult to determine which loading policy, Policy 0 or Policy 1a, is better. If this given route is highly desirable, i.e., with a high probability of being selected for implementation, then Policy 1a outperforms Policy 0, as it retains the feasibility. If this given route has almost no chance of being selected, adopting Policy 0 is likely more favorable. Unfortunately, the "desirability" of a given route is known only after the overall shipping optimization problem is solved. This example implies that we must evaluate how a loading policy impacts the total cost in the context of the automobile shipping optimization problem which involves many possible routes, instead of just a single route. To save space, we describe the automobile shipping optimization problem with a given set of routes in the online Appendix A, with the purpose of evaluating the comparative advantages of various loading policies. As clarified earlier, it is beyond the scope of this study to jointly optimize auto-carrier routes and loading decisions.

Next, we describe how loading test instances are generated. Then, we present the results from the benchmark analysis assuming a single auto-carrier, followed by the sensitivity analyses of some key input parameters. Finally, we evaluate the benefits of the proposed auto-carrier loading policies through a comprehensive analysis involving multiple types of auto-carriers.

5.1. Benchmark test instances

Due to the lack of publicly available test instances (Sun *et al.*, 2021), test instances are created for an anonymized automobile shipping company based in central Florida, whose service area consists of those Florida metropolitan areas with a population of over 100,000 in the 2020 U.S. census. All automobile shipping orders have their own pickup and drop-off locations in those metropolitan areas. With this setting, we intend to study the shipping of pre-owned vehicles rather than finished vehicles, because the latter ones typically have the same pickup location (i.e., automobile manufacturer). Figure 7 shows the population centroids of all involved metropolitan areas as well as the auto-carrier depot.

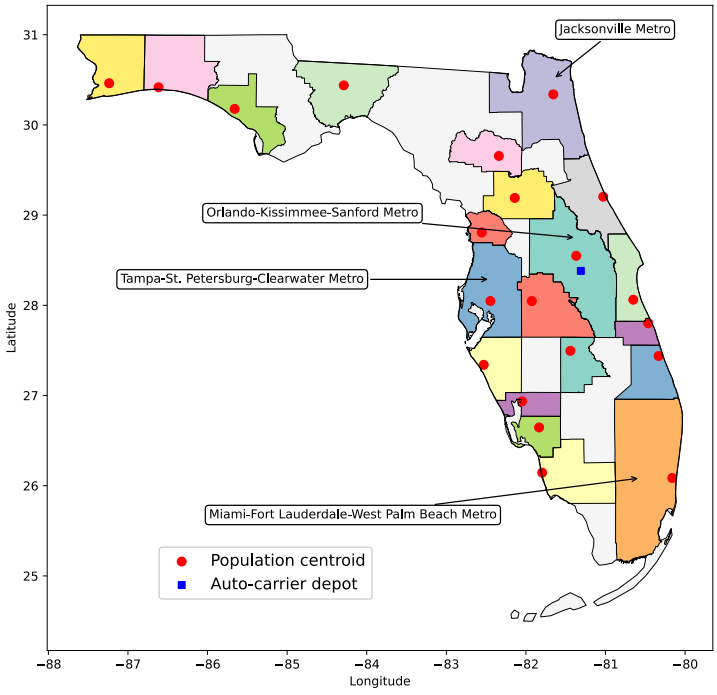

Figure 7. Metro areas with automobile pickup and drop-off locations in the benchmark.

When generating an automobile for shipping, two different metropolitan areas at least 50 miles apart are drawn through weighted random sampling with population as the weight to represent the areas of origin and destination, respectively. Then, the pickup and drop-off locations that are specific for each automobile are randomly selected in those selected metropolitan areas. For simplicity, when estimating the distance between locations in different metropolitan areas, the centroid of each metropolitan area is used. To be realistic, the distance between two centroids is obtained from the Distance Matrix API of Google Maps (Google Maps, 2022). The distance between two locations in the same area is approximated as 20 miles for convenience. After generating the pickup and drop-off locations for an automobile, we further draw one of the three types for this automobile with the following sampling weights: 1/6 for Type 1, 1/3 for Type 2, and 1/2 for Type 3.

The six-slot auto-carrier shown in Figure 1 is considered as the benchmark auto-carrier. Similar single-car and pairwise constraints to those in Agbegha *et al.* (1998) are considered: Type 2 or Type 3 vehicles cannot be assigned to slot 6; Type 1 vehicles cannot be assigned to slot 1; if a Type 2 or 3 is in slot 2 (or slot 4), another vehicle of Type 3 cannot be in slot 4 (or slot 2).

In each random experiment, a set of six automobiles are first generated as described earlier. The benchmark auto-carrier serves these automobiles in a single auto-carrier route. Given these automobiles, a permutation of the six pickup and drop-off locations where the pickup precedes drop-off for each automobile is taken as an auto-carrier route. For the same set of automobiles, a total of 250 random auto-carrier routes are generated. When a different set of six automobiles are randomly selected, another 250 random routes are generated in a new experiment. There are in total 100 such experiments in the benchmark analysis, and they are numbered from bm-exp-1 to bm-exp-100, where "bm" stands for benchmark and "exp" stands for experiment.

The total shipping cost is measured in auto-carrier miles, as defined in (5) in the online Appendix A. Each reloading operation is assumed to be equivalent to 50 auto-carrier miles in the benchmark. All the analyses are implemented in Python 3.10 and the integer program was solved by CPLEX v12.10. All programs run on a personal computer (Intel Core i7-8700 CPU 3.20 GHz, 32 GB RAM).

5.2. Benchmark analysis

In all benchmark experiments (bm-exp-1 to bm-exp-100), the same six-slot auto-carrier is used, while in different experiments, a different set of automobile shipping orders is used. In each benchmark experiment, the automobile shipping optimization problem defined in the online Appendix A becomes trivial to solve, because only one auto-carrier is involved, and all the 250 randomly generated routes cover the same set of automobiles. For any given policy, the route with the least total cost among all the feasible (250 or less) routes is selected as the optimal solution. To illustrate, we take the first benchmark experiment, namely bm-exp-1, as an example. Under Policy 0, the optimal route found is (2+, 1+, 5+, 5-, 6+, 3+, 6-, 3-, 2-, 4+, 1-, 4-). Figure 8(a) and (b) shows the resulting space-state graph for this route. Under Policy 1a, the optimal route found is (2+, 5+, 6+, 6-, 1+, 2-, 4+, 5-, 1-, 3+, 4-, 3). Figure 8(b) shows the resulting space-state graph for this route. The highlighted path (in red) in the space-state graph in each figure is the optimum loading/unloading plan. The comparison shows that the total cost is 1849.19 (unit: auto-carrier miles or simply miles) under Policy 1a, significantly lower than the total cost 2416.06 under Policy 0, even though one reload is included as part of the total cost under Policy 1a. However, the substantial cost savings are achieved with extensive computation efforts, as visualized by the much denser space-state graph under Policy 1a.

We then show the total computation time in each experiment and the resulting total cost under Policies 0 and 1a for all 100 benchmark experiments in Figure 9. We find that the solution time in every experiment under Policy 0 is close to 0, whereas the solution time in every experiment under Policy 1a is around 100 seconds. As expected, the majority of the computation time is spent on generating the dense space-state graphs. In 52% of the instances, Policy 1a yields a lower total cost than Policy 0, while they result in the same total cost in the remaining instances. In the cases where the two policies have the same total cost, the optimum number of reloads under Policy 1a is zero even though reloading is allowed. The average cost savings brought by Policy 1a as compared with Policy 0 in each instance are 88 miles, while an additional computation time of about 100 seconds per experiment is needed for Policy 1a on average.

The above analyses indicate that allowing reloading can generate significant cost savings at the expense of additional computation time. Next, all 100 benchmark experiments (bm-exp-1 to bm-exp-100) are re-conducted under the proposed policy variants, namely Policies 1b and 1c. Results are shown in Figures 10 and 11 for Policies 1b and 1c, respectively. Figure 10 indicates that Policy 1b yields the same solution as Policy 1a in all 100 experiments, although Policy 1b needs only 23.5% of the solution time used by Policy 1a when the labeling algorithm described in Section 3.3.4 is implemented. Figure 11 indicates that with a minimum solution time (less than 5 seconds), Policy 1c yields a smaller total cost than Policy 0 (on average 11.2% smaller) for 50% of the 100 experiments; for the other half, Policy 1c has the same total cost as Policy 0. As discussed earlier, Policy 1a yields cost savings for 52% of the 100 experiments, compared with Policy 0. This implies that for two of the 100 experiments, Policy 1a yields cost savings while Policy 1c does not. When Policy 1c is directly compared with Policy 1a, Policy 1c yields the same total cost as Policy 1a for 89 experiments. In summary, we conclude that Policy 1b can achieve the same solution quality for all the 100 randomly generated experiments as Policy 1a, while requiring less than 25% of the solution time. Policy 1c requires little computation time while compared to Policy 0, it can reduce the total

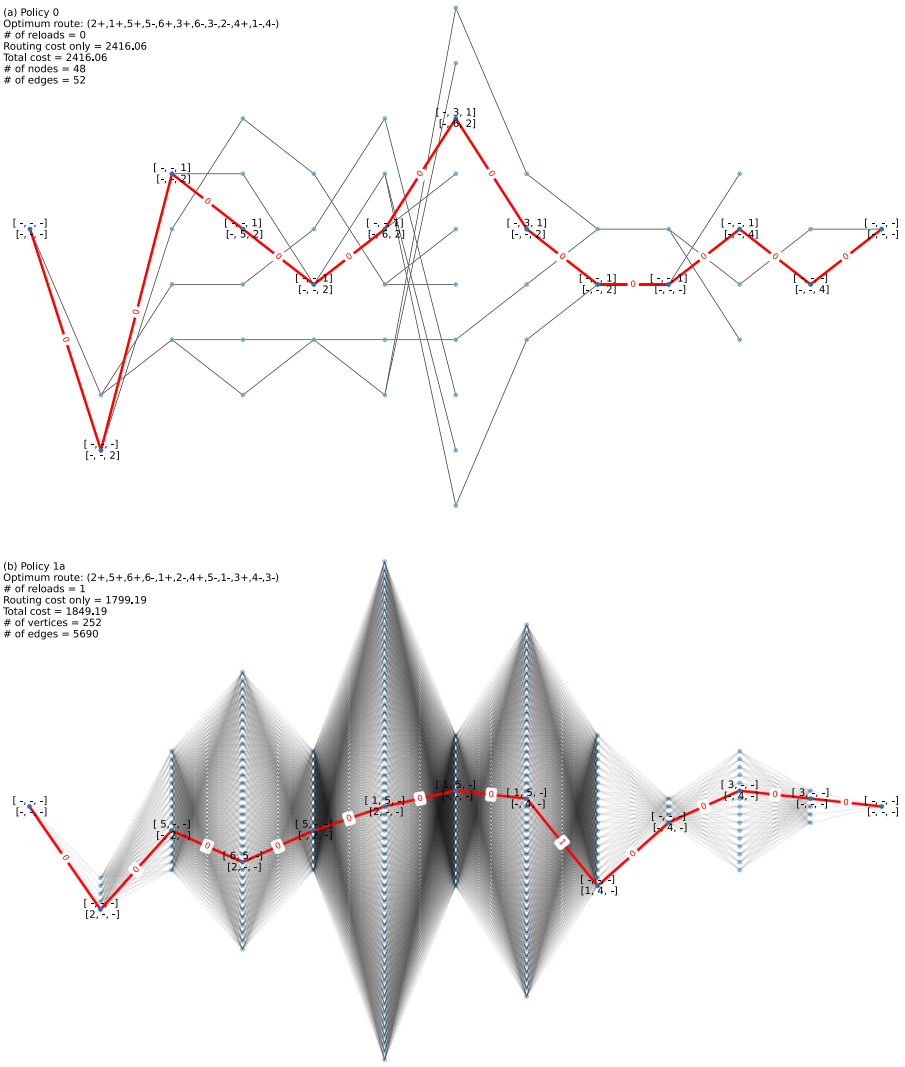

Figure 8. Comparisons of optimization results under Policies 0 and 1a in bm-exp-1.

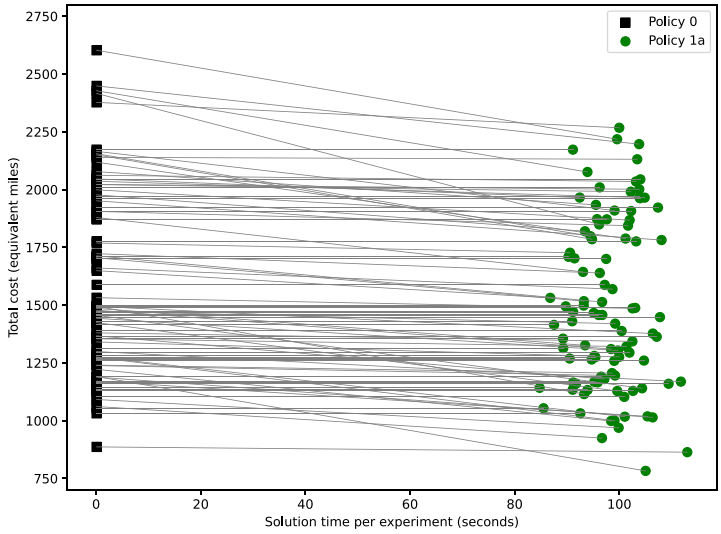

Figure 9. Trade-offs of total cost and solution time in 100 benchmark experiments.

cost by approximately 10% on average for instances where reloading is necessary.

We note that although for all the 100 benchmark experiments conducted, Policy 1b yields the same solution as Policy 1a, Policy 1b could yield suboptimal solutions in rare cases. For instance, Figure 12 compares the space-state graphs generated under Policies 1a and 1b for a four-slot auto-carrier covering a given route (2+, 3+, 4+, 1+, 3-, 2-, 1-, 4-). Here a

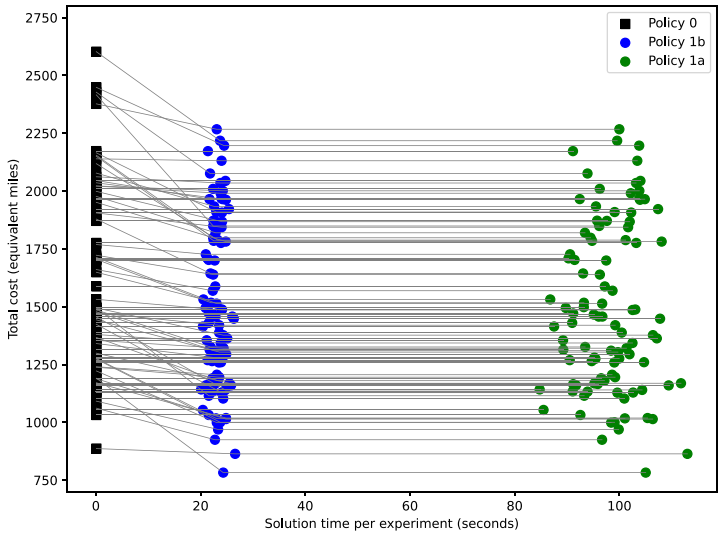

Figure 10. Performance of Policy 1b in solving 100 benchmark instances.

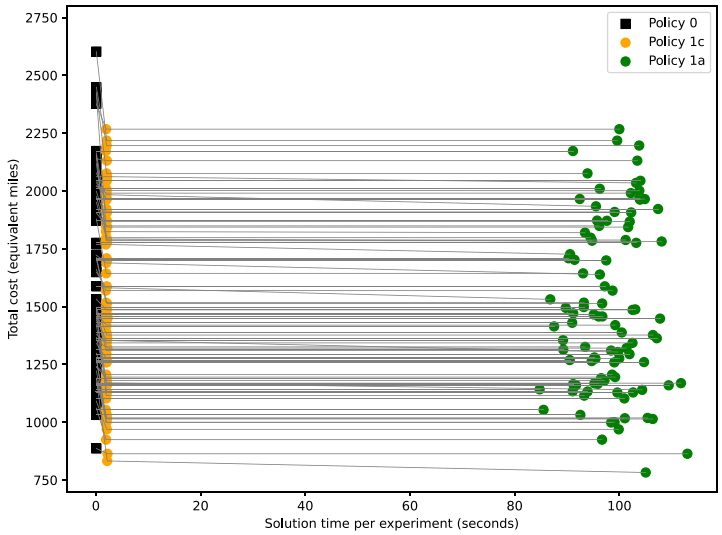

Figure 11. Performance of Policy 1c in solving 100 benchmark instances.

four-slot auto-carrier is used for ease of visualization. The optimal loading plan under Policy 1a is not generated under Policy 1b, because Policy 1b misses some possible loading states, evidenced by fewer vertices in its space-state graph. Note that here for comparison purposes, the labeling algorithm described in Section 3.3.4 is not used under Policy 1b.

To further compare different policy variants for the same auto-carrier routes, we next generate 3000 random routes for the benchmark auto-carrier covering a given set of automobile shipping orders, and compare the minimum number of reloads for a given route under each policy. We find the following: (i) for 998 routes (accounting for 33.2%), Policies 1a, 1b, and 1c all yield zero reloads, which means this route is feasible under Policy 0 as well; (ii) for 2516 routes (83.9%), Policy 1c can achieve the same number of reloads as Policy 1a; and (iii) for 2998 routes (99.9%), Policies 1b and 1a yield the same solution. In cases where Policy 1c yields suboptimal solutions, the obtained number of reloads can be substantially larger than the optimum. For instance, in the worst case the minimum number of reloads obtained under Policy 1c is five times the optimum under Policy 1a. By contrast, the minimum number of reloads obtained under Policy 1b is at most 33.3% larger than the optimum. To summarize, the numerical results indicate that Policy 1b yields optimal solutions for virtually all routes (99.9%) and Policy 1c is optimal for most of the routes (83.9%), although in some cases, Policy 1c can yield clearly inferior solutions.

5.3. Sensitivity analyses

In the above benchmark instances, we use a single value of the reloading cost coefficient (i.e., 50 miles per reload) and assume that an automobile can be shipped from any Florida metropolitan area to any other, while the sampling probability depends on its population. We next explore how the comparison results would vary as the reloading cost coefficient changes, and investigate how the comparative advantage of a proposed loading policy (Policy 1b) would change when automobile orders are generated in a different way. In the following, results averaged over 100 random experiments (bm-exp-1 to bm-exp-100) are reported.

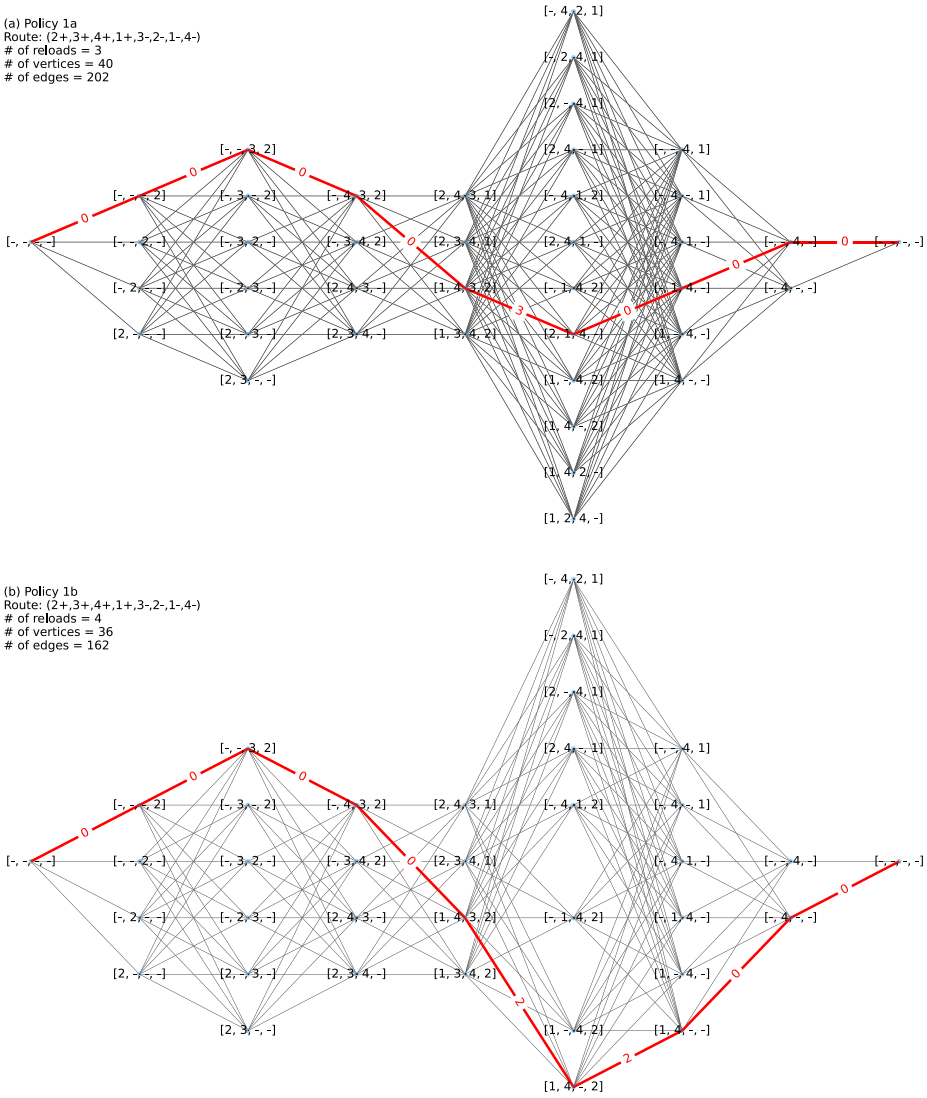

Figure 12. Suboptimality of Policy 1b.

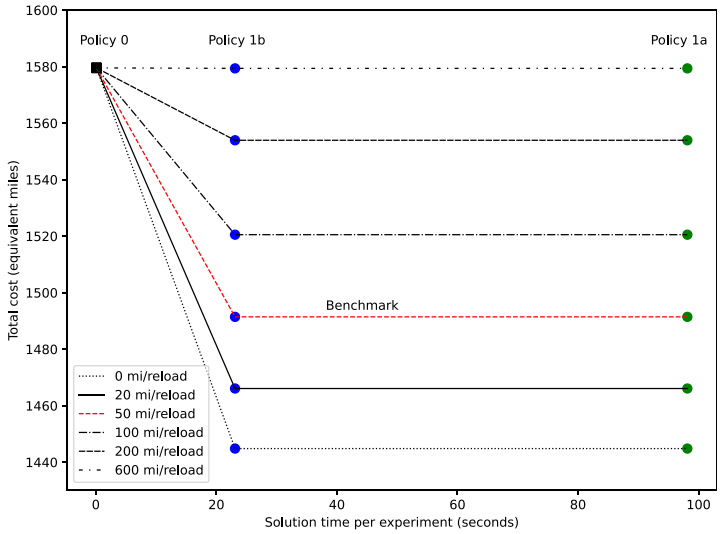

Figure 13. Effect of reloading cost coefficient.

Figure 13 shows the impact of reloading cost coefficient. As the reloading cost coefficient increases (i.e., reloading is increasingly expensive), the advantage of Policies 1a and 1b over Policy 0 decreases. The advantage diminishes to zero only when the reloading cost becomes unpractical, e.g., equivalent to 600 auto-carrier miles per reload. This implies

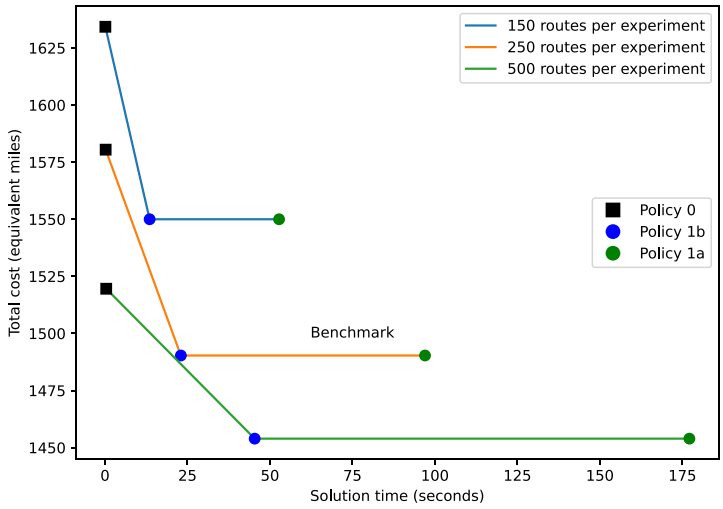

Figure 14. Effect of the number of auto-carrier routes per experiment.

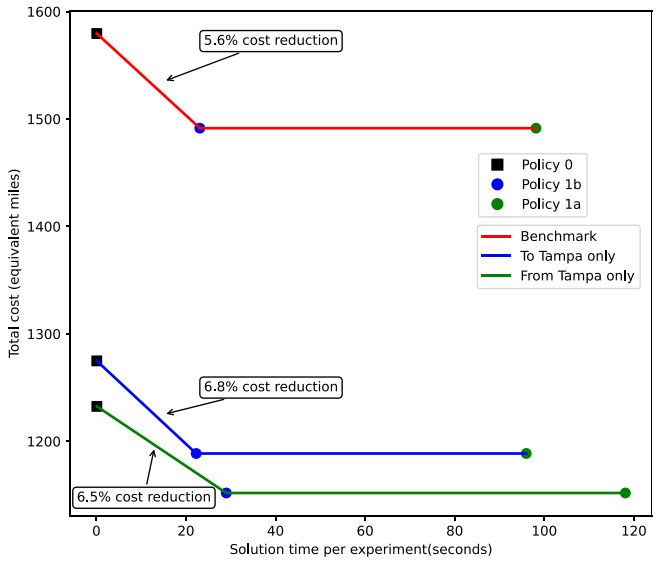

Figure 15. Effect of the spatial distribution of pickup and drop-off locations.

that in practical cases, the cost savings achieved by allowing reloading are quite robust. Between Policies 1a and 1b, the achieved cost savings are very comparable, while the solution time under Policy 1b is slightly over 20% of the time under Policy 1a. Figure 14 further shows the solution time and total cost for each loading policy when the number of randomly generated routes in the benchmark experiments varies from 150 to 500. Across all the three cases, it is clear that Policy 1b can result in significant cost savings in relative to Policy 0; the solution quality of Policy 1b is very similar to that of Policy 1a, whereas its solution time is significantly less than Policy 1a.

We next consider restricted sets of pickup and drop-off locations. For instance, automobiles are shipped only from the Tampa–St. Petersburg–Clearwater (or Tampa for brevity) metropolitan area to other areas, referred to as "From Tampa only." Alternatively, automobiles flow from non-Tampa areas in Florida to Tampa only, referred to as "To Tampa only." Figure 15 thus shows how the advantage of Policy 1b varies in each case. Overall, we find when automobiles for shipping tend to have clustered origins or destinations, larger percentage savings can be expected if Policy 1b is adopted, relative to Policy 0.

5.4. A full-scale analysis

In the benchmark analysis reported in Section 5.2, only a single auto-carrier is considered, which always covers the same set of automobiles in one benchmark experiment. In addition, only certain Florida metropolitan areas are involved. In this full-scale analysis, multiple types of auto-carriers are employed to transport automobiles in a south-eastern region of the U.S. consisting of five states, namely Alabama, Georgia, North Carolina, South Carolina, and Florida. The auto-carrier depot is now in central Georgia. A different population threshold of 300,000 instead of 100,000 is used to filter metropolitan areas in this region, as shown in Figure 16.

There are in total 10 auto-carriers of two types in the fleet as follows. The first five are homogeneous and have a single level; the other five are also homogeneous and have two levels. A single-level auto-carrier has five slots (shown in Figure 1) with a variable routing cost of $2.0 per mile and $50 per reload. For a single-level auto-carrier, there is a single loading constraint: a Type 3 automobile cannot be assigned to the exit slot (slot 1) or the innermost slot (numbered slot 9 in Figure 1). A two-level auto-carrier is the same as the benchmark auto-carrier described earlier, with a variable routing cost of $3.0 per mile and $100 per reload. Loading constraints associated with the benchmark auto-carrier are provided in Section 4.1.

A set of 45 randomly generated automobile orders to be covered by this fleet is shown in Figure 16. The numbers in the parenthesis for each metropolitan area represent the numbers of pickups and drop-offs, respectively. The automobile type distribution is the same as in the benchmark analysis, namely 1/6 (Type 1), 1/3 (Type 2), and 1/2 (Type 3). While in the benchmark analysis the auto-carrier must be fully reloaded, now this requirement is relaxed as follows: a single-level carrier can cover 3, 4, or 5 orders, whereas a

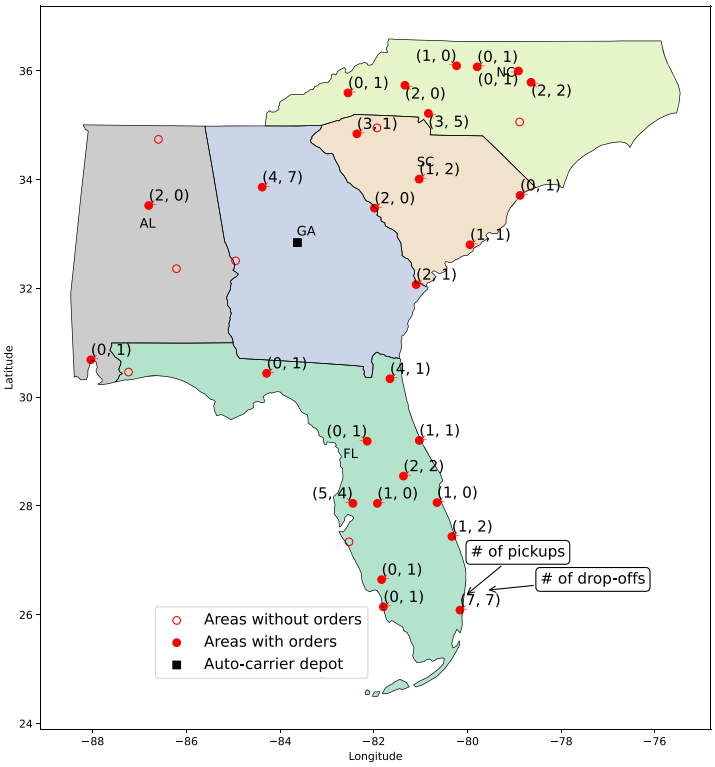

Figure 16. Metro areas with pickup and drop-off locations of the 45 automobile shipping orders.

Table 1. Trade-off of total cost and computation time (45 orders, 50 routes per given set).

Policy	*# of Auto-carriers used*	*Total cost ($)*	*Routing cost ($)*	*Reloading cost ($)*	*Computation time (s)*
Policy 0	10	59,376.4	59,376.4	0	15
Policy 1b	9	54,263.4	53,313.4	950.0	4677
Policy 1c	9	54,363.4	53,313.4	1050.0	381

two-level carrier can cover 4, 5, or 6 orders. We next generate auto-carrier routes for each of the two auto-carrier types as follows. First, we randomly draw an allowable number of orders (e.g., 3 orders for a five-slot auto-carrier) with equal probability from the 45 given orders. Then, 50 random auto-carrier routes are generated covering the sampled set of orders. This is repeated 800 times, generating a total of 40,000 routes for each auto-carrier type. Thus, in total, 80,000 auto-carrier routes are generated. From a probabilistic point of view, it is very unlikely that any two of these routes are identical because first, it is unlikely that any two draws yield the same subset of orders, and second, for the same subset of orders, it is unlikely that any two of the 50 generated routes are identical.

The revised automobile shipping formulation for homogeneous auto-carriers of the same type (presented in the online Appendix A) is solved under Policies 0, 1b, and 1c. Optimization results are shown in Table 1. First, all 10 auto-carriers are needed to cover 45 given orders under Policy 0, whereas under Policies 1b and 1c, only nine are needed, five of which are single-level auto-carriers. The total solution time under Policy 0 is 15 seconds, and the solution time under Policy 1b is approximately 1.3 hours. Relative to Policy 0, Policy 1b can reduce the total cost from $59,376.4 to $54,263.4. The benefit of Policy 1b is thus apparent: with a computation time of about 1.3 hours, a cost saving of $5,113.0 (or 8.6%) is achieved.

Policy 1c yields a slightly higher total cost than Policy 1b, with a much smaller solution time. For all nine auto-carriers, the same auto-carrier routes are used under Policy 1c in comparison with Policy 1b. Therefore, Policies 1b and 1c have the same routing cost as shown in Table 1. For five single-level auto-carriers, Policy 1c yields only two more reloads than Policy 1b, as shown in the online Appendix C, whereas for four two-level auto-carriers, Policy 1c yields the same number of reloads as Policy 1b, which is not shown due to space limitations.

We further present a slightly modified analysis where we reduce the number of random routes for a given set of orders from 50 to 25. Specifically, for each set of orders, we randomly draw 25 routes from the 50 routes generated above. The updated optimization results are shown in Table 2. As expected, the total cost under each policy increases, because now fewer routes cover a given set of orders. Proposed policy variants yield even larger cost savings. For instance, now Policy 1b yields a cost saving of $5706.1 (or 9.3%), instead of $5113.0 (or 8.6%). Notably, the computation time needed to generate the said cost saving decreases by more than 50%, which provides more convincing evidence for the advantage of Policy 1b. We next change

Table 2. Trade-off of total cost and computation time (45 orders, 25 routes per given set).

Policy	*# of Auto-carriers used*	*Total cost ($)*	*Routing cost ($)*	*Reloading cost ($)*	*Computation time (s)*
Policy 0	10	61,635.1	61,635.2	0	7
Policy 1b	10	55,929.1	55,279.1	650.0	1866
Policy 1c	10	55,979.1	55,279.1	700.0	167

Table 3. Trade-off of total cost and computation time (90 orders, 25 routes per given set).

Policy	*# of Auto-carriers used*	*Total cost ($)*	*Routing cost ($)*	*Reloading cost ($)*	*Computation time (s)*
Policy 0	21	95,789.2	95,789.2	0	7
Policy 1b	20	80,280.6	78,830.6	1450	1681
Policy 1c	20	80,454.9	78,904.9	1550	177

the number of automobile orders to 90, the number of auto-carriers of each type to 20, while keeping the number of sets of routes at 800 and the number of random routes in each set at 25. The optimization results are shown in Table 3. The relative advantages of Policies 1b and 1c are consistent with those observed in Tables 1 and 2.

Based on the analyses presented above, we conclude that both proposed policy variants, Policies 1b and 1c, have achieved a more desirable balance between total cost and computation time than Policies 0 and 1a. As neither of Policies 1b and 1c dominates the other, we recommend both policies be considered depending on the specific needs of an auto-carrier company. For instance, if a company is constrained by computational power, Policy 1c can be adopted; if a company seeks to achieve the highest cost efficiency possible, while being not much constrained by computational power, Policy 1b is preferable.

6. Conclusions

This study is motivated by the lack of understanding of how the degree to which automobile reloading is allowed in automobile shipping influences the resulting computational burden and total cost. We therefore develop a space-state network-based solution approach for auto-carrier loading/unloading optimization under various reloading policies, when auto-carrier routes are given. Benchmark analysis results indicate that the two proposed loading policies (i.e., Policies 1b and 1c) can yield a desirable compromise between cost efficiency and computation time. Specifically, Policy 1b yields optimal solutions for virtually all routes (over 99%) and Policy 1c is optimal for most of the routes (over 80%). Policy 1b requires less than 25% of the solution time than a brute-force algorithm (i.e., Policy 1a) when reloading is allowed. Policy 1c requires even less time, although the disadvantage of Policy 1c is that in certain cases, it can yield clearly inferior solutions. Through sensitivity analyses, we also find that the cost savings achieved with Policy 1b are quite robust when practical values of reloading costs are adopted. In addition, in contrast with uniform distributions of origins and destinations, when pickup or drop-off locations are more clustered (i.e., automobiles are moving in a similar direction), the advantage of Policy 1b is even more significant. Lastly, findings from a full-scale analysis involving multiple types of auto-carriers validate the substantial benefits of proposed loading policies, as they may yield more than a cost saving of $5000 (or 8.6%) with around 1.3 hours or less in solution time.

In this article, we have focused on the optimization of auto-carrier loading/unloading plans for given auto-carrier routes. A natural extension is to employ sophisticated algorithms, such as column generation, to generate in a more intelligent fashion auto-carrier routes that satisfy various practical routing constraints (such as time window requirements), while considering the impact of routing decisions on auto-carrier loading/unloading operations in real time. That way, routing and loading decisions are truly integrated.

It should also be noted that there are many studies based on the notion of space-time prism for measuring space-time accessibility, such as Tong *et al.* (2015) and Qin and Liao (2021). Specifically, a strategy for reducing the trip chaining space proposed by Qin and Liao (2021) may inspire new policies for finding the most efficient auto-carrier loading path on a space-state graph.

As indicated in Section 4.1, no publicly available problem instances are available for testing the proposed loading policies. Therefore, many instances are generated in this study and the way to generate such instances is well described. Additional efforts are clearly needed to create more standardized test instances that can be used in future studies, preferably with inputs from the automobile shipping industry.

Disclosure statement

No competing interests are declared by the authors.

Funding

Division of Human Resource Development; National Science Foundation.

Notes on contributors

Sajeeb Kumar Kirtonia was a PhD student while working on this paper. He defended his doctoral dissertation in November 2023. Prior to his doctoral study at Florida State University, he received his bachelor's degree from Shahjalal University of Science and Technology, Bangladesh.

Yanshuo Sun is an assistant professor of industrial engineering at the FAMU-FSU College of Engineering. He specializes in transportation systems optimization. He received his PhD in civil engineering from the University of Maryland, College Park.

Zhi-Long Chen is currently Orkand Corporation Professor of Management Science, and Professor of Operations Management at the Smith School at the University of Maryland, College Park. He does

research and teaches in the areas of operations management, supply chain management and management science.

ORCID

Sajeeb Kumar Kirtonia http://orcid.org/0000-0003-4825-6454
Yanshuo Sun http://orcid.org/0000-0003-2943-4323
Zhi-Long Chen http://orcid.org/0000-0002-9960-0465

Data availability statement

The authors confirm that the data supporting the findings of this study are available within the article

References

Agbegha, G.Y., Ballou, R.H. and Mathur, K. (1998) Optimizing auto-carrier loading. *Transportation Science*, **32**(2), 174–188.

Battarra, M., Erdoğan, G., Laporte, G. and Vigo, D. (2010) The traveling salesman problem with pickups, deliveries, and handling costs. *Transportation Science*, **44**(3), 383–399.

Bonassa, A.C., da Cunha, C.B. and Isler, C.A. (2023) A multi-start local search heuristic for the multi-period auto-carrier loading and transportation problem in Brazil. *European Journal of Operational Research*, **307**(1), 193–211.

Bortfeldt, A. and Yi, J. (2020) The split delivery vehicle routing problem with three-dimensional loading constraints. *European Journal of Operational Research*, **282**(2), 545–558.

Bukchin, J. and Sarin, S.C. (2004) A cyclic policy for the loading of multiple products on a vehicle with different compartment sizes. *IIE Transactions*, **36**(7), 641–653.

Garey, M.R. and Johnson, D.S. (1979) *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman and Company, New York, NY.

Chen, F. and Wang, Y. (2020) Downward compatible loading optimization with inter-set cost in automobile outbound logistics. *European Journal of Operational Research*, **287**(1), 106–118.

Chen, H.K. (2016) Sequential auto carrier loading problem. *Transportation Research Record*, **2548**(1), 53–61.

Dell'Amico, M., Falavigna, S. and Iori, M. (2015) Optimization of a real-world auto-carrier transportation problem. *Transportation Science*, **49**(2), 402–419.

Google Maps (2022) Distance matrix API overview. https://developers.google.com/maps/documentation/distance-matrix/overview. (accessed November 1, 2022).

Juárez Pérez, M.A., Pérez Loaiza, R.E., Quintero Flores, P.M., Atriano Ponce, O., Flores Lee, J., Kim, B.I. and Johnson, A.L. (2013) A two-dimensional bin packing problem with size changeable items for the production of wind turbine flanges in the open die forging industry. *IIE Transactions*, **45**(12), 1332–1344.

Lin, C.H. (2010) An exact solving approach to the auto-carrier loading problem. *Journal of Society for Transportation and Traffic Studies (JSTS)*, **1**(1), 93–107.

Mahmoudi, M. and Zhou, X. (2016) Finding optimal solutions for vehicle routing problem with pickup and delivery services with time windows: A dynamic programming approach based on state–space–time network representations. *Transportation Research Part B: Methodological*, **89**, 19–42.

Petersen, H.L. and Madsen, O.B. (2009) The double travelling salesman problem with multiple stacks–formulation and heuristic solution approaches. *European Journal of Operational Research*, **198**(1), 139–147.

Qin, J. and Liao, F. (2021) Space–time prism in multimodal supernetwork-Part 1: Methodology. *Communications in Transportation Research*, **1**, 100016.

Reil, S., Bortfeldt, A. and Mönch, L. (2018) Heuristics for vehicle routing problems with backhauls, time windows, and 3D loading constraints. *European Journal of Operational Research*, **266**(3), 877–894.

Statista.com (2022) Light vehicle retail sales in the United States from 1976 to 2020. https://www.statista.com/statistics/199983/us-vehicle-sales-since-1951/. (accessed 1 November 2022).

Sun, Y., Kirtonia, S. and Chen, Z.-L. (2021) A survey of finished vehicle distribution and related problems from an optimization perspective. *Transportation Research Part E: Logistics and Transportation Review*, **149**, 102302.

Tadei, R., Perboli, G. and Della Croce, F. (2002) A heuristic algorithm for the auto-carrier transportation problem. *Transportation Science*, **36**(1), 55–62.

Tadumadze, G. and Emde, S. (2022) Loading and scheduling outbound trucks at a dispatch warehouse. *IISE Transactions*, **54**(8), 770–784.

Tong, L., Zhou, X. and Miller, H.J. (2015) Transportation network design for maximizing space–time accessibility. *Transportation Research Part B: Methodological*, **81**, 555–576.

Veenstra, M., Roodbergen, K.J., Vis, I.F. and Coelho, L.C. (2017) The pickup and delivery traveling salesman problem with handling costs. *European Journal of Operational Research*, **257**(1), 118–132.

Wang, Y., Chen, F. and Chen, Z.-L. (2018) Pickup and delivery of automobiles from warehouses to dealers. *Transportation Research Part B: Methodological*, **117**, 412–430.