

Large Scale Middle Mile Network Design Through Efficient Local Search

Carlo S. Sartori ✉ 

Amazon Science and Technology, Luxembourg

Jian Shen ✉ 

Amazon Science and Technology, London, UK

Anastasia Kireeva ✉ 

ETH Zurich, Switzerland

Mattia Neroni ✉ 

Amazon Science and Technology, Barcelona, Spain

Philipp Loick ✉ 

Amazon Science and Technology, Luxembourg

Abstract

E-commerce competition has intensified in recent years, further raising the delivery speed and cost efficiency standards that companies must meet. The middle mile network that connects fulfillment centers to local distribution centers plays a decisive role in achieving these standards. In this paper, we study the Middle Mile Network Design Problem, which determines optimal network investments that balance the trade-off between fastest delivery speed and minimal operational costs. The problem requires computing truck schedules, shipment routes, and parcel flows through intermediate facilities subject to complex capacity constraints. Two key challenges make this problem computationally hard to solve for standard exact and heuristic approaches: (i) the use of complex black-box evaluations to predict operational outcomes, and (ii) problem instances involving over 200,000 commodities that must be solved within hours for operational planning. We propose a metaheuristic based on variable neighborhood search with specialized local search operators alongside lazy neighborhood evaluation that enable efficient exploration of the high-dimensional solution space. Our algorithm achieves up to 7.6% solution improvement over an incumbent heuristic while reducing runtime by up to 44%.

2012 ACM Subject Classification Applied computing → Transportation

Keywords and phrases Network design, Middle mile logistics, Black-box evaluation, Local search, Variable Neighborhood Search

Digital Object Identifier 10.4230/OASICS.ATMOS.2026.15

Acknowledgements We would like to thank the three anonymous reviewers for their valuable feedback. Anastasia Kireeva has contributed to this paper during her internship at Amazon.

1 Introduction

Worldwide e-commerce transported 110 billion parcels in 2024 [19], generating six trillion dollars in sales - a figure projected to reach eight trillion by 2028 [7]. This explosive growth has further intensified pressure in an already highly competitive market, requiring carriers to simultaneously improve delivery speed and reduce operational costs to maintain market share and sustain growth.

Recent studies show customers prefer delivery promises of two days or less [8]. It is not surprising then that the average delivery times in the United States fell from 6.5 days in October 2020 to 4.4 days by July 2022, with Amazon leading in delivery speed with 1.9 days [18]. However, post-COVID consumers have grown equally attentive to shipping costs [11]. Even if reducing delivery promises by one day can increase sales by 13%, the



© Carlo S. Sartori, Jian Shen, Anastasia Kireeva, Mattia Neroni, and Philipp Loick; licensed under Creative Commons License CC-BY 4.0

26th Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2026).

Editors: Valentina Cacchiani and Stefan Funke; Article No. 15; pp. 15:1–15:18



OpenAccess Series in Informatics

OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

operational costs to achieve such speed improvements may be prohibitively expensive and defeat the purpose [10]. This raises the fundamental question for carriers of whether the investments are worth it, and more specifically, *which* investments are the right ones.

This paper addresses the core challenge in e-commerce of designing a middle mile transportation network that is both fast and cost-efficient. The middle mile network refers to moving parcels from fulfillment centers (FCs), where items are stored, to delivery centers (DCs), where last mile delivery begins to the final customer. The speed of the middle mile network directly impacts both customer delivery promises and operational costs. A theoretical network connecting all FCs directly to all DCs could maximize speed, but prove prohibitively expensive to operate. Instead, companies consolidate shipments through intermediate sort centers (SCs) to reduce costs, while maintaining some direct connections for speed.

Designing such a network requires determining truck departures between sites, which legs of the network are available for shipment and how parcels travel through intermediate SCs. All of this while respecting the resource limits of the network such as site’s physical and labor capacities. We call this the *Middle Mile Network Design Problem* (MMNDP), which has been first studied by [4]. The MMNDP seeks to maximize speed while minimizing cost. We focus on Amazon’s context, where a black-box evaluation function maps a given network design to a speed metric that can be translated into expected financial gain, enabling direct tradeoff against the operational cost of achieving that speed [4]. Cost encompasses two components: linehaul transportation (trucks) and sorting operations at SCs. The MMNDP’s high-dimensional decision space, black-box objective function, and large-scale real-world instances make exact methods computationally intractable. The networks we consider involve up to 200,000 commodity flows, whereas exact approaches in the literature typically handle only a few thousand commodities. Moreover, even simpler variants of the MMNDP, such as capacitated multi-commodity network flow, are NP-Hard [15].

Our work extends Benidis et al. [2] and Brubaker et al. [4] by addressing real-world problem complexity at industrial scale. We handle instances at least ten times larger than those in prior academic literature while incorporating the full complexity of operational constraints and black-box objective functions. To achieve this scale, we design a metaheuristic based on variable neighborhood descent with specialized local search operators, lazy neighborhood evaluation, and move caching mechanisms that enable the algorithm to quickly produce high-quality solutions. Our work also addresses a recognized gap in network design research, where proper metaheuristic approaches remain underdeveloped [20]. Moreover, because many real-world optimization problems share similar characteristics (high-dimensional search space, large-scale instances, and black-box objectives), our algorithmic contributions are broadly applicable beyond the scope of middle mile network design.

The remainder of the paper is structured as follows. Section 2 describes the problem and provides illustrative examples to help understand the decision space. We then review the literature of network design problems in Section 3. Section 4 presents our algorithmic approach, including local search moves and computational techniques employed to speed up runtime. We discuss experiments and results in Section 5. Finally, we conclude the paper and provide future work directions in Section 6.

2 Problem

The Middle Mile Network Design Problem (MMNDP) is defined on a directed graph $G = (V, A)$ where V represents network sites and A represents arcs connecting these sites, each with an associated transit time. Sites are partitioned into fulfillment centers $F \subset V$, sort centers $S \subset V$, and delivery centers $D \subset V$.

Volume flow through the network is represented by a set of commodities K . Each commodity $k \in K$ has an origin $o_k \in F$, a destination $d_k \in D$, and a volume v_k (number of packages) expected to reach d_k . Each delivery center d_k has a critical injection time, defining the latest arrival time for packages to be delivered on the same day via last mile operations (e.g., 10:00 AM). While planning spans a week, we describe a single day for simplicity. Time is discretized into 15-minute intervals (buckets), with $T = \{1, \dots, 96\}$ denoting the 96 daily buckets.

Decisions

The MMNDP then consists of making four simultaneous decisions to optimize the network for transporting these commodities

1. **Schedule Fixed Truck Departures (FTDs):** An FTD represents a guaranteed truck departure from site i to site j at a specific time on arc $(i, j) \in A$. FTDs depart regardless of volume, even if empty. When total volume exceeds truck capacity, additional ad-hoc trucks are scheduled to transport excess volume earlier. These ad-hoc trucks are not decision variables, but rather operational responses simulated during network evaluation based on historical information about volume availability for each point in time. The ad-hoc trucks also define the profile of volume arrival at a site (SC or DC) which directly impacts labor capacity constraints. An arc may have multiple FTDs, subject to operational constraints limiting their number per arc. Formally, we define a binary variable x_{ij}^t that takes value 1 when an FTD is placed at time bucket $t \in T$ on arc $(i, j) \in A$.
2. **Select active arcs:** An arc is active if at least one commodity uses it and it contains at least one FTD - alternatively, a commodity can only use an active arc and an FTD can only be placed on an active arc. We denote with binary variable $y_{ij} = 1$ whether arc $(i, j) \in A$ is active.
3. **Determine sort center operations:** For paths containing at least one sort center, we decide whether packages are sorted or cross-docked at each intermediate facility. Sorting allows consolidation: packages from multiple commodities originating at o_k can be placed on shared pallets for transportation, then separated at the sort center. This increases processing time and costs at the sort center but improves consolidation. Cross-docking transfers packages directly without sortation, reducing processing time but requiring origin facilities to presort packages into destination-specific containers. This consumes additional resources at fulfillment centers and may reduce truck fill rates. We denote with binary variable $z_{ij} = 1$ whether the (pre)sort between facilities $i \in V$ and $j \in V$ is used (note that i and j are not necessarily an arc in case of cross-docking).
4. **Route commodities through the network:** Each commodity k is assigned a path p_k from origin o_k to destination d_k using active arcs in A . Paths are classified by the number of intermediate sort centers: *direct* paths connect origin to destination without intermediate stops ($p_k = (o_k, d_k)$), *one-touch* paths use a single sort center ($p_k = (o_k, s, d_k)$ for $s \in S$), and *multi-touch* paths use two or more sort centers. Furthermore, each commodity path must use *exactly* one FTD per arc in p_k as well as define sort center operations. To that end, we denote with binary variables $\psi_{ij}^k \leq y_{ij}$ whether commodity $k \in K$ uses arc y_{ij} in its path p_k , for $(i, j) \in A$; with variable $\chi_{ij}^{kt} \leq x_{ij}^t$ whether k uses FTD x_{ij}^t ; and with variable $\zeta_{ij}^k \leq z_{ij}$ whether k uses sort z_{ij} .

Constraints

All four decisions have to be made while accounting for the following constraints.

- C1** *Shift opening hours*: Each site has operating hours defining when trucks may depart (FCs and SCs) or arrive (SCs and DCs).
- C2** *Minimum time difference between two FTDs on the same arc*: Multiple FTDs on the same arc must be separated by a minimum time interval (e.g., 1 hour).
- C3** *Maximum FTDs departure and arrival per hour*: Sites can accommodate only a limited number of truck arrivals or departures per hour due to yard, dock door, and labor constraints. This is enforced by limiting inbound and outbound FTDs within any one-hour window.
- C4** *Labor capacity at SCs and DCs*: Each site can process a limited volume per time bucket (15-minute interval), constraining when and how much volume can arrive due to labor limitations. This prevents scheduling all FTDs late in the shift, which would create arrival peaks exceeding processing capacity before outbound connections. At SCs, outbound connections are downstream FTDs continuing middle mile transportation; at DCs, outbound connections represent last mile delivery. This is one of the most complex constraints because it requires simulating, via another black-box evaluation, how ad-hoc trucks are scheduled for each FTD to bring volume earlier for each intermediate site (more information in Appendix A).
- C5** *Sort capacity at FCs*: FCs must reserve physical space for outbound pallets. Sorted volume destined for a SC typically requires one space regardless of how many commodities share the pallet. However, cross-docked volume requires separate spaces for each final destination, as packages must be pre-sorted at the FC rather than consolidated at the SC. This increases FC resource consumption.

The actual set of constraints in our real problem is much broader than (C1)-(C5). However, we define this subset with the main constraints for brevity. Beyond constraints (C1)-(C5), we also require mathematical constraints to ensure solution validity. Each commodity k uses exactly one FTD per arc: $\sum_{t \in T} \chi_{ij}^{kt} = 1$ for all $(i, j) \in p_k$. Similarly, sort operations cannot overlap on sub-paths: for any $i, j, l \in V$ where l appears between i and j in p_k , if $\zeta_{ij}^k = 1$, then $\zeta_{il}^k = 0$ and $\zeta_{lj}^k = 0$. We omit the full mathematical formulation for brevity; appendix A provides a mathematical description of constraints (C1)-(C5). Interested readers are referred to [4] for a more detailed model.

We denote by σ a solution to the MMNDP and by $\mathcal{X}$ the set of all *feasible solutions* to the problem, for which the assignment of decision variables comply with constraints (C1)-(C5). Unlike typical academic studies that assume empty initial networks, we assume an initial solution σ_0 representing the current operational network. This reflects the practical setting where planners incrementally improve the existing network design. Solution quality is assessed via the objective function defined next.

Objective function

The MMNDP maximizes the expected entitlement $f(\sigma)$ of the designed network σ :

$$\max_{\sigma \in \mathcal{X}} f(\sigma) = \mathcal{S}(\sigma) - \mathcal{L}(\sigma) - \mathcal{C}(\sigma) \quad (1)$$

where $\mathcal{S}(\sigma)$ represents the monetary value of network speed, $\mathcal{L}(\sigma)$ represents linehaul transportation costs, and $\mathcal{C}(\sigma)$ represents sort center processing costs. Let us now turn to each component individually.

Speed evaluation $\mathcal{S}(\sigma)$: We use the same black-box evaluation function from [4], trained on historical data to predict customer order coverage. The function simulates customer orders associated with a particular DC and their assignment to the FC providing fastest delivery and with inventory available.

Let $\Pi(k, \sigma)$ denote the promise (delivery days) for commodity $k \in K$ under solution σ , which depends on the FTD placements χ used by path p_k . For promises $\pi \in [0, 7]$ (from same-day delivery up to a full week), the order coverage of commodity k is:

$$\Phi(k, \sigma, \pi) = \begin{cases} 0 & \text{if } \Pi(k, \sigma) > \pi \\ \phi_{\text{black-box}}(k, \sigma, \pi) & \text{if } \Pi(k, \sigma) \leq \pi \end{cases}$$

where $\phi_{\text{black-box}}$ is the black-box coverage function. When $\pi = \Pi(k, \sigma)$, coverage depends on the order cutoff time determined by which FTD $\chi_{o_k j}^{kt}$ is used departing from origin o_k . This is because with early departure, fewer customer orders can be attained by the FTD and will depart only later or next day - hence, later FTDs cutoffs enable higher coverage. When $\pi > \Pi(k, \sigma)$, we assume all orders are covered, regardless of cutoff. Order coverage is converted to monetary value using promise-dependent weights w_π (lower π yields higher value). The total speed evaluation is:

$$\mathcal{S}(\sigma) = \sum_{k \in K} \sum_{\pi \in [0, 7]} w_\pi \cdot \Phi(k, \sigma, \pi)$$

Linehaul costs $\mathcal{L}(\sigma)$: For an FTD x_{ij}^t , we first define the volume it needs to transport as v_{ij}^t computed by:

$$v_{ij}^t = \sum_{k \in K} \chi_{ij}^{kt} \cdot v_k$$

For a truck capacity q and fixed cost τ per truck, the linehaul cost is then:

$$\mathcal{L}(\sigma) = \sum_{t \in T} \sum_{(i,j) \in A} \left\lceil \frac{v_{ij}^t}{q} \right\rceil \cdot \tau$$

The ceiling operator ensures we always count full truck departures, even when volume is low, reflecting the guaranteed departure property of FTDs.

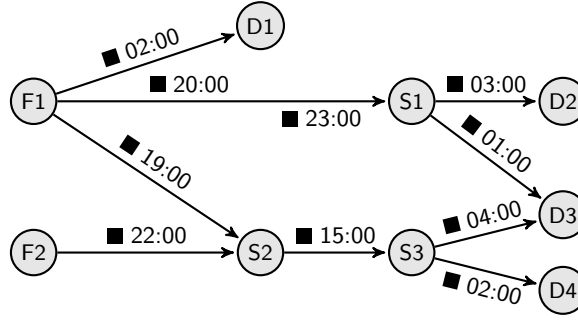
Sort costs $\mathcal{C}(\sigma)$: Sort centers incur per-package costs that depend on the operation type. Sorted volume costs α_{sort} per package, while cross-docked volume costs α_{xd} per package - typically $\alpha_{\text{xd}} < \alpha_{\text{sort}}$. We denote for a sort center s the total number of sorted packages processed as v_{sort}^s and total cross-docked packages as v_{xd}^s . The total sort cost then is:

$$\mathcal{C}(\sigma) = \sum_{s \in S} \alpha_{\text{sort}} \cdot v_{\text{sort}}^s + \alpha_{\text{xd}} \cdot v_{\text{xd}}^s$$

Let us now turn to a brief example that will help illustrate the key concepts of our problem.

Example

To illustrate the MMNDP, consider the example network in Figure 1. We assume six commodities (F1-D1, F1-D2, F1-D3, F1-D4, F2-D3, F2-D4), each with a volume of 1000 packages. Trucks have a capacity of 1000 packages and cost $\tau = \$500$ per trip. All paths through sort centers (SCs) require a 3-hour processing time at $\alpha_{\text{sort}} = \$1$ per package. Each arc has a 1-hour travel time. Commodities follow shortest-path routes: F1-D1 is a *direct* connection, F1-D2 is a *one-touch* path through S1, and F1-D4 is a *two-touch* path via S2 and S3.



■ **Figure 1** Example of an MMNDP network.

Timing and customer promises. The FTDs, shown as black squares with departure times, create time connections through the network. For example, F1-D1 has one FTD at 02:00. Between F1 and S1, there are two FTD waves at 20:00 and 23:00. Volume for F1-D2 flows via the 23:00 FTD in the first leg and 03:00 in the second leg. The 23:00 departure arrives at S1 by midnight, completes the 3-hour processing, and connects to the 03:00 departure to D2. This yields a 23:00 order cutoff for D2 customers with $\Pi(\text{F1-D2}, \sigma) = 1$ day promise delivery. In contrast, F1-D3 must use the 20:00 and 01:00 FTDs because the 23:00 departure arrives too late for the 01:00 downstream connection. This forces a much earlier 20:00 cutoff for D3, despite maintaining the same 1-day promise. The earlier cutoff likely reduces order volume significantly, as customers ordering after 20:00 see a 2-day promise instead. While both paths offer 1-day delivery, the monetary value of a 23:00 is likely to exceed that of a 20:00 cutoff due to captured demand. Meanwhile, F2-D3 has connections 22:00, 15:00 (+1 day) and 04:00 (+2 days), translating into $\Pi(\text{F2-D3}, \sigma) = 2$ days of promise with a cutoff of 22:00. This information is what constitutes the main input to $\mathcal{S}(\sigma)$.

Cost. The FTD in F1-D1 requires only one truck since all 1000 packages fit within capacity. However, the S2-S3 FTD requires three trucks since it carries volume for three commodities (F1-D4, F2-D3 and F2-D4), for a total of \$1500. In terms of sort costs, S2 processes 3000 packages across these flows, incurring \$3000 in sortation costs. Similar computations can be performed for all other components of the network to obtain the total linehaul and sortation costs.

3 Related work

The Middle Mile Network Design Problem belongs to the Service Network Design (SND) family of problems [5]. For a complete view over this rich literature, readers are referred to [6, 20]. The SND determines shipment paths for commodities and the required resources for transportation while consolidating flows to minimize network operating costs.

Early SND research considered basic capacity constraints [9, 20], but only recently has the literature incorporated more realistic operational constraints such as maximum inbound and outbound truck limits at sort centers [21, 14], which make the problem significantly more difficult to solve. Most SNDs ignore *labor efficiency* for sorting operations at intermediate facilities [4]. When labor capacity is taken into account, the volume arrival profile at a sort center directly impacts network speed and feasibility, as insufficient capacity can prevent appropriate processing for downstream connections. Moreover, the time required to move volume through a sort center may depend on how commodities were packed at upstream nodes, introducing interdependent capacity and routing decisions rarely addressed in prior work. To our knowledge, only Brubaker et al. [4] incorporates these constraints.

When modeling the time dimension in SNDs, most research opted for time discretization [21, 3]. While effective, this technique can quickly produce mathematical models that are too large to solve or even store in memory [3], something that can lead to several computational challenges. This in turn may force practitioners to use time granularities unsuitable for real applications. For example, we are unaware of exact models that can handle the 15-minute granularity and network scale we solve.

Regarding delivery speed, most SNDs specify the time when a commodity is available and its delivery deadlines, but lack any notion of network speed, as faster delivery provides no objective gain. The SND literature focuses mostly on cost minimization rather than speed maximization [6, 20]. Only two recent works consider speed optimization [2, 4], though Benidis et al. [2] examine networks with only FCs and DCs, eliminating the consolidation decisions central to typical SNDs. Even in this simplified setting, they show that the problem of scheduling FTDs to maximize network speed is NP-Hard.

In fact, most practical SNDs are NP-Hard, yet research has predominantly focused on exact methods [20], leaving a significant gap in scalable heuristic approaches for large-scale instances. This gap is significant for problems like the MMNDP, where high dimensionality and black-box evaluations for both objective function and constraints eliminate classical mathematical programming entirely. Brubaker et al. [4] showed that constraint programming required multiple hours on instances ten times smaller than ours, while a generic random key heuristic needed *multiple days* to obtain high-quality solutions. To our knowledge, no existing heuristic approach has been successfully demonstrated at the scale of our problem.

This motivates our adoption of Variable Neighborhood Search (VNS) [12], a metaheuristic successfully applied to diverse combinatorial problems including vehicle routing [17], graph coloring [1], and network design with relay operations [22]. Xiao and Konak [22] tackled an entirely different type of network design problem but still faced the challenge of having interdependent decision variables, much like us. In this context, they were able to achieve high-quality solutions by applying multiple search procedures within the VNS framework. Similarly, we employ Variable Neighborhood Descent (VND), a greedier VNS variant well-suited to the MMNDP's need for short runtimes while exploring high-dimensional decision spaces. Embedding several specialized neighborhoods within VND enables significant solution improvements, as we will discuss in the next section.

4 Algorithmic approach

The MMNDP's high dimensionality makes neighborhood-based local search particularly challenging. With over a hundred thousand commodities, hundreds of sites, and multiple decision types (FTDs, paths, sorting), the search space is enormous. Our heuristic defines multiple neighborhood structures, each containing a large number of candidate moves. Two challenges arise: first, fully enumerating and evaluating a neighborhood from scratch after every accepted move is prohibitively expensive - the black-box simulations incur several milliseconds per candidate, which can add up to an hour of computation time for a single exploration. Second, searching all neighborhoods jointly as one combined structure would yield an impractically large candidate set, making move selection too expensive within practical time budgets.

We address these challenges through two key design decisions. For the first, we compute neighborhoods once and cache them in priority queues, selectively updating them when the incumbent solution changes rather than regenerating from scratch. This amortizes the high cost of candidate generation across multiple iterations of the search. For the second, we

partition the search into smaller, carefully composed neighborhoods that define our VND structure. Each neighborhood groups moves that compete for similar resources and could block each other, whereas neighborhoods are ordered in such a way that earlier moves can enable better moves in later waves.

The metaheuristic approach we propose is described by Algorithm 1. We assume that an initial solution is already provided representing the current network in operation, which should be improved. The algorithm runs for up to $ITER$ iterations. In each of these, we loop over the neighborhood sets defined in $N = (N_1, N_2, N_3, \dots)$ and pick the best solution from all those which are reachable with the current neighborhood $N_n \in N$ (line 6). We denote by Q the set of priority queues holding the ordered list of candidate moves available - i.e., the neighborhood (see Section 4.2 for more details). Following the basic principle in VND, we explore neighborhood N_n until full exhaustion, always updating the best solution found with the new improving solution (lines 8-9). Once no improvement can be found, we move to the next neighborhood (lines 11-12).

■ **Algorithm 1** Variable Neighborhood Descent.

Input: Initial solution σ_0 , set of neighborhoods N , max iterations $ITER$
Output: Improved solution σ_{best}

```

1  $\sigma_{best} \leftarrow \sigma_0$ ;
2 for  $i \leftarrow 1$  to  $ITER$  do
3    $n \leftarrow 1$ ;
4    $Q \leftarrow \text{initNeighborhood}(\sigma_{best}, N_1)$ ;
5   while  $n \leq |N|$  do
6      $\sigma_* \leftarrow \text{bestImproving}(Q)$ ;
7     if  $\sigma_* \neq \emptyset$  then
8        $\sigma_{best} \leftarrow \sigma_*$ ;
9        $Q \leftarrow \text{updateNeighborhood}(\sigma_{best}, Q)$ ;
10    else
11       $n \leftarrow n + 1$ ;
12       $Q \leftarrow \text{initNeighborhood}(\sigma_{best}, N_n)$ ;
13    end
14  end
15 end
16 return  $\sigma_{best}$ 

```

Note that here we apply the *pipe* neighborhood change [13], in which we keep exploring the current neighborhood instead of resetting $n = 1$ as in standard VND. The rationale behind this decision is runtime, given how time consuming it can be to initialize a neighborhood exploration in our case - some neighborhoods can take up to one hour to initialize.

In what follows, we will describe the move types we designed for local search in Section 4.1. We define our neighborhoods formed by these moves in Section 4.2, while discuss some of our main contributions towards algorithmic techniques enabling Algorithm 1 to scale in Section 4.3. The reasoning behind multiple iterations is explained in the final Section 4.4.

4.1 Move types

Let us now turn our attention to the moves used by our local search to explore the large search space.

Direct FTD optimization

Given a direct path p_1 , this move tries to reoptimize the FTD by pushing it later in time and therefore gaining speed. When this is not possible because, e.g., there is too much volume already inbound late at the DC (recall constraint C4), another direct path p_2 may be considered to *pull its FTD earlier*. This leads to speed loss of orders in p_2 , but may enable the push of p_1 's FTD. The move then tries to tradeoff different coverages of FCs. If the FC for p_1 has a dominating coverage, we may even be able to gain most (if not all) of the speed lost from pulling p_2 . The overall idea of this move is depicted in Figure 2(a).

Indirect FTD optimization

Indirect FTD optimization differs from direct FTD due to the concept of *waves*. While in direct FTD a single placement is sufficient (because a single commodity is shipped), in the indirect network multiple commodities are shipped, potentially with largely different downstream connections and arrival dates. To maximize speed in this context, it may be worth to spend extra linehaul by adding a second (or more) FTD on the same leg, potentially reducing the promise of some commodities by an entire day. Indirect optimization can also have the reverse effect and *remove* extra waves which may not be needed given other optimizations done to the network. Figure 2(b) shows the idea behind this move type, where one FTD is converted into two waves.

Direct path addition and removal

This type of move selects one commodity $k \in K$ and converts its path from (or to) a direct path to (or from) a one-touch path. Figure 2(c) depicts the basic process behind the move. When converting to a direct path, the move also includes design of FTDs, since we always need at least one FTD per active leg to move volume between sites. When removing a direct path, FTDs are already available since we use existing active legs. The removal of a direct path may, however, re-design existing FC-SC FTDs to make the move feasible.

Adding a direct path contributes speed to the network at the expense of less consolidation. Meanwhile, removing the direct offers benefits of consolidation while losing speed - the typical tradeoff of the MMNDP. Note that we have specialized moves for direct and one-touch paths because they contribute the most speed to the network and carry the most volume.

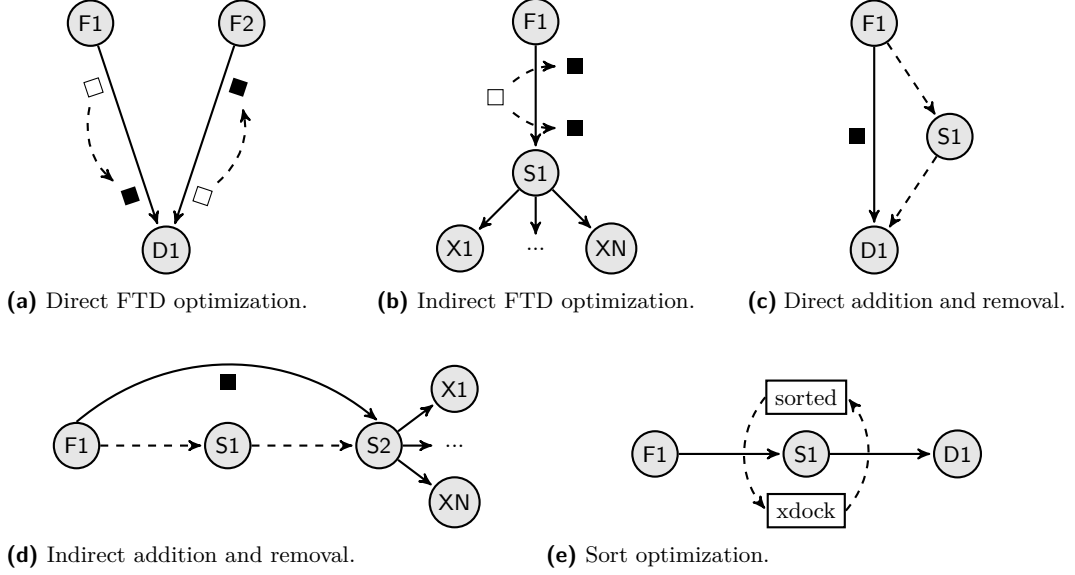
Indirect path addition and removal

Similar to the direct path addition and removal, this move is responsible for changing legs between FCs and SCs - we currently do not consider changes between SCs and other SCs or DCs. Adding a leg always involves designing new FTDs with the goal of increasing network speed, whereas removing an indirect leg will often reduce speed, but also reduce cost via consolidation. When an indirect leg is added or removed, a large number of commodities will be impacted, making this one of the most computationally expensive moves in our selection. Figure 2(d) depicts the general idea behind this type of move.

Sort optimization

As discussed in Section 2, one of the decisions in the MMNDP is also how individual commodities flow through SCs. They have essentially two modes: *sorted* and *cross-docked* (xdock). This move is responsible for looking at opportunities of optimizing this decision by either converting a sorted one-touch path into a xdock one-touch path, or by swapping two

paths (one sorted, one xdock). Note that FCs have a maximum capacity to send presorted (xdock) volume to SCs, which is why in some cases a swap may be needed. Sorted paths are slower at the SC and typically cost more, but given the limited capacity at the FCs, the decision of which path to xdock needs to be considered carefully. Figure 2(e) shows the idea behind this move.



■ **Figure 2** Move types available during local search.

4.2 Neighborhood definition

In our VND, each neighborhood is comprised of *multiple* move types. We have a total of five neighborhoods defined as follows:

- N_1 - direct path addition and direct FTD optimization
- N_2 - direct path removal and indirect FTD optimization
- N_3 - direct path addition and direct FTD optimization
- N_4 - indirect path addition and removal
- N_5 - sort optimization

The idea is that a neighborhood is then formed by the union of the neighborhoods defined by each move. The selection of move combinations to create a neighborhood is largely based on experimentation, experience and how synergies work across moves. For example, direct path removal can directly impact indirect FTD optimization since it adds more volume to an FC-SC leg. Hence, it makes sense to have both in the same neighborhood to compete for the best solution in a given iteration, rather than having them in separate neighborhoods when one could block the other move.

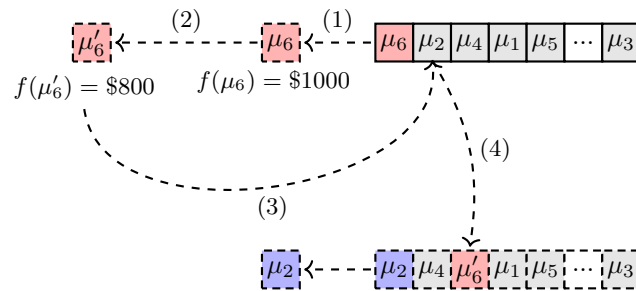
Note that neighborhoods N_1 and N_3 are the same. This is on purpose and comes from the fact that some opportunity may open after N_2 and we would like to capture these moves quickly to help ensure the speed of the network is maximized.

4.3 Neighborhood exploration, evaluation and caching

Our local search scales through several speedup techniques. Each neighborhood maintains a priority queue of candidate moves ordered by potential objective gain, initialized in lines 4 and 12 of Algorithm 1.

Neighborhood initialization. We enumerate all possible moves and add them to the priority queue. Full evaluation of the move feasibility and its objective value is computationally expensive. To reduce initialization cost, for three move types (direct path addition/removal and direct FTD optimization) we check only a subset of the feasibility constraints and order the priority queue by the upper bound estimate of the objective gain instead of the exact value of the objective. For example, direct path addition assigns the latest possible FTD, which injects at the critical time at the destination, regardless of labor capacity, providing an upper bound on potential gain without expensive computation. The remaining move types generate fully feasible candidates, as their complexity makes it more difficult to design simple upper bounds. These queues also serve as move caches to avoid redundant computations and candidate generation in our large search space.

Move selection. We employ *lazy evaluation* when selecting moves (line 6, Algorithm 1), illustrated in Figure 3. Here we start with a priority queue filled with candidate moves μ . Recall that these candidate moves may be only partially feasible (due to our initialization). The first step (1) is to take the top candidate μ_6 and optimize it. Optimizing a candidate means that we create a (new) fully feasible move over the current network state σ_{best} . Note that even fully feasible moves over a previous network state may need to be re-created at this stage. For example, if the move is to add a direct path, we will perform this path addition alongside a *proper* FTD design - and not the FTD design done during initialization. This process produces a new move μ'_6 (step 2). Optimization typically reduces the gain (here to \$800), as constraints are enforced or the network changes. We then compare μ'_6 against the next-best candidate μ_2 (step 3). If μ'_6 dominates, we select it; otherwise, we reinsert μ'_6 into the queue and repeat for μ_2 (step 4). This continues until finding a move with no superior alternative. Infeasible moves after optimization are always discarded.

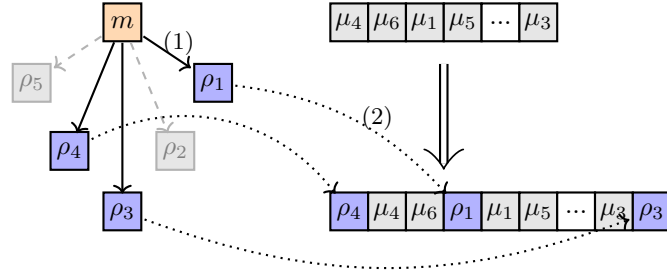


■ **Figure 3** Move selection process to extract a move from its queue during neighborhood exploration.

Neighborhood update. Full queue update is computationally prohibitive and therefore, we update the queue only partially. When a move m is accepted (line 8, Algorithm 1), the priority queues are partially updated to include newly available moves (line 9), as illustrated in Figure 4. In step (1), we enumerate new moves enabled by the acceptance of m , shown as blue moves ρ_1, ρ_3 , and ρ_4 . The remaining gray moves ρ_2 and ρ_5 represent opportunities

the algorithm does not detect, as full enumeration would be prohibitively expensive. For example, blue candidates may be those sharing the same leg as m , while gray ones are candidates on other legs which we have decided not to explore (see below for a more detailed example). This partial update increases search myopia but maintains scalability - we discuss a mitigation strategy in Section 4.4. Detected moves are added to the queue in step (2) ordered by objective gain. Note that these moves may include candidates with a structure similar to those already in the queue. This choice is by design because iterating over existing moves and updating the estimate one by one is time consuming; therefore we add the new move to the queue without removing an old move. Because the move selection process fully reevaluates a move, this prevents accepting infeasible or suboptimal moves, even if it may create duplicates of moves in the queue.

For example, when updating the queue for indirect FTD optimization, we look at the new paths created by the accepted move m (if any) and try to reoptimize all indirect legs in this new path. However, there may still be other indirect FTD moves that had their objective function reduced or increased because of the accepted move, but which will not be updated and reordered in the queue, leading the search to potentially miss some candidates.



■ **Figure 4** Process to update a move queue after a move has been accepted and the incumbent solution has changed.

Variable neighborhoods. Full queue updates or regeneration after each move acceptance are impractical. However, neighborhoods are fully initialized based on the current σ_{best} when starting exploration (lines 4 and 12 in Algorithm 1), allowing later neighborhoods to exploit information from earlier ones. For instance, N_2 generates all candidates from the σ_{best} modified by N_1 , not from the initial σ_0 . In contrast, placing all moves in a single neighborhood would only generate candidates once from σ_0 , substantially limiting move discovery. The other extreme, placing a single move per neighborhood causes the search to be too greedy and myopic due to the *pipe* search process [13] - yet, using the typical VNS process of always resetting to the first neighborhood upon solution improvement proved too time consuming.

4.4 Multiple iterations

Algorithm 1 can perform multiple iterations. The idea is to enable an easy trade-off between runtime and solution quality. While a single iteration is the most commonly setup we use in planning cycles, we have observed that up to three iterations can further provide significant benefits, particularly cost savings while maintaining the same network speed. This comes mainly from the fact that new iterations reset the candidates of each neighborhood, by-passing to some extent the limitations of our myopic queue update described in Section 4.3.

5 Experiments

To confirm the effectiveness of our algorithmic approach we perform a series of computational experiments using real world instances from previous network planning periods at Amazon. We first consider the specific characteristics of each instance, then compare results of VND against a previous algorithmic approach and finally consider the benefit brought by the multiple iterations. All of our experiments were performed in a computer equipped with Intel(R) Xeon(R) Gold 6455B running at 3GHz and equipped with 32 GB of RAM.

5.1 Instances

We consider six real-world instances in our experiments spanning different regions where Amazon operates. Table 1 shows their key characteristics, such as number of FCs, SCs, DCs and commodities.

■ **Table 1** Real-world instances used in the experiments and their key characteristics.

Instance	$ F $	$ S $	$ D $	$ K $
RW1	650	38	1297	121,098
RW2	602	27	1254	110,251
RW3	692	41	1080	180,982
RW4	570	36	1101	173,346
RW5	82	29	170	11,071
RW6	675	110	908	219,796

From this table, we note that the magnitude of the number of commodities $|K|$, one of the main drivers of instance complexity in SNDs, is approximately 10x larger than in the closely related work from Brubaker et al. [4], but is also almost 100x larger than instances in other network design studies that often include at most a few thousand commodities.

5.2 Comparison with a previous algorithmic approach

We compare the VND performance against the previous algorithmic approach used in planning and the same one used as baseline in [4]. This algorithm can be seen as our VND approach, but one where there is a single neighborhood N_1 with *all move types* in this single neighborhood N_1 . We refer to this as *baseline*. Note that both Baseline and VND are completely deterministic and therefore we report results for a single run per instance.

Table 2 presents the comparison of the two approaches. We report the relative difference between the final solution found by Baseline and the one found by VND. We do not provide absolute values to protect business financial values. Additionally, we report the total number of moves accepted by each approach, as well the absolute difference and the runtime, in hours, required by each algorithm as well as their relative difference.

It becomes clear from the results in Table 2 that VND dominates Baseline in essentially all instances except RW2. In this case, the entitlement drop is somewhat significant, reaching 7.5%. Despite that, we see that the runtime of VND is almost half that of Baseline for this same instance and we could likely achieve better solutions by fine tuning the neighborhood exploration further. However, VND provides an average improvement over these instances of 2.9% with an average runtime decrease of 24%. It is also important to note that the number of moves always increases, clearly demonstrating the additional ability of VND of exploring the search space further than the simple greedy Baseline approach.

■ **Table 2** Comparison of Baseline and VND.

Instance	Objective function	Moves			Time (h)		
	Δ (%)	Baseline	VND	Δ	Baseline	VND	Δ (%)
RW1	+1.7	338	381	+43	7.02	4.48	-36.1
RW2	-7.5	424	433	+9	8.36	4.68	-44.0
RW3	+4.4	278	314	+36	3.02	2.29	-24.1
RW4	+5.5	256	311	+55	1.99	1.84	-7.5
RW5	+5.3	387	407	+20	0.73	0.55	-24.1
RW6	+3.9	1414	1621	+207	1.70	1.49	-12.4

5.3 Improvement over multiple iterations and search progress

Let us now turn our attention to the tradeoff between runtime and solution quality when we increase the number of iterations of VND. Table 3 presents results for VND with one iteration and VND with three iterations. We present the objective function relative difference against both Baseline (Δ_B) and VND with one iteration (Δ_V). The number of moves and time are all comparing VND with one iteration against three iterations.

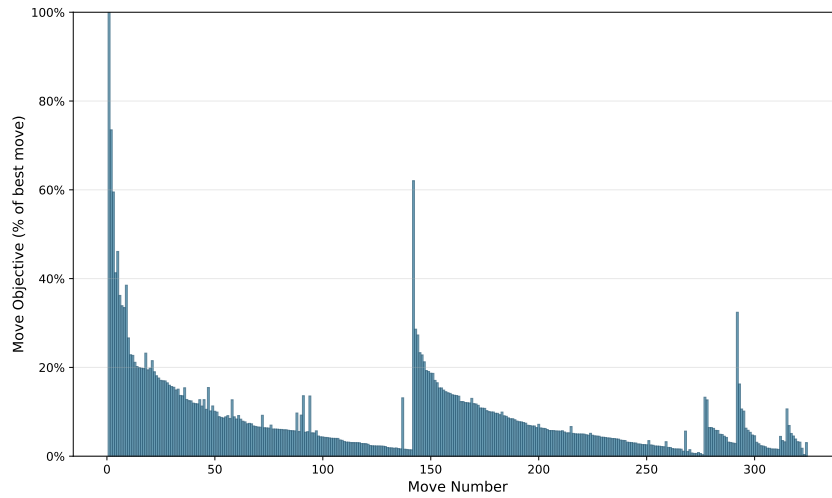
■ **Table 3** Comparison of VND runs with one iteration and three iterations.

Instance	Objective function		Moves			Time (h)		
	Δ_B (%)	Δ_V (%)	1-iter	3-iter	Δ	1-iter	3-iter	Δ (%)
RW1	+3.6	+1.9	381	455	+74	4.48	10.00	+122.9
RW2	-2.7	+5.3	433	484	+51	4.68	10.83	+131.4
RW3	+7.7	+3.2	314	346	+32	2.29	4.59	+100.3
RW4	+7.5	+1.9	311	324	+13	1.84	3.36	+82.6
RW5	+9.4	+3.9	407	440	+33	0.55	1.07	+93.9
RW6	+5.7	+1.7	1621	1734	+113	1.49	4.06	+172.4

It becomes clear that increasing the number of iterations does bring significant extra improvements compared to the single-iteration run - and even more so compared to Baseline. However, while the average gain on top of VND with one iteration is of almost 3% (and 5% for Baseline), the runtime explodes up to almost 11h in some cases. The number of moves performed also increases significantly, clearly demonstrating the capability of the algorithm in finding further new improving moves, albeit with ever increasing diminishing returns.

However, the table does not show the whole picture. Figure 5 depicts the evolution of moves performed for dataset RW4 for VND with three iterations. The x-axis shows the move acceptance index (0 is the first accepted move), while the y-axis shows the relative objective function gain of the move over the whole search (100% is the move with highest gain).

The local search progresses in the typical way: quick progression and improvement at the early stages and then quickly converging towards diminishing returns for moves. The key difference happens when we see the spikes after move index 150. This spike marks the change of neighborhood from N_1 to N_2 . Meanwhile, the transition from iteration one to iteration two happens at around move 270, at which point a new group of moves is found, with some occasional new spikes bringing additional gain, though not as high as at the beginning of the search. It becomes clear from this graph that iteration three brings little value. Indeed, a closer inspection shows that the third iteration produces only a single accepted move. Hence,



■ **Figure 5** Progress of accepted moves for RW4 and their relative objective function gain.

we could have stopped at the second iteration to save runtime without any significant loss for RW4. Nevertheless, the key message here is that re-initializing the search does bring benefits even if the precise number of iterations is instance-dependent.

6 Conclusion

In this work we have tackled a large scale middle mile network design problem, pushing the boundaries of what has been previously explored in the literature in terms of instance size, including cases with 100,000-200,000 commodities. The problem we solved seeks to maximize network speed at minimum cost, while respecting many realistic constraints with a high-dimensional decision space.

To tackle this Middle Mile Network Design Problem at scale, we design a specialized metaheuristic able to provide average improvements of 2.9% to solution quality, and a runtime up to 44% shorter when compared to a previous algorithm. We achieve these significant improvements by using a highly specialized neighborhood exploration to navigate the large search space combined with caching mechanisms, lazy evaluation and partial neighborhood updates. Furthermore, the algorithm explores different neighborhoods in a combined and sequential approach to ensure short runtimes at minimal solution quality loss. The techniques presented here are applicable beyond network design, as many real-world optimization problems share similar characteristics with the MMNDP, namely high-dimensional search spaces and expensive black-box evaluations.

Looking forward, we value the diversification of the search process. Initial experiments with perturbation mechanism to escape local minima demonstrated that we could gain further improvements of up to 5%. Unfortunately, this comes with prohibitively long computational times, which brings additional questions on how can we better explore the search space to shorten these runtimes. It may be worth exploring ways of decomposing the search, or identifying specific parts of the network which are worth (or not worth) exploring. Finally, we recognize that the order of neighborhood and move combinations in our algorithmic approach have a big impact in solution quality. Hence, it is also worth exploring approaches for automatic algorithm selection [16], where instance characteristics could be learned to automatically identify the best orders and moves to use to maximize solution quality while keeping runtimes controlled for practical use.

References

- 1 Cédric Avanthay, Alain Hertz, and Nicolas Zufferey. A variable neighborhood search for graph coloring. *European Journal of Operational Research*, 151(2):379–388, 2003. Meta-heuristics in combinatorial optimization. doi:10.1016/S0377-2217(02)00832-9.
- 2 Konstantinos Benidis, Georgios Paschos, Martin Gross, and George Iosifidis. Last truck scheduling for middle-mile next-day delivery coverage, 2025. arXiv:2310.18388.
- 3 Natasha Boland, Mike Hewitt, Luke Marshall, and Martin Savelsbergh. The continuous-time service network design problem. *Operations Research*, 65(5):1303–1321, 2017. doi:10.1287/opre.2017.1624.
- 4 Kyle Brubaker, Kyle E. C. Booth, Martin J. A. Schuetz, Philipp Loick, Jian Shen, Arun Ramamurthy, and Georgios Paschos. Optimization of next-day delivery coverage using constraint programming and random key optimizers, 2025. doi:10.48550/arXiv.2504.18749.
- 5 Teodor Gabriel Crainic. Service network design in freight transportation. *European Journal of Operational Research*, 122(2):272–288, 2000. doi:10.1016/S0377-2217(99)00233-7.
- 6 Teodor Gabriel Crainic and Walter Rei. 50 years of operations research for the tactical planning of consolidation-based freight transportation. *EURO Journal on Transportation and Logistics*, 14:100157, 2025. doi:10.1016/j.ejtl.2025.100157.
- 7 Ethan Cramer-Flood. Worldwide retail ecommerce forecast 2025. <https://www.emarketer.com/content/worldwide-retail-ecommerce-forecast-2025>, 2025. Accessed: 2026-04-19.
- 8 Vinayak Deshpande and Pradeep K. Pendem. Logistics performance, ratings, and its impact on customer purchasing behavior and sales in e-commerce platforms. *Manufacturing & Service Operations Management*, 25(3):827–845, 2023. doi:10.1287/msom.2021.1045.
- 9 Antonio Frangioni and Bernard Gendron. 0–1 reformulations of the multicommodity capacitated network design problem. *Discrete Applied Mathematics*, 157(6):1229–1241, 2009. Reformulation Techniques and Mathematical Programming. doi:10.1016/j.dam.2008.04.022.
- 10 Katrijn Gielens. Fulfillment, the new retailer battlefield. *Journal of Retailing*, 99(1):1–4, 2023. doi:10.1016/j.jretai.2023.03.001.
- 11 Sandy Gosling, Isabelle Pan, Loraine Li, and Sandeep Shenoy. What do US consumers want from e-commerce deliveries? <https://www.mckinsey.com/industries/logistics/our-insights/what-do-us-consumers-want-from-e-commerce-deliveries#/>, 2025. Accessed: 2026-04-19.
- 12 Pierre Hansen and Nenad Mladenović. Variable neighborhood search: Principles and applications. *European Journal of Operational Research*, 130(3):449–467, 2001. doi:10.1016/S0377-2217(00)00100-4.
- 13 Pierre Hansen, Nenad Mladenović, Raca Todosijević, and Saïd Hanafi. Variable neighborhood search: basics and variants. *EURO Journal on Computational Optimization*, 5(3):423–454, 2017. doi:10.1007/s13675-016-0075-x.
- 14 Edward He, Natasha Boland, George Nemhauser, and Martin Savelsbergh. An exact algorithm for the service network design problem with hub capacity constraints. *Networks*, 80(4):572–596, 2022. doi:10.1002/net.22128.
- 15 R. M. Karp. On the computational complexity of combinatorial problems. *Networks*, 5(1):45–68, 1975. doi:10.1002/net.1975.5.1.45.
- 16 Pascal Kerschke, Holger H. Hoos, Frank Neumann, and Heike Trautmann. Automated algorithm selection: Survey and perspectives. *Evolutionary Computation*, 27(1):3–45, 2019. doi:10.1162/evco_a_00242.
- 17 Jari Kytöjoki, Teemu Nuortio, Olli Bräysy, and Michel Gendreau. An efficient variable neighborhood search heuristic for very large scale vehicle routing problems. *Computers & Operations Research*, 34(9):2743–2757, 2007. doi:10.1016/j.cor.2005.10.010.
- 18 Sara Lebow. Delivery has sped up, but US retailers still eat amazon’s dust. <https://www.emarketer.com/content/delivery-sped-up-us-retailers-amazon>, 2023. Accessed: 2026-04-19.

- 19 David Niemeier. Global parcel market 2025: Growth at scale, power in few hands. <https://ecdb.com/blog/global-parcel-market-2025-growth-at-scale-power-in-few-hands/5168>, 2025. Accessed: 2026-04-19.
- 20 Michel Gendreau Teodor Gabriel Crainic and Bernard Gendron. *Network design with applications to transportation and logistics*. Springer Cham, 2021. doi:10.1007/978-3-030-64018-7.
- 21 Haotian Wu, Ian Herszterg, Martin Savelsbergh, and Yixiao Huang. Service network design for same-day delivery with hub capacity constraints. *Transportation Science*, 57(1):273–287, 2023. doi:10.1287/trsc.2022.1155.
- 22 Yiyong Xiao and Abdullah Konak. A variable neighborhood search for the network design problem with relays. *Journal of Heuristics*, 23(2):137–164, 2017. doi:10.1007/S10732-017-9332-2.

A

 Constraints

We provide a formal description of our constraints below. These are not intended as a mathematical programming formulation, but rather as a precise definition of what each constraint enforces. Constraints C4 and C5 depend on other network factors (truck scheduling simulation, path definitions) not described here. Constraints relating variables are also out of scope. In what follows, we assume that operations over the time buckets $t \in T$ work in a rolling horizon. In other words, if t^* denotes the last bucket in T , $t = t^* + 1 = 1 \in T$ denotes the first period.

Constraint C1 - Shift opening hours

Let $O_i^+ \subset T$ be the set of buckets when site $i \in V$ is open for inbound and $O_i^- \subset T$ the set of buckets when it is open for outbound. The shift constraint can then be written as:

$$\begin{aligned} x_{ij}^t &= 0, & \forall (i, j) \in A, t \in T, t' = t + a_{ij} \in T \setminus O_j^+ \\ x_{ij}^t &= 0, & \forall (i, j) \in A, t \in T \setminus O_i^- \end{aligned}$$

where a_{ij} denotes the travel time (as a multiple of 15 minutes) in arc $(i, j) \in A$.

Constraint C2 - Minimum time difference between two FTDs on the same arc

Let L_{ij} denote the minimum time difference, in 15-minutes buckets, required between two FTDs placed on arc $(i, j) \in A$. The constraint then is written as:

$$x_{ij}^t + x_{ij}^{t'} \leq 1, \forall (i, j) \in A, t \in T, t' \in [t + 1, t + L_{ij}], t \neq t'$$

Constraint C3 - Maximum FTDs departure and arrival per hour

Let H_i^+ denote the maximum FTD inbound per hour at site $i \in V$ and H_i^- the maximum FTD outbound per hour at i .

$$\begin{aligned} \sum_{(i,j) \in A} \sum_{t'=t}^{t+4} x_{ij}^{t'} &\leq H_i^-, & \forall i \in V, t \in T \\ \sum_{(j,i) \in A} \sum_{t'=t-a_{ij}}^{t-a_{ij}+4} x_{ij}^{t'} &\leq H_i^+, & \forall i \in V, t \in T \end{aligned}$$

where here we look into windows of 4 buckets (1 hour).

Constraint C4 - Labor capacity at SCs and DCs

Labor capacity at SCs and DCs is our most complex constraint, requiring two simulations: (1) simulating truck schedules based on volume assigned to an FTD under the current network state σ (since ad-hoc trucks are scheduled before FTDs to spread volume arrivals over time); and (2) simulating sortation at each site for the inbound truck volume, which depends on how much labor capacity is available.

Building the ad-hoc truck schedule is out of scope for this paper and is done so using a black-box simulation that maps an FTD to a complete truck schedule based on the amount of volume assigned to the FTD. Briefly, we use historical volume accumulation curves to estimate how many packages are available for ad-hoc trucks departing before the FTD truck.

Let u_i^t denote total inbound volume at time t at site $i \in S \cup D$ (summed over all truck arrivals), b_i^t the backlog of unprocessed packages at site i and time t , c_i^t the maximum processing capacity at site i and time t , and $[t_i^{start}, t_i^{end}]$ the sortation window within which all packages must be sorted. We assume that $u_i^t \geq 0$, $t \in [t_i^{start}, t_i^{end}]$ and $u_i^t = 0$ otherwise. Volume that arrives earlier is aggregated at t_i^{start} and volume should not be injected later than t_i^{end} . We then define our labor constraints as:

$$\begin{aligned} b_i^{t^{start}} &= u_i^{t^{start}}, & \forall i \in S \cup D \\ b_i^t &= \max\{0, b_i^{t-1} - c_i^{t-1} + u_i^t\}, & \forall i \in S \cup D, t \in [t_i^{start} + 1, t_i^{end}] \\ b_i^{t^{end}} &= 0, & \forall i \in S \cup D \end{aligned}$$

Constraint C5 - Sort capacity at FCs

Let P_i denote the maximum number of sorts that FC $i \in F$ can process. We count each destination connected via a sort or presort as one resource consumption for simplicity.

$$\sum_{j \in V} z_{ij} \leq P_i, \forall i \in F$$