Recursive Partitioning and Batching for Network Design with Service Time Guarantees at Massive Scale

Myungeun Eom Alan Erera Alejandro Toriello
H. Milton Stewart School of Industrial and Systems Engineering
Georgia Institute of Technology
Atlanta, Georgia, USA 30332

November 18, 2025

Abstract

Motivated by the parcel delivery industry, we study a network design problem with service time guarantees at industrial scale. This tactical service network design problem determines primary paths and delivery schedules for packages to minimize transportation and handling costs while ensuring committed service times. To construct a solution for a real-world instance with over 1,000 nodes, one million arcs, and 40,000 commodities, we propose a recursive graph partitioning and batching method. This method partitions the network into smaller regions and solves the problem for each region, considering only commodities whose origin and destination are in the same region. For commodities crossing regions, we first determine an appropriate sub-network, then solve a restricted model on this sub-network. To handle the large number of commodities, we divide these into batches based on schedule slack (intuitively, how flexibly a commodity can be scheduled) and solve the problem sequentially over each batch. Finally, we reoptimize commodities in low-utilization trailers. We demonstrate the scalability and efficiency of our approach through computational studies on real-world instances from an industry partner. Our method finds high-quality solutions after several hours of computing time, while a commercial solver is unable to even build a model for a much smaller instance.

1 Introduction

Service network design is a classic optimization problem that has been increasingly used for the tactical design of intercity freight transportation operations (Bakir et al. 2021, Crainic 2000, Hewitt and Lehuédé 2023). Package carriers, which specialize in parcel shipments that are small relative to vehicle capacity, face challenging problems in service network design. Package carriers have grown in importance with the growth of e-commerce supply chains. For example, UPS reported $91 billion in revenue from its US and international small package operations in 2023, handling 22 million pieces of packages per day (UPS 2023). To efficiently handle a high volume of small packages, these carriers typically adopt a *hub-and-spoke* network structure. In this structure, hub terminals serve as major consolidation and transfer points, while spoke terminals only serve local pickup and delivery. In practice, carriers usually assign each spoke to a single nearby hub. This design reduces operational complexity, since flows between spokes must pass through their associated hubs. In this study, we take hub-spoke assignments as given and focus exclusively on optimizing the hub network. Shipments originating or terminating at spoke terminals are mapped to hubs, so the hub network fully captures the flow decisions of the underlying hub-and-spoke system.

Within the hub network, carriers design repeatable primary service routes, which serve as the backbone of daily operations. More precisely, given a pattern of package shipment demands, or *commodities*, carriers plan freight routes and capacity schedules to serve all shipments, aiming to minimize total transportation and operational costs. Some carriers also offer a *service time guarantee*, typically defined as the number of days within which a shipment must be delivered (Bakir et al. 2021). Motivated by the operations of such carriers in the United States, we study a hub-level service network design problem with service time guarantees at nationwide scale.

To make joint decisions for routes and schedules, time-space networks have been widely used in service network design problems. These networks model terminal activities at different times using distinct nodes; however, optimization models based on time-space networks are notoriously difficult to solve in industrial settings due to their size (Bakir et al. 2021). The large network size, even with a crude time discretization, and the large number of shipment commodities both contribute to the intractability of the model. For example, using only a quarter of the input data provided by our industry partner, a typical integer program (IP) for service network design may require more than 4 million variables.

To address this intractability, we propose a scalable method that can be used to solve problems at the scale of the entire United States network or networks of similar size. Specifically, we develop a partitioning and batching algorithm that recursively reduces the problem size using the problem's inherent structure. We demonstrate the scalability and efficiency of our approach through computational studies on instances generated from real-world data provided by an industry partner: our instances include over 1,000 nodes, 1 million arcs, and 40,000 commodities, which is the largest instance considered in the relevant literature when constructing a solution. Our approach finds solutions for this massive-scale problem in under four hours and identifies up to 10% cost savings when compared to a plan derived from freight routing rules used by our industrial partner.

The remainder of the paper is organized as follows. In Section 2, we discuss relevant literature. In Section 3, we formally describe the service network design problem, or less-than-truckload (LTL) load planning problem, with service time guarantees. We present our solution approach to solve this problem in

Section 4. In Sections 5 and 6, we present and discuss the results of the computational experiments on real-world and random instances, respectively. Finally in Section 7, we make concluding remarks and discuss potential future work.

2 Literature Review

Service network design problems have been studied in many transportation applications, including trucking (e.g. Powell and Sheffi 1983, Powell 1986, Kim et al. 1999, Jarrah et al. 2009, Üster and Maheshwari 2007, Üster and Agrahari 2010, Erera et al. 2013, Lindsey et al. 2016), rail (e.g. Newton et al. 1998, Lulli et al. 2011, Zhu et al. 2014), and public transit (e.g. Pattnaik et al. 1998, Mauttone and Urquhart 2009). As our motivating application is small package carrier operations, we focus our attention on LTL service network design. For general service network design in all areas of freight transportation, we refer the reader to the comprehensive reviews (Crainic 2000, Wieberneit 2008) and the book (Crainic et al. 2021).

We briefly review the literature on capacitated multi-commodity fixed-charge network design (CMND), which is the broad category our problem belongs to. CMND is known to be challenging and heuristics are usually necessary to find high-quality solutions for large-scale problems. Many researchers have used metaheuristics to find solutions; Crainic (2000) develop a tabu search method using simplex pivoting and column generation, and Crainic and Gendreau (2002) present its parallelized version. Crainic et al. (2004) propose further improvements based on slope scaling; this method iteratively updates the linearized fixed charge costs until a stopping criterion is met. Ghamlouche et al. (2003, 2004) present tabu search algorithms with cycle-based neighborhood structures. There was also an effort to strengthen formulations; Balakrishnan et al. (2017) develop several classes of cuts to solve CMND with service requirements. While these models allow the flow for a commodity to be split among multiple paths, our model selects exactly one path for each commodity, making it more challenging to solve.

We next review papers on LTL flow and load planning, which focus on similar models to ours. Flow planning determines freight routes for commodities, where a freight route for a commodity is a sequence of direct trailers connecting its origin terminal to its destination terminal. Here, the word “direct" refers to the fact that a commodity assigned to a direct trailer (A$\rightarrow$B) is loaded into a trailer at terminal A and not unloaded until it reaches terminal B; see Bakir et al. (2021) for more details. Jarrah et al. (2009) study LTL flow planning with explicit service commitments to customers. They decompose a large optimization model into separate, smaller IP models for each destination terminal and use a slope-scaling technique. Their algorithm solves instances with up to 725 nodes, 30,000 arcs, and 680 commodities, significantly smaller than the problem size we consider in this study. In addition, their model uses daily granularity, which may substantially overestimate consolidation opportunities. In contrast, our model divides a day into four time shifts to better model true consolidation opportunities.

Load planning is more detailed than flow planning and typically considers dispatch timing. Erera et al. (2013) develop a detailed time-space network model with a fine discretization of time, dividing a day into eight time periods, and propose an improvement heuristic. Their IP-based local search method (a large neighborhood search approach) iteratively solves an IP defined for each destination terminal to find improvement and is shown to be effective for an instance with 5,000 nodes, 500,000 arcs, and 40,000 commodities, which is of similar scale to ours. Lindsey et al. (2016) further improve this method by simultaneously rerouting commodities destined for multiple terminals in each neighborhood search iteration. While these heuristics improve a given solution, our goal is to construct a solution.

Recent work in LTL flow and load planning has focused on managing time-dependent or uncertain freight demand (Erera et al. 2013, Bai et al. 2014, Hewitt et al. 2019, Wang and Qi 2020, Baubaid et al. 2021). In addition, some researchers have integrated driver and trailer resource planning with flow and load planning (Crainic et al. 2016, 2018, Hewitt et al. 2019, Hewitt and Lehuédé 2023). Other recent work has developed approaches based on dynamic discretization discovery (Boland et al. 2017, 2019, Hewitt 2019, Scherr et al. 2020, Marshall et al. 2021, Hewitt 2022), which dynamically determines dispatch timing instead of specifying a time discretization in advance. While these recent works address important and practical factors, they introduce additional computational complexity. The solution approaches presented in these works, except perhaps for the improvement heuristics in Erera et al. (2013), remain computationally expensive and intractable at our desired scale. There has also been progress for industrial-scale optimization in related but different settings. For example, Lara et al. (2023) study an instance with 1,226 nodes, 10,003 arcs, and 102,476 commodities, but their focus is on scheduling shipments along predetermined paths rather than optimizing routing decisions. In contrast, we optimize both routes and schedules simultaneously.

3 Problem Definition

We model the operations of a carrier with a time-space network, $G = (N, A)$, where terminal activities in different shifts are modeled with different nodes. Specifically, given the set of terminals L in the carrier's network and the set of labor shifts T in the terminals, we define the set of nodes as $N = L \times T$. We assume only a subset of terminals can act as transfer locations. Shipments can occur between any two terminals, defining the arcs A of the network. Each arc $(i, j) \in A$ has a transit time t_{ij}; the transit time and origin shift (arc tail) determine the destination shift (arc head) in the time-space network. Figure 1 illustrates an example of a time-space network. As depicted in the figure, we differentiate arcs into two types, *holding* and *transportation* arcs. Holding arcs connect consecutive shifts at the same terminal, while transportation arcs connect two different terminals. We consider a representative day of demand and assume that the time-space network models a single, "wrapped" day. For instance, arc (k, ℓ) in Figure 1 indicates an arc from terminal L_2 at shift T_4 to terminal L_3 at shift T_2 the next day.

In our motivating application, terminals do not store inventory for freight in transit. Therefore, we do not allow packages to flow on holding arcs unless they have arrived to their destination terminal. For example, path $i - j - k - \ell$ in Figure 1 is infeasible because it includes a holding arc in a non-destination terminal. On the other hand, we do allow a trailer to arrive at a node before the start of its corresponding shift and wait until this shift starts. In the same example, we assume arc (i, k) exists and its transit time is $t_{ij} + t_{jk}$. The operation modeled by arc (i, k) is slightly different from that modeled by path $i - j - k$, since it does not involve certain sorting activities, and packages on arc (i, k) cannot be consolidated with packages on arc (i, j). Because of these holding arc restrictions and the fact that only some nodes allow transfers, the set of feasible arcs for each commodity is not necessarily identical.

The set of commodities K captures daily demand for the representative day; commodity k represents an aggregated demand from origin node o_k to destination node d_k with a service time guarantee of g_k days and volume q_k. We assume homogeneous trailers with capacity Q. The transportation cost on arc (i, j) is m_{ij} per trailer, where the number of trailers per arc is determined by the total volume of commodities flowing on the arc and the trailer capacity. We also consider handling cost of f^H per volume, incurred each time a commodity is processed at a transfer node. The goal is to determine paths for all commodities that satisfy service guarantees while minimizing the sum of transportation and handling costs.

We formulate the problem as IP (1) with two types of decision variables: $x_{ij}^k \in \{0, 1\}$ indicates whether

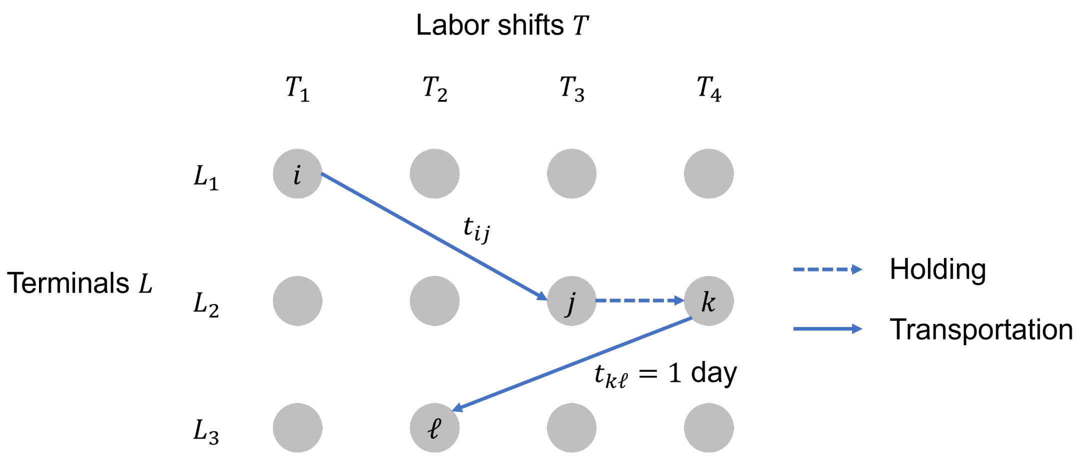


Figure 1: An example of a time-space network

the path for commodity k includes arc (i, j), defined only for a commodity's feasible arcs; $z_{ij} \in \mathbf{Z}_+$ represents the number of trailers on arc (i, j). Our approach, detailed in the next section, iteratively solves a restriction of this model given a partial solution. That is, predetermined arc flow p may be given as an input in addition to graph $G = (N, A)$ and commodity set K. To clarify the input, we use the notation IP (G, K, p):

$$\min_{x_{ij}^k \in \{0,1\},\; z_{ij} \in \mathbf{Z}_+} \sum_{(i,j)\in A} m_{ij} z_{ij} + f^H \sum_{k\in K} q_k \sum_{(i,j)\in A:\text{ transportation arc}} x_{ij}^k$$

$$\text{s.t.} \sum_{(i,j)\in A} x_{ij}^k - \sum_{(j,i)\in A} x_{ji}^k = \begin{cases} 1 & \text{if } i = o_k \\ -1 & \text{if } i = d_k \\ 0 & \text{otherwise} \end{cases} \quad \forall k \in K, i \in N \tag{1a}$$

$$p_{ij} + \sum_{k\in K} q_k x_{ij}^k \le Q z_{ij} \quad \forall (i,j) \in A \tag{1b}$$

$$\sum_{(i,j)\in A} t_{ij} x_{ij}^k \le g_k \quad \forall k \in K \tag{1c}$$

The objective function minimizes the total cost. Constraints (1a) are flow balance constraints, (1b) determine the number of trailers on each arc, and (1c) ensure service guarantees.

4 Solution Methodology

Our approach uses graph partitioning and iteratively solves restricted versions of (1) to obtain a solution for a large-scale instance. We begin by outlining the motivation and then formally describe the algorithms.

Our model has two challenging factors: the network size and the number of commodities. To reduce the network size, we partition the network into smaller *regions* and then solve the problem for each region, considering only commodities whose origin and destination are in the same region. This strategy naturally avoids paths with excessive detours. For commodities that cross regions (referred to as *crossing commodities*), we determine an appropriate sub-network and then solve (1) on this sub-network. Because of the large number of commodities, we divide these into batches and solve the problem iteratively; after solving for a batch, we fix its solution and proceed with the next batch, repeating until we consider all commodities.

We present an overview of our proposed solution approach in Algorithm 1. The algorithm takes the terminal set L and commodity set K as input. If the graph G induced by L is small enough, we directly apply the batching algorithm (Algorithm 2) on G, K, with zero initial arc flow. Otherwise, we partition L into regions $\{L_1,\ldots,L_r\}$ using a partitioning logic. This also partitions K into $\{K_1,\ldots,K_r,K_c\}$ where K_r represents the set of commodities with both origin and destination terminals in L_r, and K_c represents the set of crossing commodities. We then run Algorithm 2 for each region (using input L_ℓ and K_ℓ for region ℓ) independently and in parallel. We add any commodity without feasible paths within its region to K_c. After obtaining solutions for all regions, we run the crossing commodity algorithm (Algorithm 3) for K_c with initial arc flow $\sum_\ell x_\ell$, where x_ℓ is the solution for region ℓ. As a final step, we improve the final solution x by reoptimizing commodities in trailers with low utilization (Algorithm 4). We only apply this *polishing* step to the final solution of the entire network, rather than polishing partial solutions in intermediate steps; in preliminary results, we observed that the latter would result in a lower quality final solution. We provide detailed explanations and results in Section 5.3.

Algorithm 1 Recursive Partitioning and Batching

```
1: Input: Set of terminals $L$, Set of commodities $K$
2: **if** $|L| \leq$ given threshold **then**
3:     **return** $\textit{Batching}(G,K,0)$
4: **end if**
5: $(L_1,K_1),\ldots,(L_r,K_r),K_c \leftarrow \textit{Partitioning}(L,K)$
6: **for** each region $\ell$ **do**
7:     $x_\ell, K_\ell^0 \leftarrow \textit{RecursivePartitioningAndBatching}(L_\ell,K_\ell)$
8: **end for**
9: final solution $x \leftarrow \textit{CrossingCommodity}(\{L_1,\ldots,L_r\},K_c \cup \bigcup_\ell K_\ell^0, \sum_\ell x_\ell)$
10: $x \leftarrow \textit{Polishing}(x,K)$
11: **return** $x$
```

4.1 Batching

Algorithm 2 describes how we determine batch size b and select commodity batches. When selecting a batch, we prioritize commodities with less flexibility to enable consolidation. Specifically, we compute a *slack* for each commodity and consider commodities in increasing order of slack; the set of possible slack values is discrete and typically small, usually up to four. For commodities with negative slack time, which do not have feasible paths within graph G, we return them as K^0 and reconsider them later on a larger network in Algorithm 3; these commodities do not have a feasible path that only uses arcs within their region. Among commodities with the same slack, we randomly choose up to b commodities, with selection probability proportional to the commodity's volume.

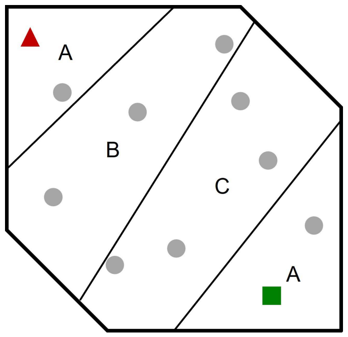
The motivation behind Algorithm 2 is the following. Commodities that are more strictly time-constrained to meet their service time guarantee only have a limited number of candidate paths. Once these commodities' paths are fixed, we expect to use the arcs in those paths for less time-constrained commodities, taking advantage of consolidation. To roughly estimate the time flexibility of each commodity, we calculate its slack and consider commodities with smallest slack first. In addition to time flexibility, we consider a kind of trailer flexibility, which represents the number of "new" trailers needed to deliver each commodity. Intuitively, commodities with low volume are more flexible, as it is likely that we can squeeze them into existing trailers. With this intuition, we prefer to include high-volume commodities first when generating a batch; to generate different batches in different runs, we choose commodities for a batch randomly, where a commodity's probability of inclusion in the batch is proportional to its volume (line 8).

Algorithm 2 Batching (Least Flexible/High Volume First)

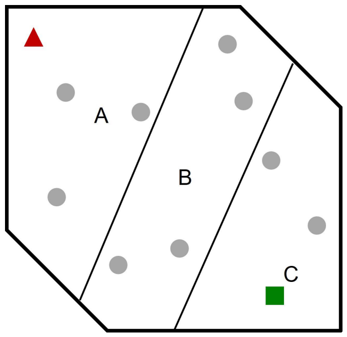
```
1:  Input: Graph $G$, Set of commodity $K$, Predetermined arc flow $x$
2:  Compute slack $s_k = g_k -$ shortest transit time in days from $o_k$ to $d_k$ in $G$ for each $k \in K$
3:  Set of infeasible commodities $K^0 \leftarrow \{k \in K : s_k < 0\}$, $K \leftarrow K \setminus K^0$
4:  for each (slack $s$, service time guarantee $g$) do
5:      $K_{s,g} = \{k \in K : s_k = s, g_k = g\}$
6:      Calculate batch size $b \leftarrow \max\{1, \lfloor \frac{\text{Maximum number of decision variables}}{\text{Average number of decision variables per commodity in IP}(G, K_{s,g}, x)} \rfloor\}$
7:      while $K_{s,g} \neq \emptyset$ do
8:          $\tilde{K} \leftarrow$ Randomly choose $\min\{b, |K_{s,g}|\}$ commodities (selection probability for $k \propto q_k$)
9:          $x \leftarrow x +$ solution from IP$(G, \tilde{K}, x)$
10:         Remove $\tilde{K}$ from $K_{s,g}$
11:     end while
12: end for
13: return $x, K^0$
```

4.2 Partitioning

We next describe the graph partitioning method. The purpose of partitioning is to find a reasonably small sub-graph that contains feasible routes for commodities whose origin and destination terminals are within the same sub-graph. We therefore seek a partition where the sub-networks are contiguous and compact. In Figure 2a, we illustrate an example of a non-contiguous partition. Region A in Figure 2a is not contiguous, as it consists of two disconnected components. Thus, for a commodity originating at the red triangle and going to the green rectangle, we may not be able to find a feasible path by considering terminals only in region A; we may need to transfer at other terminals in region B or C. In contrast, in Figure 2b, the partitioned regions are all contiguous and compact, and the same commodity is now considered a crossing commodity.



(a) Discontiguous graph partition



(b) Contiguous graph partition

Figure 2: Examples of graph partitions where circles, triangles, and rectangles represent terminals and letters represent partitioned regions.

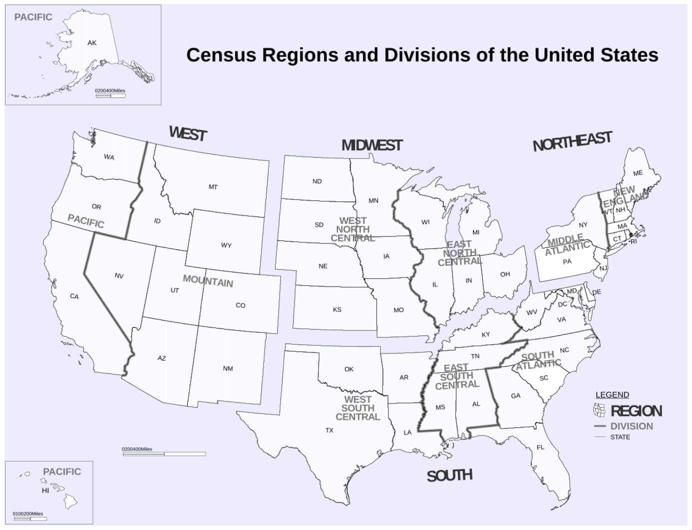


Figure 3: Divisions of the United States (United States Census Bureau 2013)

In practical applications of our problem, we expect to use existing geographic and demographic information to design these partitions. For example, in the case of the US, we use census divisions (Figure 3). We divide the US into four regions (West, Midwest, South, Northeast) and, if necessary, partition each region into multiple sub-regions. For example, we may partition the South region into {West South Central, East South Central, South Atlantic} or even smaller sub-regions defined by states or groups of states. Furthermore, we refine the geographic partition using commodity data. Starting from the predetermined geographic sub-regions, we apply a local search heuristic to redefine the state assignment of each terminal with the objective of minimizing the total volume of crossing commodities. At each iteration, we either swap two terminals across states in different sub-regions or move one terminal to a state in another sub-region, while ensuring that the resulting partitions remain contiguous and compact. Figure 4 illustrates the process. The leftmost figure shows three regions (A, B, C) and their sub-regions (e.g., {A1, A2, A3} within region A), obtained from the initial geographic partitioning. For simplicity, suppose each sub-region corresponds to a state. In one iteration, we move one terminal in B2 (red rectangle) to A2 if this reduces the crossing commodity volume; this yields the middle partition. In the next iteration (the rightmost figure), we swap two terminals in B1 and C1 (green triangles) if the same condition is met. Note that the local search heuristic only redefines the state of each terminal while preserving the given region and sub-region hierarchy. In Section 5, we test our method with various partitioning configurations and analyze their impact on solution quality and computational time.

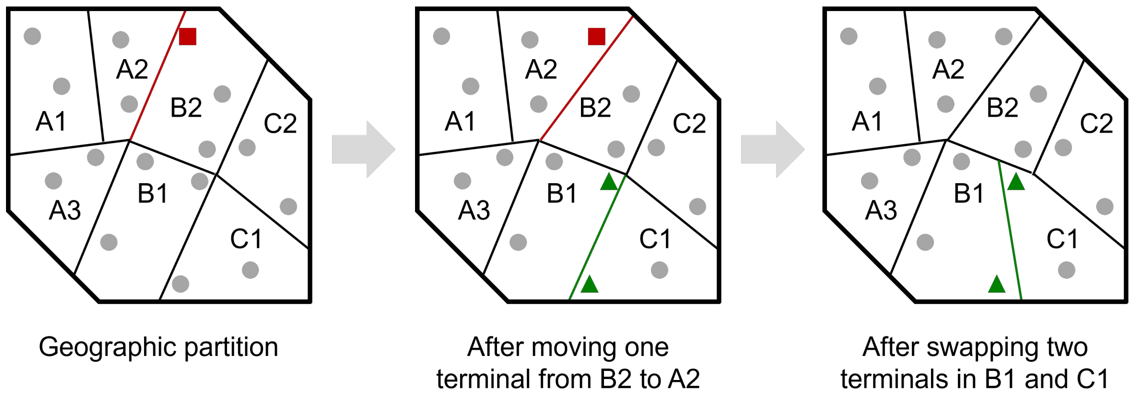

A1
A2
A3
B1
B2
C1
C2
Geographic partition
After moving one terminal from B2 to A2
After swapping two terminals in B1 and C1

Figure 4: Example of local search operations used to refine the geographic partition.

Applying the local search heuristic reduced the package volume crossing four regions (West, Midwest, South, Northeast) by 5.4% (from 1.56×10^7 to 1.48×10^7 cubic feet). Within the West, Midwest, Northeast, and South regions, the volume crossing sub-regions decreased by 8.6%, 22.9%, 8.9%, and 36.3%, respectively. Through preliminary tests, we verified that using the refined state assignment decreased the solution time by more than 60% while having negligible impact on solution quality, when compared to using the initial geographic state information. Therefore, we use the refined state assignment throughout the paper.

Our computational experiments all use instances with geographic information. If this information is not available, it may be possible to leverage methods from graph partitioning (Alpert and Yao 1995, Hendrickson et al. 1995, Karypis and Kumar 1997, Andreev and Räcke 2004), including methods that explicitly require contiguity, e.g. van't Hof et al. (2009). It may also be possible to adapt methods from political districting, e.g. Bozkaya et al. (2003), Bacao et al. (2005), Ricca et al. (2008). We refer the interested reader to these references for more information.

4.3 Crossing Commodities

Algorithm 3 details how we construct sub-networks for crossing commodities and the order in which we process crossing commodities. Recall that crossing commodities have origin and destination nodes in different regions. To find feasible paths for them, we need to identify a reasonably small sub-network that includes multiple regions. To minimize the network size and also encourage consolidation, we add only a few arcs connecting different regions to the arcs that are already used within regions; we denote this set of arcs A^* (line 2). The number of selected arcs may still be large, resulting in small batch sizes. To handle this, we partition each region into sub-components and construct a sub-component-based adjacency graph (lines 4-12). Using A^* and the adjacency graph, we solve the problem for crossing commodities in increasing order of sub-component-based distance (lines 6-11).

Algorithm 3 Crossing Commodity

```
 1: Input: Partition of terminals $\{L_1,\dots,L_r\}$, Set of commodity $K$, Predefined arc flow $x$
 2: $A^* \leftarrow \mathit{ArcSelection}(\{L_1,\dots,L_r\},K,x)$
 3: Define set of subcomponents $S$ by partitioning regions
 4: Construct adjacency graph $H$ of $S$
 5: Sort $\{(s_o,s_d) : s_o,s_d \in S$ in different regions$\}$ in increasing order of distance in $H$
 6: for each $(s_o,s_d)$ do
 7:     $K^* \leftarrow \{k \in K : o_k$ is in $s_o, d_k$ is in $s_d\}$
 8:     $L^* \leftarrow \{l \in L :$ subcomponent of $l$ is included in any shortest path from $s_o$ to $s_d$ in $H\}$
 9:     Construct a subgraph $G^*$ induced by $L^*$ and $A^*$
10:     For commodity $k \in K^*$ with no feasible paths in $G^*$, add arc $(o_k,d_k)$ to $G^*$
11:     Add solution from $\mathit{Batching}(G^*,K^*,x)$ to $x$
12: end for
13: return $x,\emptyset$
```

The arc selection process (line 2) begins with arcs that have already been used within regions and adds additional arcs through four steps:

1. Add arcs from origin to destination nodes, say *OD arcs*, for the top $\alpha\%$ of high-volume commodities in K.
2. Add the top $\beta\%$ nearest missing arcs between transfer nodes in different regions.
3. For each commodity with no feasible path using only A^*, add arcs from its origin/destination node to the γ nearest transfer nodes.
4. For each commodity still without any feasible path using arcs in A^*, add its OD arc (if it is the only feasible path in the original graph), or add the arcs in the shortest path with at least one transfer in the original graph.

We choose the best combination of (α, β, γ) through preliminary experiments.

We next illustrate how we construct and use the sub-component-based adjacency graph (Figure 5). Given the adjacency graph, we identify connected sub-component pairs. For example, the shortest path from A3 to C1 is A3-B1-C1 in the figure. Thus, for commodities with origin in A3 and destination in C1, we generate a sub-graph using arcs in A^* whose endpoints are in $\{$A3,B1,C1$\}$. The usage of the sub-component-based adjacency graph enables us to decrease the sub-network size for commodities whose origin and destination are close to each other.

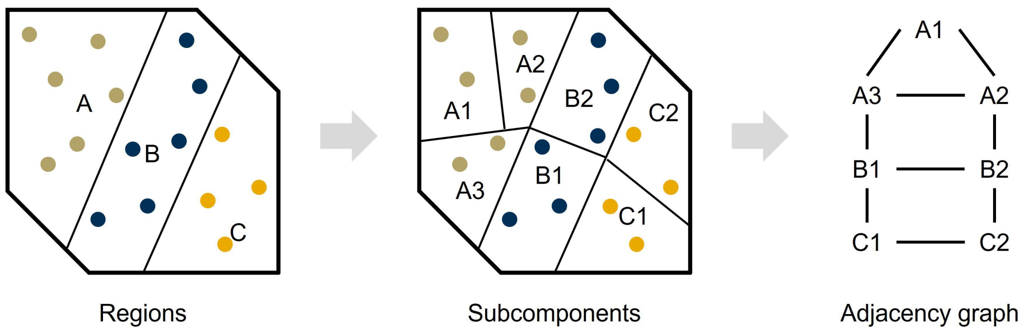


Figure 5: Construction of a subcomponent-based adjacency graph

4.4 Polishing

After obtaining a solution for a region, we polish it by reoptimizing a small number of commodities; Algorithm 4 summarizes the procedure. We first identify the set of arcs used in the solution, denoted by $\tilde{A}$, and compute their trailer utilization values. We then define the set of low-utilization arcs A^{low} based on the tenth percentile of these values, and the set of commodities K^{low} that traverse such arcs in the solution. We fix the solution paths for $K \setminus K^{low}$ and reoptimize only K^{low}, only considering arcs in $\tilde{A}$. Because $|\tilde{A}|$ is much smaller than the original set of arcs and $|K^{low}|$ is also small, we can solve the IP formulation for K^{low} all at once. In Appendix A.4, we evaluate the effectiveness of the polishing algorithm.

Algorithm 4 Polishing

```
1: Input: Arc flow $x$, Set of commodities $K$
2: Set of arcs with positive arc flow $\tilde{A} \leftarrow \{a : \sum_k x_a^k > 0\}$
3: For each arc $a \in \tilde{A}$, compute its trailer utilization $\frac{\sum_k x_a^k}{Q\lceil(\sum_k x_a^k/Q)\rceil}$
4: $A^{low} \leftarrow$ arcs in $\tilde{A}$ with trailer utilization below the 10th percentile value
5: $K^{low} \leftarrow \{k \in K : x_a^k > 0$ for some $a \in A^{low}\}$
6: Construct a subgraph $\tilde{G}$ induced by $\tilde{A}$
7: Define arc flow $\tilde{x}$ only for $K \setminus K^{low}$, i.e. $\tilde{x} = \sum_{k \in K \setminus K^{low}} x_a^k$
8: Reoptimized solution $x^{low} \leftarrow$ solution from IP($G, K^{low}, \tilde{x}$)
9: **return** $\tilde{x} + x^{low}$
```

5 Computational Study: Industrial Instances

We conduct experiments to assess the effectiveness of our approach and to understand features that contribute to this effectiveness, using instances of industrial scale. We create these instances using real-world data provided by an industrial partner. We compare the performance of the plan produced by our algorithm (Proposed) against a representation of the baseline plan used by the partner (Baseline) and the plan generated by a commercial MIP solver (Gurobi). We test with three handling cost coefficients, 0.01, 0.2, and 0.4. In preliminary experiments, we observe that using a small positive handling cost such as 0.01 instead of zero yields better solutions and improved algorithm performance; see Appendix A.1 for further details. We performed experiments on a high-performance computing cluster with AMD EPYC-Rome processors and 80 GB of RAM, using Python for implementation and Gurobi 10.01 for solving MIPs. We used a 600-second time limit and 0.1% lower bound gap tolerance for all IP solves.

Section 5.1 describes the instances. In Section 5.2, we study the impact of each feature in our proposed method. Finally, we benchmark the method's performance in Section 5.3.

5.1 Instances

We design six instances with varying network size and number of commodities. Specifically, we choose a subset of terminals from the system of our industrial partner, construct a network based on the chosen set of terminals, and consider only commodities whose origin and destination are both in the constructed network. We present the characteristics of these instances in Table 1. *Commodity volume* represents the fraction of total US commodity volume represented in each instance. Corresponding to the number of arcs and commodities, we classify instances 1 - 3 as "small", instances 4 - 5 as "medium" and instance 6 as "large".

Instance	Region	$\|L\|$	$\|N\|$	$\|A\|$	$\|K\|$	Commodity volume (%)
1	West	60	240	57,360	1,196	1.03
2	Midwest	52	208	43,056	1,988	7.09
3	Northeast	47	188	35,156	2,274	7.70
4	South	95	380	144,020	8,549	29.63
5	West + Midwest	112	448	200,256	5,699	12.63
6	Entire US	254	1,016	1,032,140	40,167	-

Table 1: Characteristics of instances.

Each instance differs in the number of commodities and network size but shares a similar portfolio of service time guarantees, with the exception of instance 3. In instance 3, more than half of the commodities require a one-day service time guarantee, whereas in all other instances the majority of commodities are assigned a two-day guarantee. This difference may stem from the geographic characteristics of the Northeast region, which is relatively small compared to the other regions. In Section 5.2, we evaluate algorithm features on instance 3 in addition to instance 1, which is representative of the other instances.

We next describe the parameters used in the experiments, all of which were provided by our industrial partner. The per-vehicle arc transportation cost m_{ij} increases linearly with distance, using a higher rate for distances below a certain threshold and a lower rate for longer ones. For the trailer capacity Q, we use the size of a 28 ft. pup trailer; in practical terms, for the total daily volume in Instance 6, this corresponds to at least 13,690 trailers per day, only considering the volume in cubic feet. The actual number of trailers dispatched per day by the partner is approximately 22,500.

5.2 Impact of Algorithm Features

In this subsection, we test the proposed method with various configurations to understand which features contribute to its performance. We first evaluate the impact of randomization on the batching algorithm (Algorithm 2) and graph partitioning on representative small instances (instances 1 and 3). We also examine the sensitivity of the batching algorithm to batch size. In addition, we evaluate the impact of the crossing commodity method (Algorithm 3) on a medium instance (instance 4).

5.2.1 Randomization in Batching

In Algorithm 2, we select commodities to add to the batch by randomly choosing a commodity with probability proportional to its volume. To evaluate the impact of randomization, we run the algorithm using three different random seeds, in addition to the deterministic version of the algorithm (which always adds the commodity with the highest volume), and compare the results. For simplicity, we report only the results on instance 1; results on instance 3 are in Appendix A.2. Table 2 summarizes the objective values and key business metrics obtained on instance 1. The best result for each handling cost scenario is in bold.

Instance	f^H	Seed	Obj	Solution Gap (%)	APL	Util	Time (s)
1	0.01	-	114,089	0.21	2.48	0.76	7,419
		1	115,198	1.17	2.47	0.76	7,402
		2	**113,852**	**0.00**	**2.40**	**0.75**	**6,728**
		3	114,445	0.52	2.47	0.77	8,126
	0.2	-	199,470	0.07	1.78	0.72	3,554
		1	200,866	0.76	1.80	0.73	3,106
		2	202,677	1.65	1.80	0.72	5,027
		3	**199,332**	**0.00**	**1.80**	**0.73**	**2,727**
	0.4	-	283,687	0.56	1.68	0.68	1,921
		1	282,462	0.13	1.65	0.68	1,996
		2	283,618	0.53	1.69	0.68	2,145
		3	**282,108**	**0.00**	**1.65**	**0.68**	**2,732**

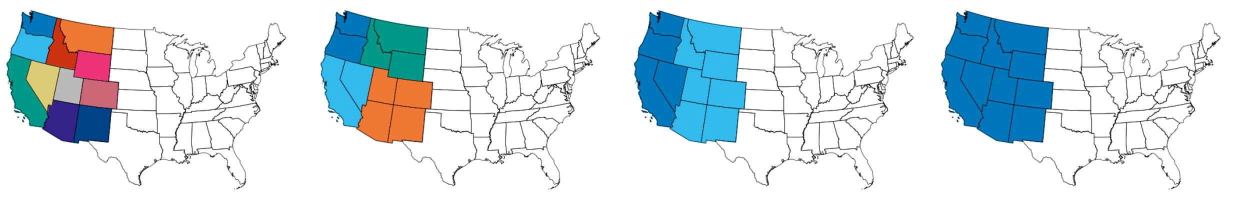
Table 2: Results obtained by the proposed method with different random seeds and the deterministic version of the method on instance 1.

A seed marked with "-" indicates the use of the deterministic version of the algorithm, while numeric values represent the random seeds used in test runs. *Obj* refers to the objective, and *Solution Gap* represents the percentage gap between the best-found objective and the objective obtained for each configuration. i.e., Solution gap (%) = (objective obtained − best-found objective) / (objective obtained) $\times$ 100. *APL* denotes the volume-weighted average path length in the solution paths, and *Util* measures the distance-weighted trailer utilization, APL $= \frac{\sum_{k\in K} q_k \sum_{(i,j)\in A} \bar{x}^k_{ij}}{\sum_{k\in K} q_k}$, Util $= \frac{\sum_{(i,j)\in A} d_{ij} \sum_{k\in K} (q_k/Q)\bar{x}^k_{ij}}{\sum_{(i,j)\in A} d_{ij}\bar{z}_{ij}}$, where $\bar{x}$ and $\bar{z}$ represent the solutions to IP (1). Additionally, we report the total solution runtime in seconds in the last column.

As shown in Table 2, the randomized algorithm outperforms the deterministic version across all handling cost scenarios on instance 1. This result suggests that the usage of randomization in batching is beneficial. In the following sections, we present only the best result obtained through randomization for each scenario.

5.2.2 Graph Partitioning

In this subsection, we test the algorithm using four different sub-region partitioning options on instance 1 and compare the results. As before, we report only the results on instance 1 and defer the results on instance 3 to Appendix A.3. Figure 6 illustrates sub-region partitioning options for the West region (instance 1), where each color represents a different partition. Options 1, 2, and 3 correspond to dividing the region into two, four, and 11 sub-regions, respectively, while option 4 corresponds to no further sub-region partitioning. For options 1 through 3, we applied the least restricted version of the crossing commodity algorithm, selecting all arcs within the West region in line 2 of Algorithm 3. Table 3 summarizes the results obtained for these partitioning options. *Partition* refers to the partitioning option used, and all other metrics are the same as in Table 2. The best result for each handling cost scenario is in bold.



(a) Option 1 (b) Option 2 (c) Option 3 (d) Option 4

Figure 6: Four sub-region partitioning options for West region.

Comparing the runtimes across handling cost scenarios, we observe that using smaller sub-regions reduces runtime. For example, when $f^H = 0.4$, the runtime for partitioning option 1 is around two thirds of the runtime for option 4. One possible explanation for this difference is that smaller sub-regions enable the solver to quickly identify near-optimal solutions. Recall that we use a time limit of 600 seconds and a 0.1% lower bound gap tolerance when solving an IP for each commodity batch. When sub-regions are small, we typically obtain solutions within 10 seconds for a single batch. In contrast, with large sub-regions we may stop at solutions with gaps exceeding 5% when reaching the time limit.

Instance	f^H	Partition	Obj	Solution Gap (%)	APL	Util	Time (s)
1	0.01	1	113,852	3.14	2.40	0.75	6,728
		2	**110,276**	**0.00**	**2.38**	**0.79**	**8,153**
		3	113,088	2.49	2.57	0.79	12,073
		4	121,154	8.98	2.89	0.83	12,223
	0.2	1	199,332	0.04	1.80	0.73	2,727
		2	**199,257**	**0.00**	**1.82**	**0.75**	**4,443**
		3	204,035	2.34	1.85	0.73	6,251
		4	214,390	7.06	2.03	0.78	6,461
	0.4	**1**	**282,108**	**0.00**	**1.65**	**0.68**	**2,732**
		2	284,071	0.69	1.70	0.70	3,292
		3	283,314	0.43	1.67	0.69	4,622
		4	300,230	6.04	1.81	0.75	4,045

Table 3: Results obtained by the proposed method with different sub-region partitions on instance 1.

As shown in Table 3, we find the best solutions using partitioning option 1 or 2 across all handling cost scenarios. This indicates that partitioning the region into smaller sub-regions is an effective strategy. Furthermore, this demonstrates that our crossing commodity method constructs reasonable sub-networks for crossing commodities; from an optimization perspective, the feasible region under option 2 is smaller than that of option 4. The fact that option 2 achieves a better objective than option 4 shows that our algorithm does not excessively cut off the feasible region. Based on these results, we partition the Midwest and South regions into small sub-regions; sub-region partitioning options used for Midwest, Northeast, and South regions are discussed in Appendix A.3.

5.2.3 Batch Size

In this subsection, we study the sensitivity of the batching algorithm to batch size. Recall that we determine the commodity batch size based on the maximum number of decision variables we wish to include in a single IP formulation (line 6 of Algorithm 2). Intuitively, we choose the largest possible batch size that avoids a memory issue, excessive solution time and/or poor solution quality for the batch's IP. To investigate whether larger batch sizes always yield better results, we run the algorithm with a reduced batch size by half and compare the results for each handling cost scenario and partitioning option. We use partitioning options 1 and 2 described in Section 4.2. Table 4 summarizes the results obtained using the original and reduced batch sizes. *Batch Size* refers to the batch size used, either 1 (original) or 0.5 (reduced). All other metrics are the same as in Table 3 except for *Solution Gap*. To facilitate a direct comparison between batch sizes, we calculate *Solution Gap* for each handling cost scenario and partitioning option. The best result for each handling cost scenario is in bold.

Instance	f^H	Partition	Batch Size	Obj	Solution Gap (%)	APL	Util	Time (s)
1	0.01	1	0.5	116,190	2.01	2.54	0.75	17,512
		1	**1**	**113,852**	**0.00**	**2.40**	**0.75**	**6,728**
		2	0.5	113,082	2.48	2.44	0.77	18,070
		2	**1**	**110,276**	**0.00**	**2.38**	**0.79**	**8,153**
	0.2	1	0.5	204,340	2.45	1.84	0.71	10,223
		1	**1**	**199,332**	**0.00**	**1.80**	**0.73**	**2,727**
		2	0.5	202,494	1.60	1.83	0.72	9,278
		2	**1**	**199,257**	**0.00**	**1.82**	**0.75**	**4,443**
	0.4	1	0.5	286,949	1.69	1.69	0.68	6,771
		1	**1**	**282,108**	**0.00**	**1.65**	**0.68**	**2,732**
		2	0.5	284,937	0.30	1.67	0.69	8,580
		2	**1**	**284,071**	**0.00**	**1.70**	**0.70**	**3,292**

Table 4: Results obtained by the propose method with different batch sizes on instance 1

As expected, using the original batch size outperforms the reduced batch size both in solution quality and runtime. Based on these results, we adhere to the original batch size in our remaining experiments.

5.2.4 Crossing Commodity Method

In this subsection, we test the proposed crossing commodity method (Algorithm 3) on instance 4 (South region), using several method variants. Recall that Algorithm 3 consists of two main steps: arc selection (step A) and sub-component-based adjacency graph construction (step S). To understand how each step contributes to the objective and runtime, we consider the following configurations:

- AS(30,25,3) / AS(40,30,3): Algorithm 3 with $(\alpha, \beta, \gamma) = (30, 25, 3)$ or $(40, 30, 3)$
- A(30,25,3) / A(40,30,3): Algorithm 3 without step S and with $(\alpha, \beta, \gamma) = (30, 25, 3)$ or $(40, 30, 3)$
- S: Algorithm 3 without step A (i.e. considering all available arcs)

We determined these (α, β, γ) configurations through preliminary experiments.

For a fair comparison, we initialize all five variants of the crossing commodity method with the same solution within sub-region partitions (i.e. the same predefined arc flow used as input for Algorithm 3). Specifically, we use partitioning option 3 in Figure 10 and find the best solutions for each sub-region through randomization. Note that we use all options in Figure 10 for the benchmarking experiment in Section 5.3. We report the results from crossing commodity methods in Table 5. *CC Method* denotes the crossing commodity method option, and all other metrics are consistent with those in Table 2. *Time* represents the runtime exclusively for the crossing commodity method in this table.

Instance	f^H	CC Method	Obj	Solution Gap (%)	APL	Util	Time (s)
4	0.01	**AS(30,25,3)**	**2,146,485**	**0.00**	**1.66**	**0.95**	**7,289**
		AS(40,30,3)	2,153,670	0.33	1.66	0.94	7,877
		A(30,25,3)	2,262,684	5.14	1.93	0.94	8,952
		A(40,30,3)	2,201,114	2.48	1.95	0.96	11,871
		S	2,156,005	0.44	1.90	0.96	72,693
	0.2	AS(30,25,3)	3,950,522	0.38	1.30	0.92	1,483
		AS(40,30,3)	**3,935,485**	**0.00**	**1.30**	**0.92**	**1,524**
		A(30,25,3)	4,031,554	2.38	1.36	0.95	2,495
		A(40,30,3)	4,020,622	2.12	1.36	0.95	1,892
		S	3,985,067	1.24	1.36	0.93	21,522
	0.4	AS(30,25,3)	5,775,147	0.58	1.26	0.89	376
		AS(40,30,3)	**5,741,863**	**0.00**	**1.25**	**0.89**	**364**
		A(30,25,3)	5,835,014	1.60	1.28	0.92	1,044
		A(40,30,3)	5,817,000	1.29	1.28	0.92	1,152
		S	5,786,235	0.77	1.30	0.88	13,457

Table 5: Results obtained by variants of crossing commodity method on instance 4.

We observe in Table 5 that the AS method consistently outperforms both the A and S method. Furthermore, the results demonstrate the complementary impact of steps A and S. Step S positively influences solution quality, while step A reduces runtime. Across all handling cost scenarios, the methods without step S produced inferior solutions, and the methods without step A required significantly longer runtimes, increased by a factor of 10 to 36. Given the effectiveness of the AS method, we test only its variants with different (α, β, γ) configurations on instances 5 and 6.

5.3 Benchmarking

Finally, we compare the performance of our algorithm (Proposed) against a plan generated by using freight routing rules from our industry partner (Baseline) and the plan produced by optimization via a commercial MIP solver (Gurobi). For the proposed method, we report the best results obtained from various configurations. We run Gurobi with a time limit of 86,400 seconds and a gap tolerance of 0.05%. Table 6 presents the results on small instances; *I* refers to the instance. *Tcost* and *Hcost* represent the transportation and handling costs, respectively. *LB gap* refers to the gap with respect to the best lower bound from Gurobi, i.e. LB gap (%) = (Objective obtained - lower bound)/(Objective obtained) $\times 100$. All other metrics remain consistent with those in Table 2.

I	f^H	Method	Obj	LB gap (%)	Tcost	Hcost	APL	Util	Time (s)
1	0.01	Proposed	110,276	26.76	103,984	6,292	2.38	0.79	9,420
		Baseline	271,196	70.22	267,586	3,610	1.34	0.29	-
		Gurobi	**107,462**	**24.84**	**102,598**	**4,863**	**1.90**	**0.73**	**86,400**
	0.2	Proposed	199,257	17.85	106,575	92,682	1.82	0.75	5,027
		Baseline	339,780	51.82	267,586	72,194	1.34	0.29	-
		Gurobi	**186,349**	**12.16**	**106,839**	**79,510**	**1.60**	**0.72**	**86,400**
	0.4	Proposed	282,108	13.70	114,906	167,201	1.65	0.68	3,292
		Baseline	411,974	40.90	267,586	144,388	1.34	0.29	-
		Gurobi	**263,539**	**7.62**	**111,978**	**151,561**	**1.56**	**0.69**	**86,400**
2	0.01	**Proposed**	**243,037**	**10.81**	**215,595**	**27,442**	**1.61**	**0.93**	**8,811**
		Baseline	308,963	29.84	286,827	22,136	1.18	0.69	-
		Gurobi	255,201	15.06	233,062	22,139	1.37	0.83	86,400
	0.2	Proposed	656,396	4.06	229,419	426,976	1.29	0.85	976
		Baseline	729,552	13.68	286,827	442,725	1.18	0.69	-
		Gurobi	**635,309**	**0.87**	**215,071**	**420,238**	**1.34**	**0.91**	**86,400**
	0.4	Proposed	1,073,749	2.37	252,506	821,243	1.29	0.77	660
		Baseline	1,172,278	10.58	286,827	885,451	1.18	0.69	-
		Gurobi	**1,055,501**	**0.68**	**235,150**	**820,351**	**1.28**	**0.83**	**86,400**
3	0.01	Proposed	172,436	10.46	144,442	27,994	1.51	0.92	10,391
		Baseline	198,782	22.32	174,329	24,453	1.24	0.77	-
		Gurobi	**167,758**	**7.96**	**143,190**	**24,568**	**1.35**	**0.89**	**86,400**
	0.2	Proposed	615,190	2.58	165,629	449,561	1.27	0.78	1,360
		Baseline	663,390	9.66	174,329	489,062	1.24	0.77	-
		Gurobi	**605,046**	**0.94**	**158,172**	**446,875**	**1.27**	**0.82**	**86,400**
	0.4	Proposed	1,055,020	1.23	188,667	866,353	1.23	0.67	1,319
		Baseline	1,152,452	9.58	174,329	978,124	1.24	0.77	-
		Gurobi	**1,042,527**	**0.04**	**169,348**	**873,179**	**1.25**	**0.76**	**5,572**

Table 6: Results obtained by the proposed method, baseline, and Gurobi on small instances.

Our proposed method outperforms the baseline in all instances, regardless of the handling cost. It should be noted that the improvements over the baseline are not necessarily representative of cost improvements possible in practice: freight routing rules sometimes vary by day of week, and they are sometimes overridden to avoid loads with low utilization. When compared to the plan produced by Gurobi, our solution is competitive; the solver usually slightly outperforms our method. However, Gurobi uses a full day of computing time on all instances save one, whereas our method consistently finds solutions in three hours or less. We could have further improved our solution by applying the polishing algorithm (Algorithm 4). However, we observed that polishing intermediate solutions worsens the final solution for the entire US region through our preliminary experiments.

We next present results on medium and large instances in Table 7. For instances 5 and 6, we apply the polishing algorithm to the final solution and present the polished result; the results before polishing are in Appendix A.4. Since we could not build a Gurobi model for these very large instances, we report only the results from our proposed method and the baseline. All metrics are the same as in Table 6. For the gap, we used the optimal objective of the LP relaxation of IP (1). The LP relaxation decomposes into independent resource-constrained shortest path problems for each commodity, and we can thus solve it even for the largest instance. *Time* in Table 7 refers to the total runtime to find the best solution, which is the sum of the time to solve problems for all relevant regions or sub-regions (e.g. West and Midwest for instance 5), the time to run the crossing commodity algorithm, and the time for the polishing algorithm (if applied). We report detailed runtimes in Table 8.

I	f^H	Method	Obj	LB gap (%)	Tcost	Hcost	APL	Util	Time (s)
4	0.01	**Proposed**	**2,146,485**	**10.71**	**2,022,437**	**124,049**	**1.66**	**0.95**	**34,679**
		Baseline	2,462,869	22.18	2,364,122	98,746	1.28	0.80	-
	0.2	**Proposed**	**3,935,485**	**12.49**	**2,040,301**	**1,895,184**	**1.30**	**0.92**	**8,265**
		Baseline	4,339,049	20.63	2,364,122	1,974,927	1.28	0.80	-
	0.4	**Proposed**	**5,741,863**	**12.02**	**2,109,700**	**3,632,163**	**1.25**	**0.89**	**3,207**
		Baseline	6,313,976	19.99	2,364,122	3,949,853	1.28	0.80	-
5	0.01	**Proposed**	**757,352**	**20.84**	**702,369**	**54,984**	**1.77**	**0.87**	**11,813**
		Baseline	1,331,546	54.97	1,275,360	56,186	1.29	0.62	-
	0.2	**Proposed**	**1,564,682**	**20.05**	**717,350**	**847,332**	**1.40**	**0.83**	**5,767**
		Baseline	2,399,073	47.86	1,275,360	1,123,713	1.29	0.62	-
	0.4	**Proposed**	**2,372,090**	**18.37**	**766,988**	**1,605,102**	**1.35**	**0.77**	**3,645**
		Baseline	3,522,787	45.03	1,275,360	2,247,426	1.29	0.62	-
6	0.01	**Proposed**	**7,428,697**	**12.38**	**6,973,879**	**454,818**	**1.82**	**0.95**	**81,028**
		Baseline	8,716,722	25.33	8,359,403	357,319	1.37	0.78	-
	0.2	**Proposed**	**13,899,950**	**16.08**	**7,146,477**	**6,753,473**	**1.38**	**0.91**	**12,840**
		Baseline	15,505,778	24.78	8,359,403	7,146,375	1.37	0.78	-
	0.4	**Proposed**	**20,350,978**	**16.02**	**7,530,234**	**12,820,744**	**1.32**	**0.86**	**6,140**
		Baseline	22,652,153	24.55	8,359,403	14,292,750	1.37	0.78	-

Table 7: Results obtained by the proposed method and baseline on medium and large instances

As shown in Table 7, our method consistently outperforms the baseline, demonstrating both scalability and efficiency. Specifically, our method solves the largest instance in 1.7 hours when the handling cost is large. This is particularly notable given that we could not even build the Gurobi model for the South region (instance 4). It is also surprising that our method finds a solution for instance 4 in under one hour when the handling cost is 0.4.

Another key observation is that the average path length and trailer utilization of our solutions decrease with larger handling cost on all instances (Tables 6 and 7); this indicates we can control the structure of our

I	f^H	Total time (s)	Solving all relevant region sub-problems (s)	Running crossing commodity algorithm (s)	Running polishing algorithm (s)
	0.01	34,679	26,416	8,263	-
4	0.2	8,265	6,588	1,677	-
	0.4	3,207	3,366	2,234	-
	0.01	11,813	9,420	5,165	57
5	0.2	5,767	5,027	695	45
	0.4	3,645	3,292	309	44
	0.01	81,028	34,679	44,010	2,339
6	0.2	12,840	8,265	3,285	1,291
	0.4	6,140	3,291	1,528	1,321

Table 8: Runtimes for region sub-problems, crossing commodity algorithm, and polishing algorithm.

solution by adjusting the handling cost coefficient. For example, if we prefer paths without excessive detours, we can use large handling cost coefficient instead of imposing constraints on the number of transfers.

Lastly, we elaborate on the detailed runtimes in Table 8. Recall that our method roughly consists of two main stages: (1) solving the problem for all relevant regions or sub-regions and (2) solving the problem for crossing commodities. For example, when solving the problem for the entire US region, we first find solutions for the West, Midwest, Northeast, and South regions. We calculate the runtime for this stage as the maximum value among the runtimes for each region. We take the maximum, not the summation, because we run these in parallel using our computing cluster; if we ran the experiments sequentially, the total runtime for the first stage on instance 6 would be 717,754 seconds, 260,445 seconds, and 61,280 seconds for handling costs of 0.01, 0.2, and 0.4, respectively. The second stage runtime is simply the time taken for the crossing commodity algorithm. We also report the runtimes of the polishing algorithm for instances 5 and 6 and provide a detailed discussion in Appendix A.4.

5.4 Low-Volume Instance

To verify the robustness of our method, we generate a low-volume instance representing weekend demand. Analysis of historical data from our industry partner shows that weekend and weekday demand data sets had nearly identical portfolios in terms of origins, destinations, and service time guarantees, with the only difference being package volume. Based on this observation, we scale down the volume of each commodity in the instance from Section 5.1 by a factor of five and then remove commodities with volumes lower than the 5th percentile.

As in Section 5.3, we evaluate the performance of our method on this low-volume instance against the baseline and Gurobi. Tables 9 and 10 summarize the results on small and medium/large instances, respectively. It is noticeable that all methods show relatively higher LB gaps compared to Tables 6 and 7. This may be due to weak lower bounds and the difficulty of consolidating low-volume commodities. For example, it is inevitable to under-utilize some trailers in this low-volume instance to meet the service time guarantees. Despite this additional challenge, our method finds high-quality solutions, outperforming Gurobi in around half of the handling cost scenarios, and requiring 8.7 hours or less on small instances. On the large instance, it takes up to 22.6 hours to find a solution when the handling cost is close to zero. The proposed method outperforms the baseline in all instances, which demonstrates its robustness against varying demand conditions.

I	f^H	Method	Obj	LB gap (%)	Tcost	Hcost	APL	Util	Time (s)
1	0.01	**Proposed**	**52,322**	**55.84**	**50,645**	**1,677**	**3.08**	**0.47**	**22,895**
		Baseline	239,088	90.34	238,366	722	1.34	0.06	-
		Gurobi	54,318	57.46	52,968	1,350	2.53	0.31	86,400
	0.2	Proposed	84,235	43.96	56,341	27,894	2.61	0.36	23,274
		Baseline	252,805	81.33	238,366	14,439	1.34	0.06	-
		Gurobi	**81,412**	**42.01**	**61,533**	**19,878**	**1.90**	**0.26**	**86,400**
	0.4	**Proposed**	**105,711**	**42.64**	**55,228**	**50,483**	**2.37**	**0.36**	**22,651**
		Baseline	267,243	77.31	238,366	28,878	1.34	0.06	-
		Gurobi	182,102	66.70	152,674	29,428	1.51	0.10	86,400
2	0.01	**Proposed**	**72,267**	**36.46**	**64,703**	**7,564**	**2.13**	**0.68**	**30,223**
		Baseline	160,467	71.38	156,040	4,427	1.18	0.25	-
		Gurobi	95,976	52.15	90,656	5,320	1.54	0.44	86,400
	0.2	**Proposed**	**175,134**	**15.60**	**69,587**	**105,547**	**1.57**	**0.59**	**31,172**
		Baseline	244,583	39.57	156,040	88,543	1.18	0.25	-
		Gurobi	224,471	34.15	131,884	92,588	1.43	0.30	86,400
	0.4	Proposed	271,824	8.56	79,487	192,337	1.47	0.51	21,359
		Baseline	333,126	25.38	156,040	177,086	1.18	0.25	-
		Gurobi	**265,297**	**6.31**	**83,597**	**181,700**	**1.42**	**0.48**	**86,400**
3	0.01	Proposed	49,287	21.89	40,730	8,557	2.14	0.80	17,340
		Baseline	86,808	55.65	81,918	4,891	1.24	0.33	-
		Gurobi	**43,458**	**11.41**	**36,485**	**6,973**	**1.79**	**0.78**	**86,400**
	0.2	Proposed	161,123	7.94	48,773	112,349	1.48	0.59	8,022
		Baseline	179,730	17.47	81,918	97,812	1.24	0.33	-
		Gurobi	**158,449**	**6.39**	**56,351**	**102,098**	**1.41**	**0.49**	**86,400**
	0.4	Proposed	261,553	4.68	61,629	199,925	1.40	0.45	5,769
		Baseline	277,542	10.17	81,918	195,625	1.24	0.33	-
		Gurobi	**257,813**	**3.30**	**64,450**	**193,363**	**1.38**	**0.42**	**86,400**

Table 9: Results obtained by the proposed method, baseline, and Gurobi on small low-volume instances

I	f^H	Method	Obj	LB gap (%)	Tcost	Hcost	APL	Util	Time (s)
4	0.01	**Proposed**	**522,226**	**26.60**	**481,810**	**40,416**	**2.64**	**0.89**	**49,781**
		Baseline	957,275	59.96	937,526	19,749	1.28	0.40	-
	0.2	**Proposed**	**1,020,342**	**32.49**	**484,682**	**535,660**	**1.82**	**0.83**	**40,127**
		Baseline	1,332,503	48.31	937,526	394,977	1.28	0.40	-
	0.4	**Proposed**	**1,457,850**	**30.70**	**523,342**	**934,509**	**1.64**	**0.76**	**28,820**
		Baseline	1,727,480	41.51	937,526	789,954	1.28	0.40	-
5	0.01	**Proposed**	**385,360**	**68.88**	**370,628**	**14,732**	**2.30**	**0.40**	**31,424**
		Baseline	665,463	81.98	656,751	8,712	1.30	0.19	-
	0.2	**Proposed**	**592,666**	**57.79**	**374,379**	**218,287**	**1.77**	**0.35**	**31,497**
		Baseline	830,986	69.89	656,751	174,235	1.30	0.19	-
	0.4	**Proposed**	**800,065**	**51.59**	**402,994**	**397,071**	**1.64**	**0.32**	**22,914**
		Baseline	1,005,221	61.47	656,751	348,470	1.30	0.19	-
6	0.01	**Proposed**	**1,917,965**	**32.13**	**1,764,045**	**153,920**	**2.96**	**0.84**	**81,474**
		Baseline	3,536,821	63.19	3,465,358	71,463	1.37	0.38	-
	0.2	**Proposed**	**3,861,173**	**39.58**	**1,874,617**	**1,986,556**	**1.99**	**0.75**	**49,268**
		Baseline	4,894,621	52.34	3,465,358	1,429,263	1.37	0.38	-
	0.4	**Proposed**	**5,569,080**	**38.62**	**2,137,866**	**3,431,214**	**1.76**	**0.64**	**34,819**
		Baseline	6,323,883	45.95	3,465,358	2,858,525	1.37	0.38	-

Table 10: Results obtained by the proposed method and baseline on medium/large low-volume instances

6 Computational Study: Random Data

To verify the effectiveness of our approach on other instances with different network structure and demand patterns, we design a random instance based on geographical information from Mexico. These experiments use the same computational environment setup as in Section 5. We describe how we generate the random instance in Section 6.1 and report on performance in Section 6.2.

6.1 Instance

To generate a terminal network for Mexico, we create a terminal for every city with population over 50,000, using the city's central coordinate as the terminal's location. Based on World Population Review (2024), this yields 225 terminals. The travel times and distances between terminals are given by OpenStreetMap using the Open Source Routing Machine API (Giraud 2022). For the labor shifts, we use the same four shifts as in our previous instance.

Next, we generate 40,000 commodities by randomly selecting origin and destination nodes, and a service time guarantee. We randomly choose origin and destination terminals with probability proportional to the corresponding city's population. If the selected terminals are identical, we repeat the selection process.

Next, we randomly choose origin and destination labor shifts, and a service time guarantee based on the distribution of service time guarantees in our real-world data. To ensure feasibility, we compute the shortest transit time (in days) between the selected origin and destination nodes and then randomly select a service time guarantee among the values greater than or equal to this value. The tuple of (origin node, destination node, service time guarantee) defines a commodity; if the generated commodity is redundant, we discard it and repeat the procedure until we have 40,000 distinct commodities.

Last, we distribute a total of 25 million cubic feet per day of package volume over commodities by assuming that the volume from origin o to destination d is proportional to the sum of total monthly salaries in o and d. We obtain salary data from Statista Research Department (2024) and populations from World Population Review (2024). To add some randomness, we multiply each commodity's volume by a random factor in $(0.5, 1.5)$. Finally, we ensure that each commodity volume is at least 3% of the trailer capacity by discarding low-volume commodities.

6.2 Benchmarking

We apply our proposed approach to solve the random instance. First, we partition the network into four regions as shown in Figure 7 (blue, light blue, green, and orange); this partition is based on Álvarez Carmona et al. (2020), Moye-Holz et al. (2018), since there is no official division of Mexican states into regions similar to what the US Census uses. We also consider smaller sub-region partitions within each region. For each region, we solve the problem using the batching algorithm with four different random seeds. After obtaining solutions for all regions, we fix the best solution for each region and then run the crossing commodity method. We consider the following configurations: $(\alpha, \beta, \gamma) \in \{(50,25,3),(50,30,3),(60,25,3)\}$. We run the experiments with four different handling costs (0.01, 0.1, 0.2, and 0.4) and report the best results in Table 11. All metrics are the same as in Table 7.

f^H	Method	Obj	LB gap (%)	Tcost	Hcost	APL	Util	Time (s)
0.01	Proposed	3,132,069	71.63	2,596,343	535,725	2.69	0.78	190,806
0.1	Proposed	8,073,945	37.89	3,927,676	4,146,269	2.15	0.64	171,805
0.2	Proposed	12,587,057	30.67	5,150,015	7,437,041	1.88	0.60	112,481
0.4	Proposed	19,966,540	25.03	6,599,785	13,366,755	1.72	0.58	166,871

Table 11: Results obtained by the proposed method on random instance

As shown in Table 11, the proposed method achieves a gap of 25% in 46 hours when the handling cost

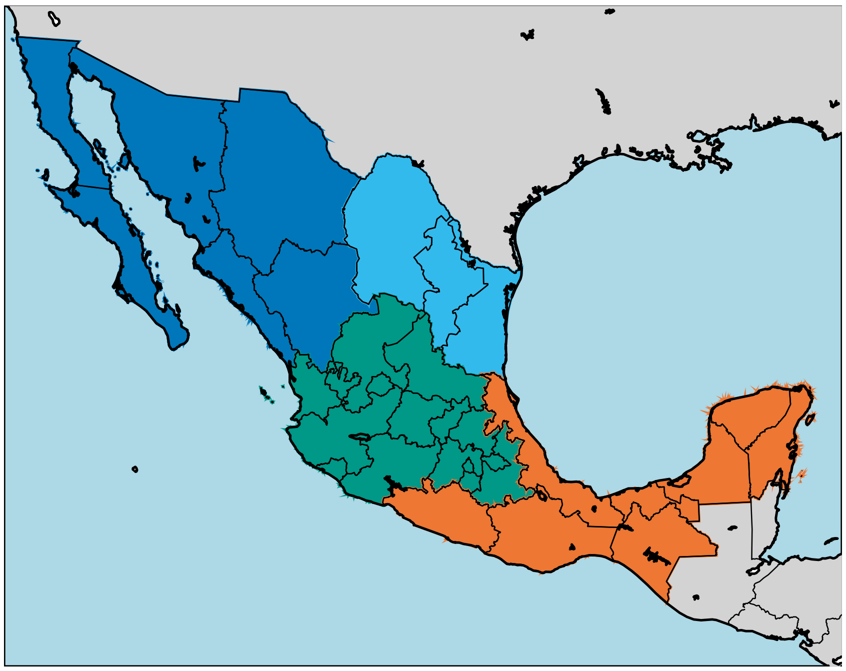


Figure 7: Four regions in Mexico.

is 0.4. With smaller handling costs, the gaps and runtimes increase, reaching up to 38% and 48 hours, respectively. In addition, the average path length and trailer utilization decrease as the handling cost increases, as seen in Table 7. Compared to Table 7, the average path length is larger, which may be related to the geography of Mexico. In summary, this computational study demonstrates that our proposed approach is effective and scalable on instances with diverse network structures and demand distributions.

7 Conclusion

We studied an industrial-scale service network design problem with service time guarantees. While an IP formulation for this problem is extremely large and computationally intractable, we find high-quality solutions with a recursive partitioning and batching algorithm. Our proposed approach is based on two main intuitions: (1) it is preferable to avoid excessive detours when designing paths for commodities, and (2) fixing paths for commodities with less flexibility first will likely lead to consolidation. Our computational study on real-world instances demonstrates the scalability and efficiency of our method. Specifically, we find solutions in under four hours on small instances, sometimes outperforming the best solution obtained by Gurobi with a 24-hour time limit. More importantly, for the largest instance covering the entire US, we find solutions in just over three hours, achieving a plan with 10% cost savings compared to an industrial benchmark. Furthermore, the method is also effective and scalable on randomly generated instances, which vary in package volumes, network structure, and demand distribution.

A natural extension is to incorporate the management of resources, such as trailers and drivers. In our model, we assume such resources are always available; however, the number of resources available at a terminal may in fact be limited. In this case, the optimization model would require resource capacity constraints and balance constraints, which for instance equalize the total number of inbound and outbound trailers at a terminal. Methodologically, we could improve the algorithm by allowing paths chosen in earlier iterations to be altered later. This adjustment would require careful modifications to avoid significantly increasing the computational burden.

References

C. J. Alpert and S.-Z. Yao. Spectral partitioning: The more eigenvectors, the better. In *Proceedings of the 32nd annual ACM/IEEE design automation conference*, pages 195–200, 1995.

M. Á. Álvarez Carmona, E. Villatoro Tello, M. Montes y Gómez, and L. Vilaseñor Pineda. Author profiling in social media with multimodal information. *Computación y Sistemas*, 24(3):1289–1304, 2020.

K. Andreev and H. Räcke. Balanced graph partitioning. In *Proceedings of the sixteenth annual ACM symposium on Parallelism in algorithms and architectures*, pages 120–124, 2004.

F. Bacao, V. Lobo, and M. Painho. Applying genetic algorithms to zone design. *Soft Computing*, 9:341–348, 2005.

R. Bai, S. W. Wallace, J. Li, and A. Y.-L. Chong. Stochastic service network design with rerouting. *Transportation Research Part B: Methodological*, 60:50–65, 2014.

I. Bakir, A. Erera, and M. Savelsbergh. Motor carrier service network design. *Network design with applications to transportation and logistics*, pages 427–467, 2021.

A. Balakrishnan, G. Li, and P. Mirchandani. Optimal network design with end-to-end service requirements. *Operations Research*, 65(3):729–750, 2017.

A. Baubaid, N. Boland, and M. Savelsbergh. The value of limited flexibility in service network designs. *Transportation Science*, 55(1):52–74, 2021.

N. Boland, M. Hewitt, L. Marshall, and M. Savelsbergh. The continuous-time service network design problem. *Operations Research*, 65(5):1303–1321, 2017.

N. Boland, M. Hewitt, L. Marshall, and M. Savelsbergh. The price of discretizing time: a study in service network design. *EURO Journal on Transportation and Logistics*, 8(2):195–216, 2019.

B. Bozkaya, E. Erkut, and G. Laporte. A tabu search heuristic and adaptive memory procedure for political districting. *European Journal of Operational Research*, 144(1):12–26, 2003.

T. G. Crainic. Service network design in freight transportation. *European Journal of Operational Research*, 122(2): 272–288, 2000.

T. G. Crainic and M. Gendreau. Cooperative parallel tabu search for capacitated network design. *Journal of Heuristics*, 8:601–627, 2002.

T. G. Crainic, B. Gendron, and G. Hernu. A slope scaling/lagrangean perturbation heuristic with long-term memory for multicommodity capacitated fixed-charge network design. *Journal of Heuristics*, 10:525–545, 2004.

T. G. Crainic, M. Hewitt, M. Toulouse, and D. M. Vu. Service network design with resource constraints. *Transportation Science*, 50(4):1380–1393, 2016.

T. G. Crainic, M. Hewitt, M. Toulouse, and D. M. Vu. Scheduled service network design with resource acquisition and management. *EURO Journal on Transportation and Logistics*, 7:277–309, 2018.

T. G. Crainic, M. Gendreau, and B. Gendron, editors. *Network Design with Applications to Transportation and Logistics*. Springer Cham, 2021. ISBN 978-3-030-64018-7. doi: 10.1007/978-3-030-64018-7. URL `https://link.springer.com/book/10.1007/978-3-030-64018-7`.

A. Erera, M. Hewitt, M. Savelsbergh, and Y. Zhang. Improved load plan design through integer programming based local search. *Transportation Science*, 47(3):412–427, 2013.

I. Ghamlouche, T. G. Crainic, and M. Gendreau. Cycle-based neighbourhoods for fixed-charge capacitated multicommodity network design. *Operations Research*, 51(4):655–667, 2003.

I. Ghamlouche, T. G. Crainic, and M. Gendreau. Path relinking, cycle-based neighbourhoods and capacitated multicommodity network design. *Annals of Operations Research*, 131:109–133, 2004.

T. Giraud. osrm: Interface Between R and the OpenStreetMap-Based Routing Service OSRM. *Journal of Open Source Software*, 7(78):4574, aug 2022.

B. Hendrickson, R. W. Leland, et al. A multi-level algorithm for partitioning graphs. In *Supercomputing '95:Proceedings of the 1995 ACM/IEEE Conference on Supercomputing*, volume 95, pages 1–14, 1995.

M. Hewitt. Enhanced dynamic discretization discovery for the continuous time load plan design problem. *Transportation Science*, 53(6):1731–1750, 2019.

M. Hewitt. The flexible scheduled service network design problem. *Transportation Science*, 56(4):1000–1021, 2022.

M. Hewitt and F. Lehuédé. New formulations for the scheduled service network design problem. *Transportation Research Part B: Methodological*, 172:117–133, 2023.

M. Hewitt, T. G. Crainic, M. Nowak, and W. Rei. Scheduled service network design with resource acquisition and management under uncertainty. *Transportation Research Part B: Methodological*, 128:324–343, 2019.

A. I. Jarrah, E. Johnson, and L. C. Neubert. Large-scale, less-than-truckload service network design. *Operations Research*, 57(3):609–625, 2009.

G. Karypis and V. Kumar. Metis: A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices. Technical report, University of Minnesota, 1997.

D. Kim, C. Barnhart, K. Ware, and G. Reinhardt. Multimodal express package delivery: A service network design application. *Transportation Science*, 33(4):391–407, 1999.

C. L. Lara, J. Koenemann, Y. Nie, and C. C. de Souza. Scalable timing-aware network design via lagrangian decomposition. *European Journal of Operational Research*, 309(1):152–169, 2023.

K. Lindsey, A. Erera, and M. Savelsbergh. Improved integer programming-based neighborhood search for less-than-truckload load plan design. *Transportation Science*, 50(4):1360–1379, 2016.

G. Lulli, U. Pietropaoli, and N. Ricciardi. Service network design for freight railway transportation: the italian case. *Journal of the Operational Research Society*, 62(12):2107–2119, 2011.

L. Marshall, N. Boland, M. Savelsbergh, and M. Hewitt. Interval-based dynamic discretization discovery for solving the continuous-time service network design problem. *Transportation Science*, 55(1):29–51, 2021.

A. Mauttone and M. E. Urquhart. A route set construction algorithm for the transit network design problem. *Computers & Operations Research*, 36(8):2440–2449, 2009.

D. Moye-Holz, R. Soria Saucedo, J. P. van Dijk, S. A. Reijneveld, and H. V. Hogerzeil. Access to innovative cancer medicines in a middle-income country-the case of Mexico. *Journal of Pharmaceutical Policy and Practice*, 11 (1):25, 2018.

H. N. Newton, C. Barnhart, and P. H. Vance. Constructing railroad blocking plans to minimize handling costs. *Transportation Science*, 32(4):330–345, 1998.

S. Pattnaik, S. Mohan, and V. Tom. Urban bus transit route network design using genetic algorithm. *Journal of Transportation Engineering*, 124(4):368–375, 1998.

W. B. Powell. A local improvement heuristic for the design of less-than-truckload motor carrier networks. *Transportation Science*, 20(4):246–257, 1986.

W. B. Powell and Y. Sheffi. The load planning problem of motor carriers: Problem description and a proposed solution approach. *Transportation Research Part A: General*, 17(6):471–480, 1983.

F. Ricca, A. Scozzari, and B. Simeone. Weighted voronoi region algorithms for political districting. *Mathematical and Computer Modelling*, 48(9-10):1468–1477, 2008.

Y. O. Scherr, M. Hewitt, B. A. N. Saavedra, and D. C. Mattfeld. Dynamic discretization discovery for the service network design problem with mixed autonomous fleets. *Transportation Research Part B: Methodological*, 141: 164–195, 2020.

Statista Research Department. Average monthly salary in mexico 2023, by state. `https://www.statista.com/statistics/1399150/average-monthly-salary-by-sate-mexico/`, 2024. Accessed: 2024-12-01.

United States Census Bureau. Census regions and divisions of the united states. `https://www.census.gov/geo/pdfs/mapsdata/maps/reference/us_regdiv.pdf`, 2013. Accessed: 2024-08-12.

UPS. Ups 2023 annual report. `https://investors.ups.com/`, 2023. Accessed: 2024-06-16.

H. Üster and H. Agrahari. An integrated load-planning problem with intermediate consolidated truckload assignments. *IIE Transactions*, 42(7):490–513, 2010.

H. Üster and N. Maheshwari. Strategic network design for multi-zone truckload shipments. *IIE Transactions*, 39(2): 177–189, 2007.

P. van't Hof, D. Paulusma, and G. J. Woeginger. Partitioning graphs into connected parts. *Theoretical Computer Science*, 410(47-49):4834–4843, 2009.

Z. Wang and M. Qi. Robust service network design under demand uncertainty. *Transportation Science*, 54(3):676–689, 2020.

N. Wieberneit. Service network design for freight transportation: a review. *OR Spectrum*, 30(1):77–112, 2008.

World Population Review. Mexico cities by population 2024. `https://worldpopulationreview.com/cities/mexico`, 2024. Accessed: 2024-12-01.

E. Zhu, T. G. Crainic, and M. Gendreau. Scheduled service network design for freight rail transportation. *Operations Research*, 62(2):383–400, 2014.

A Appendix

A.1 Comparison between zero and nearly-zero handling costs

To understand the impact of positive handling costs, we test our proposed method with zero and nearly-zero (0.01) handling costs and report the results in Table 12. In the table, I refers to the instance and f^H represents the handling cost. *Obj*, *Tcost*, and *Hcost* represent the objective, transportation, and handling costs, respectively. *APL* represents the volume-weighted average path length, while *Util* represents the distance-weighted trailer utilization. Finally, *Time* refers to the runtime in seconds.

I	f^H	Obj	Tcost	Hcost	APL	Util	Time (s)
1	0.0	110,581	110,581	0	2.86	0.75	29,802
	0.01	110,276	103,984	6,292	2.38	0.79	9,420
2	0.0	222,733	222,733	0	2.39	0.90	18,345
	0.01	243,037	215,595	27,442	1.61	0.93	8,811
3	0.0	146,330	146,330	0	210	0.93	40,550
	0.01	172,436	144,442	27,994	1.51	0.92	10,391

Table 12: Comparison between zero and nearly-zero handling costs on small instances

Comparing the results between zero and nearly-zero handling costs for each instance, we see that a positive but small handling cost decreases the transportation cost and the runtime (Table 12). One possible explanation for this result that the handling cost acts as a small regularization term to the objective function. Motivated by this observation, we run our experiments with a handling cost of 0.01 instead of zero.

A.2 Omitted results of Section 5.2.1

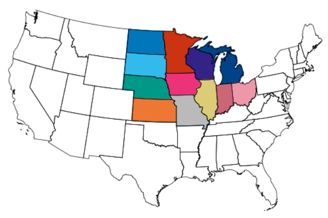
Table 13 summarizes the results obtained with different random seeds on instance 3.

A.3 Omitted results of Section 5.2.2

Figures 8, 9 and 10 depict sub-region partitioning options for Midwest, Northeast, and South regions (instances 2, 3, and 4), respectively. We run the algorithm with different sub-region partitions on instance 3 and compare the results. Table 14 summarizes the results obtained using partitioning options in Figure 9.

Instance	f^H	Seed	Obj	Solution Gap (%)	APL	Util	Time (s)
3	0.01	-	172,923	0.28	1.52	0.92	4,806
		1	172,600	0.09	1.50	0.92	4,628
		2	172,708	0.16	1.48	0.91	4,300
		3	**172,436**	**0.00**	**1.51**	**0.92**	**4,212**
	0.2	-	615,546	0.06	1.27	0.77	536
		1	615,236	0.01	1.27	0.78	538
		2	615,601	0.07	1.27	0.78	561
		3	**615,190**	**0.00**	**1.27**	**0.78**	**552**
	0.4	-	1,055,759	0.07	1.23	0.67	410
		1	1,055,667	0.06	1.24	0.67	455
		2	**1,055,020**	**0.00**	**1.23**	**0.67**	**451**
		3	1,055,602	0.06	1.23	0.67	406

Table 13: Results obtained by the proposed method with different random seeds and the deterministic version of the method on instance 3


(a) Option 1

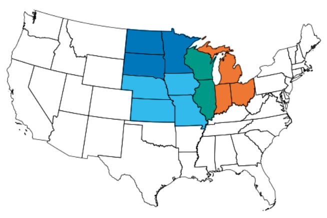
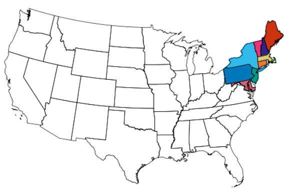
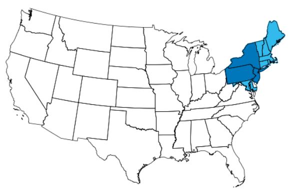

(b) Option 2

Figure 8: Two sub-region partitioning options for Midwest region.

A.4 Effectiveness of Polishing Algorithm

We compare the final solutions before and after applying the polishing algorithm on instances 5 and 6. The *Improvement* column reports the decrease in the objective value. The polishing algorithm improves the solution by more than 1% in under a minute on instance 5. Since the reoptimization requires very little time, perhaps we could achieve additional improvement by adopting a higher arc-utilization threshold in Algorithm 4. For instance 6, the algorithm reduces the objective by between 2% and 3% within 20 minutes when the handling costs are positive. Notably, the algorithm achieves more than 5% of improvement in 40 minutes when the handling cost is nearly zero. These results demonstrate the effectiveness of the polishing algorithm.


(a) Option 1


(b) Option 2

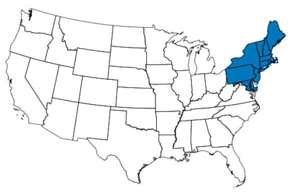
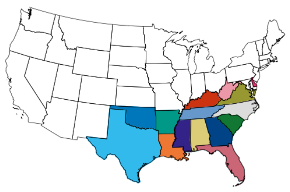
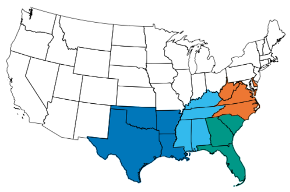

(c) Option 3

Figure 9: Three sub-region partitioning options for Northeast region.


(a) Option 1


(b) Option 2

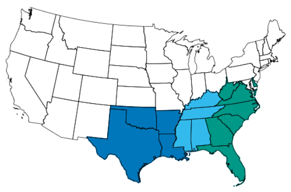

(c) Option 3

Figure 10: Three sub-region partitioning options for South region.

Instance	f^H	Partition	Obj	Solution Gap (%)	APL	Util	Time (s)
3	0.01	**1**	**172,436**	**0.00**	**1.51**	**0.92**	**4,212**
		2	174,842	1.38	1.59	0.94	7,531
		3	174,373	1.11	1.62	0.95	9,439
	0.2	**1**	**615,190**	**0.00**	**1.27**	**0.78**	**552**
		2	617,303	0.34	1.25	0.78	622
		3	619,662	0.72	1.25	0.79	1,061
	0.4	**1**	**1,055,020**	**0.00**	**1.23**	**0.67**	**451**
		2	1,057,255	0.21	1.23	0.67	546
		3	1,060,295	0.50	1.21	0.66	1,011

Table 14: Results obtained by the proposed method with different sub-region partitions on instance 3

I	f^H	Result	Obj	Improvement (%)	Tcost	Hcost	APL	Util	Time (s)
5	0.01	Unpolished	767,405	-	707,559	59,846	1.91	0.87	-
		Polished	757,352	1.31	702,369	54,984	1.77	0.87	57
	0.2	Unpolished	1,609,127	-	763,249	845,877	1.39	0.78	-
		Polished	1,564,682	2.76	717,350	847,332	1.40	0.83	45
	0.4	Unpolished	2,416,379	-	813,691	1,602,688	1.35	0.72	-
		Polished	2,372,090	1.83	766,988	1,605,102	1.35	0.77	44
6	0.01	Unpolished	7,856,862	-	7,403,346	453,516	1.82	0.89	-
		Polished	7,428,697	5.45	6,973,879	454,818	1.82	0.95	2,339
	0.2	Unpolished	14,332,472	-	7,604,040	6,728,432	1.38	0.85	-
		Polished	13,899,950	3.02	7,146,477	6,753,473	1.38	0.91	1,291
	0.4	Unpolished	20,797,010	-	8,027,015	12,769,995	1.31	0.80	-
		Polished	20,350,978	2.14	7,530,234	12,820,744	1.32	0.86	1,321

Table 15: Results before and after applying the polishing algorithm on instances 5 and 6.