Optimizing the Trade-Off Between Batching and Waiting: Subadditive Dispatching

Ignacio Erazo Alejandro Toriello
H. Milton Stewart School of Industrial and Systems Engineering
Georgia Institute of Technology, Atlanta, Georgia 30332
ierazo@gatech.edu, atoriello@gatech.edu

June 13, 2024

Abstract

Motivated by applications in e-commerce logistics where orders or items arrive at different times and must be dispatched or processed in batches, we propose the subadditive dispatching problem (SAD), a strongly NP-hard problem defined by a set of orders with release times and a non-decreasing subadditive dispatch time function. A single uncapacitated vehicle must dispatch orders in batches to minimize the makespan, the time at which all orders have been dispatched. We propose a mixed-integer linear formulation for SAD; based on the linear relaxation's optimal solution, we construct a two-dispatch solution with a 3/2 approximation ratio, and a solution with at most three dispatches that has a 4/3 approximation ratio under an additional assumption. The guarantees are respectively best possible for solutions using at most two or three dispatches. Furthermore, we analyze FIFO solutions, discuss cases when they are optimal, and provide a dynamic program to obtain them. Finally, we computationally test our methods on applications inspired by same-day delivery and routing on restricted topologies.

1 Introduction

From 2012 to 2021, the share of retail sales in the U.S. stemming from e-commerce grew from 8.0% to 19.1%, and year-over-year growth in online sales exceeded 12% every year (Digital Commerce 360, 2022). Business-to-business e-commerce sales are also expected to increase annually by 17.5% from 2020 to 2027 (Caldwell, 2021). The rapid growth in e-commerce underscores the importance of last-mile delivery, the last portion of the order fulfilment process, when items ordered are delivered to the end customer; research in this space is critical, as the last mile may incur up to 50% of a supply chain's total logistics costs (Vanelslander et al., 2013).

Last-mile delivery systems are increasingly complex; same-day delivery (SDD) systems are particularly difficult to design and operate, because the order arrival and packaging process overlaps significantly with the dispatching and delivery process, increasing the system's dynamism and reducing opportunities to consolidate orders and decrease routing costs (Klapp et al., 2020). Operational aspects and day-to-day decisions in SDD systems, such as how to solve related underlying vehicle routing problems (VRP) to design delivery routes, have received significant attention from the research community, e.g. Klapp et al. (2018b,a, 2020); Reyes et al. (2018); Shelbourne et al. (2017); Sun et al. (2021); Voccia et al. (2019); Wölck and Meisel (2022). Recently, some studies have focused on the design of SDD systems, e.g. Banerjee et al. (2022, 2023); Carlsson et al. (2021); Stroh et al. (2022), focusing on fleet sizing, region partitioning and other tactical-level questions. The inherent tension between order arrival times and the economies of scale obtained by batching more orders together for dispatching is the main motivation for our present work.

Set functions are defined using a ground set N, and assign a value to each subset $S \subseteq N$. Submodular functions are set functions often used to model the economies of scale that arise in the aforementioned applications, and have been extensively studied in combinatorial optimization (e.g. Krause and Golovin, 2014; Nemhauser and Wolsey, 1999; Schrijver, 2003). Formally, let $N := \{1, 2, \ldots, n\}$ and $f : 2^N \to \mathbb{R}$; f is submodular if

$$f(S \cup S') + f(S \cap S') \leq f(S) + f(S'), \qquad S, S' \subseteq N.$$

Another way to characterize submodular functions is by their *discrete concavity*, or diminishing

margins: the marginal change in value from adding an element to a subset decreases as the subset includes more elements,

$$f(S \cup \{j\}) - f(S) \geq f(S' \cup \{j\}) - f(S'), \qquad \forall S \subseteq S' \subseteq N, j \in N \setminus S'.$$

In many applications, submodular functions are also monotonically non-decreasing, $f(S) \leq f(S')$ for $S \subseteq S' \subseteq N$. By translating if necessary to obtain $f(\emptyset) = 0$, f becomes non-negative and trivially satisfies subadditivity,

$$f(S \cup S') \leq f(S) + f(S'), \qquad S, S' \subseteq N, \quad S \cap S' = \emptyset.$$

In fact, monotone submodular functions are a special case of *fractionally subadditive functions*, set functions f with the following property: for any non-empty $S \subseteq N$ and any $\alpha \in \mathbb{R}_+^{2^S}$ satisfying

$$\sum_{S' \subseteq S, S' \ni i} \alpha_{S'} \geq 1, \quad i \in S, \qquad \text{we have} \qquad \sum_{\emptyset \neq S' \subseteq S} \alpha_{S'} f(S') \geq f(S).$$

By using a binary α above, we obtain (non-fractional) subadditivity. The fact that submodular functions are fractionally subadditive follows from the Bondareva-Shapley theorem (Bondareva, 1963; Shapley, 1967) and the non-emptiness of base polyhedra of submodular functions (Edmonds, 1970; Shapley, 1971).

1.1 Problem Definition and Applications

To study the trade-off between batching economies of scale and the idle time spent waiting for orders, we propose the *subadditive dispatching* problem (SAD). The problem is characterized by a finite set N to dispatch or serve, where each element $i \in N$ has a release time, and a non-decreasing subadditive set function $f : 2^N \to \mathbb{R}_+$ with $f(\emptyset) = 0$. Depending on the context, e.g. delivery or production, N and f may represent different things; for clarity of exposition we adopt delivery terminology throughout the rest of the paper. Thus, N is a set of *orders*, a subset $S \subseteq N$ is a *batch* of orders, and f is the *dispatch time* function, $f(S)$ representing the time required for a vehicle to be dispatched from a depot to deliver the orders in batch S and return to the depot. A single delivery vehicle (or more generally, a server) is available to execute dispatches. Informally, the

goal is a partition of the order set N into batches that the vehicle can dispatch while minimizing the makespan, the end time of the last dispatch. We define SAD under the general case where f is subadditive, but derive most results under stronger assumptions that are often met in important applications:

(A1) f is subadditive.

(A2) f is fractionally subadditive.

(A3) f is submodular.

(B) $\min_{S \subseteq N}\{f(S) - \sum_{i \in S} \gamma_i\}$ can be solved in polynomial time for all $\gamma \in \mathbb{R}^N$ under the oracle model.

As we explain above, from (A1) to (A3) we progressively make a stronger assumption. Intuitively, condition (B) arises in column generation, where γ is a vector of dual multipliers. Furthermore, (A3) implies (B) because submodular minimization is solvable in polynomial time (Iwata et al., 2001; Schrijver, 2000), but there are also important cases in which (B) is satisfied but (A3) and (A2) are not.

Several special cases of SAD define important applications, including the following:

- $f(S) = g\left(\sum_{i \in S} \tau_i\right)$, where $g : \mathbb{R} \to \mathbb{R}$ is a non-decreasing concave function with $g(0) = 0$, and $\tau_i > 0$ for all $i \in N$; this function is submodular. In particular, when $\tau_i = 1$ for all $i \in N$, f is a function of the batch's cardinality. A specific application is $f(S) = a + b|S| + c\sqrt{|S|}$ for $S \neq \emptyset$, with $a, b, c \geq 0$; this function is a continuous approximation of expected routing and delivery time, used to model average-case SDD system behavior and to perform tactical-level design of SDD systems (Banerjee et al., 2022, 2023; Stroh et al., 2022). The SAD framework allows arbitrary, non-stationary (discretized) order arrivals, whereas previous analysis requires additional assumptions on arrival rates.

- Consider a set of nodes V with $N \subseteq V$, a depot node 0 and an undirected network $(V \cup 0,\ E)$ with non-negative edge lengths. For $S \subseteq N$, define $f(S)$ as the optimal length of a Steiner traveling salesman problem (TSP) through $S \cup 0$; a Steiner TSP tour must visit nodes $S \cup 0$ but may also visit other nodes in the graph. With this function, SAD captures operational SDD models in which same-day deliveries must be made to locations N in the network, where different orders are ready for delivery at different points in the operating day (Klapp

et al., 2018a). The function f is trivially subadditive in S, and is submodular for the class of *naturally submodular graphs* (Herer and Penn, 1995), which includes paths (Klapp et al., 2018b), trees and other similar topologies.

- Let $\mathcal{F} \subseteq 2^N$ be a collection of subsets of N called *families*, and define for each family $F \in \mathcal{F}$ a number $\tau_F > 0$. Then

$$f(S) = \sum_{F \in \mathcal{F}, F \cap S \neq \emptyset} \tau_F$$

is submodular and non-decreasing; with this function, SAD can model single-machine scheduling problems with release times, serial batching and family setups (Schutten et al., 1996; Goemans et al., 2002; Yuan et al., 2006), or the more general class of weighted coverage functions (Karimi et al., 2017).

- Consider a set of p products that can be produced via m production modes, where a matrix $(a_{k\ell}) = A \in \mathbb{R}_+^{p \times m}$ indicates production efficiency: $a_{k\ell}$ units of product k are produced when using mode ℓ for one unit of time. N represents a set of production orders, each given by a vector $b^i \in \mathbb{R}_+^p$, where order i requests b_k^i units of product k. The *linear production* function

$$f(S) = \min_{y \geq 0} \left\{ \sum_{\ell=1}^{m} y_\ell : Ay \geq \sum_{i \in S} b^i \right\}$$

describes the minimum time required to produce all products from orders in $S \subseteq N$ when fractional production is possible; this function is fractionally subadditive, and remains so for any non-negative objective vector (Owen, 1975). Although f is not necessarily submodular, condition (B) holds under additional assumptions; see Appendix D.4.

1.2 Contributions and Organization

We summarize our main contributions as follows:

1. We propose the subadditive dispatching problem (SAD) and show that it is strongly NP-hard even in the case of routing on generalized star graphs, a special case in which f is submodular. We present a mixed-integer linear programming (MILP) formulation for SAD and show that under condition (B) its linear programming (LP) relaxation can be solved in

polynomial time.

2. We show that, even for modular functions f, a heuristic using at most $d < n$ dispatches cannot have a multiplicative approximation guarantee better than $1 + 1/d$, and we provide heuristics matching this best-possible guarantee for $d = 1$ when (A1) holds, for $d = 2$ when (A2) holds, and for $d = 3$ when (A2) and an additional assumption hold. For $d = 2, 3$, our heuristics construct a SAD solution in polynomial time using an optimal solution of the LP relaxation. We also show that the LP relaxation can have a multiplicative gap of $\sqrt{3} - 1 \approx 0.73 < 3/4$ even when (A3) holds, implying that a solution based on the LP relaxation cannot have an approximation guarantee of $4/3$ for $d = 3$ without additional assumptions.

3. We analyze the performance of First-In First-Out (FIFO) solutions, in which orders are dispatched according to release times. The best FIFO solution can be computed efficiently via dynamic programming, and is a 2-approximation for SAD. We discuss FIFO-optimal functions, which define SAD instances with an optimal FIFO solution; we also generalize the concept to functions for which optimal solutions exhibit a partial FIFO structure, which we denote Q-FIFO-optimality.

4. We perform a computational analysis for two applications: tactical design of SDD systems under heterogeneous order arrival rates, and routing on tree topologies. The latter demonstrates the empirical effectiveness of our heuristics, whereas the former shows an application where optimal results can be obtained with our FIFO algorithm, and the practical insights that can be derived from the solution.

The rest of the paper has the following organization. Section 2 presents a brief literature review, while Section 3 defines SAD, formulates it as an MILP and discusses some preliminary complexity results. Section 4 presents our two approximation algorithms based on an optimal solution of the linear relaxation of the MILP formulation; furthermore, we also discuss a set of valid inequalities that can strengthen the bound given by the relaxation. Section 5 discusses FIFO solutions, their approximation guarantee, FIFO-optimality and its generalization. Section 6 summarizes our computational studies, and Section 7 concludes and outlines future avenues of work. The appendix

includes proofs omitted from the main body of the paper.

2 Literature Review

2.1 Same-Day Delivery

Different operational problems that arise in SDD systems have received recent attention; in particular, models that study dispatching, routing and delivery of orders in SDD are a particular case of vehicle routing problems (VRP) and have been studied under different conditions, including deterministic or stochastic order arrivals, and single- or multi-vehicle fleets, and also with varying objectives, such as minimizing makespan, minimizing total routing distance or maximizing the expected number of served orders. The common element in many models is the presence of release times that prevent a route from including an order if its departure occurs before the order is released. These models also usually have one common deadline for all deliveries (the end of the service day) rather than order-specific deadlines more common in food delivery (Reyes et al., 2018; Shelbourne et al., 2017). Because the literature on SDD models includes routing times in general road networks, e.g. Klapp et al. (2018a, 2020); Sun et al. (2021); Voccia et al. (2019); Wölck and Meisel (2022), they are generally not submodular nor fractionally subadditive except in special cases such as paths (Klapp et al., 2018b).

Recently, SDD tactical design models have used continuous approximations to study the average behavior of an SDD system (Banerjee et al., 2022, 2023; Stroh et al., 2022). Whereas the routing time for subsets of fixed locations is not necessarily submodular, under reasonably mild conditions the expected routing time when locations are sampled randomly from a geographic distribution exhibits economies of scale as the number of locations increases, growing in proportion to the square root of the number of locations; see Beardwood et al. (1959) and its many extensions. This translates to submodularity when discretizing arrivals, and is thus a suitable application for SAD. For a recent survey on applications of continuous approximations in logistics, see Franceschetti et al. (2017); other recent applications of these techniques in the last mile include Carlsson et al. (2021); Carlsson and Song (2017); Liu et al. (2020).

This article contributes to the SDD literature by generalizing deterministic models when routing times are submodular or fractionally subadditive, such as in Klapp et al. (2018b). Moreover,

SAD generalizes SDD tactical design models by allowing arbitrary arrival rates, which was not accommodated by previous works; this allows for a more accurate representation of the order arrival pattern.

2.2 Submodular and Subadditive Optimization

Submodular functions have been extensively studied in combinatorial optimization (e.g. Krause and Golovin, 2014; Nemhauser and Wolsey, 1999; Schrijver, 2003). Submodular minimization can be performed in polynomial time in the oracle model (Iwata et al., 2001; Schrijver, 2000), but many other submodular optimization problems are NP-hard, including submodular maximization and extensions of the minimization objective. In particular, SAD seeks a partition of a ground set to minimize the makespan, a scheduling objective. While we are not aware of similar models in the submodular optimization literature, some related models require the ground set to be partitioned while optimizing a different objective, usually the sum of the function values for sets in the partition, often with additional side constraints; see Bogunovic et al. (2017); Chekuri and Ene (2011); Hirayama et al. (2023); Wang et al. (2021); Wei et al. (2015) for recent examples of work in this area.

Subadditive and fractionally subadditive functions often appear in game theory and are used to model utility valuation for items that cannot be divided among multiple agents. In particular, the valuation class of fractionally subadditive functions, termed *XOS functions*, was defined in the context of combinatorial auctions (Nisan, 2000). Subadditive functions have been widely used to model multiple variants of welfare maximization problems (Barman and Sundaram, 2020; Barman et al., 2020; Chaudhury et al., 2020; Feige, 2009) and multi-item multi-bidder games (Cai and Zhao, 2017, 2019). The space between subadditive and fractionally subadditive functions remains an active area of study; see e.g. Bangachev and Weinberg (2023).

2.3 Machine Scheduling with Batching

Machine scheduling problems concern the assignment of jobs to one or several machines, with typical objectives such as minimizing makespan, lateness, weighted completion times, etc. Within the vast body of scheduling research, our current work is related most to scheduling models that consider release times, batching, or both, starting with Hariri and Potts (1983), among the first to

consider release times. SAD generalizes the canonical serial-batching and parallel-batching single-machine scheduling problems, and is also able to capture problems where there are batch specific setup times. Batch setup times first appeared in Monma and Potts (1989), and then Schutten et al. (1996) considered both release times and *family setup times*, a submodular generalization of batch setup times. Related work includes Ahmadi et al. (1992); Brucker et al. (1998); Dobson and Nambimadom (2001); Li et al. (2005); Yuan et al. (2006); the surveys Allahverdi (2015); Webster and Baker (1995) cover relevant literature, the former particularly focusing on setup times or costs, while the latter, older review covers batching. To the best of our knowledge, Yuan et al. (2006) is one of the few relatively recent works to study family setup times, release times, and the makespan objective for a single machine (or vehicle in our terminology); the authors show that this particular problem is strongly NP-hard, implying SAD is NP-hard as well. Other similar articles do not consider release times, but add deterioration or learning effects to the processing times, e.g. Bai et al. (2012); Lee and Wu (2010); Yang and Yang (2010).

SAD contributes to the machine scheduling literature by generalizing both sequential and parallel batching for single-machine scheduling with release times. We consider an uncapacitated machine, which differs from some of the parallel batching literature but allows us to study both problems through the same lens. Moreover, a recent parallel batching survey (Fowler and Mönch, 2022) identified a need to study problems with a combination of serial and parallel batching, such as *mixed-batch processing* models, in which processing times are a weighted sum of serial and parallel batch times (Wang et al., 2020); SAD captures such models. We also specifically improve on the 2-approximation algorithm proposed by Yuan et al. (2006) and give better approximation guarantees for a more general model.

3 Model Formulation and Preliminaries

The subadditive dispatching problem (SAD) is characterized by a finite set $N := \{1, 2, \ldots, n\}$ of orders with release times $r \in \mathbb{R}_+^N$ satisfying $0 = r_1 \leq r_2 \leq \cdots \leq r_n$, and by a single delivery vehicle that dispatches batches of orders according to a non-decreasing and subadditive dispatch time set function $f : 2^N \to \mathbb{R}_+$ with $f(\emptyset) = 0$.

The goal is to find a partition of the order set N into batches that the vehicle can dispatch while

minimizing the makespan, i.e. the end time of the last dispatch. The dispatches may not overlap in time, as they are performed by a single vehicle, and if a batch contains an order i, its dispatch cannot begin before i's release time r_i. A solution for SAD is thus comprised of a partition of N, and a feasible schedule for those dispatches.

The challenge stems from the trade-off between the economies of scale derived from batching and the waiting for orders to be released; if either element is removed, the problem is much easier. If all order release times are equal, subadditivity implies that it is optimal to dispatch the entire set N in a single batch. Conversely, the following result shows that if the batches are given, finding an optimal schedule is straightforward.

Proposition 1. *For any partition of N, we can compute the optimal ordering of the batches, the corresponding dispatch schedule, and the makespan in $O(n)$ time.*

Proof. We can map a batch $S \subseteq N$ to its largest-indexed order, $S \mapsto \max\{j \in S\}$, and no two batches in a partition are mapped to the same index; the mapping is one-to-one. Define the family of batches with largest-indexed element i, $\mathcal{N}_i := \{S \subseteq \{1, \dots, i\} : i \in S\}$; this results in a partition of the power set, $2^N \setminus \{\emptyset\} = \bigcup_{i \in N} \mathcal{N}_i$. Consider the following indexing procedure: for each batch S in the partition, retrieve its largest-indexed order, and label S with this index; S_i means $i = \max\{j \in S_i\}$. This takes $O(n)$ time, as we go over each order once. For each $i \in N$, check if there is a batch associated with i; if not, define $S_i = \emptyset$. This takes $O(n)$ time as well.

With this procedure, we construct in linear time an ordered list of subsets $S_1, \dots, S_n$ that partition N, where $S_i = \emptyset$ or $S_i \in \mathcal{N}_i$. Denote the start time of S_i's dispatch as t_i; if we dispatch the batches in the ordering given by the indexes, the feasibility conditions for a schedule are

$$t_i \geq r_i \qquad i \in N \tag{1a}$$

$$t_{i+1} \geq t_i + f(S_i) \qquad i \in N \setminus n. \tag{1b}$$

Condition (1a) requires a batch containing i to be dispatched no earlier than r_i. Condition (1b) requires the vehicle to complete dispatch i before starting dispatch $i+1$; the condition still applies when $S_i = \emptyset$, because $f(\emptyset) = 0$. Intuitively, in this case the condition can be traced back to the last non-empty batch, and ensures dispatch $i+1$ does not start before this dispatch ends.

Consider the following recursion, which computes feasible dispatch times and requires $O(n)$ operations:

$$t_1 = r_1 = 0; \qquad t_{i+1} = \max\{t_i + f(S_i), r_i\}, \quad i \in N \setminus n.$$

The makespan for this schedule is then $t_n + f(S_n)$. It is easy to see that if batches $S_1, \ldots, S_n$ are processed in this order, this makespan is the best possible.

We complete the proof by showing that no other sequencing of the batches can produce a better makespan. Suppose the batches are dispatched in a different sequence; starting from index 1, consider the first time we dispatch a non-empty batch S_i with a higher index before a non-empty batch $S_{i'}$ with a lower index, i.e. $i > i'$. If we swap the dispatch order of S_i and $S_{i'}$, we cannot increase the makespan because $r_{i'} \leq r_i$, and we may decrease it if $r_{i'} < r_i$. Therefore, if the batches are not dispatched in the ordering given by their indexes, we can perform pairwise swaps until we dispatch them in this order, and the swaps can only improve the makespan. □

Proposition 1 implies that SAD reduces to choosing the partition of N into batches. Using the batch families $\mathcal{N}_i = \{S \subseteq \{1, \ldots, i\} : i \in S\}$, the proof verifies that we can map any partition of N to a set of ordered tuples $(t_i, S_i)_{i \in N}$, where $S_i = \emptyset$ or $S_i \in \mathcal{N}_i$, and in the latter case t_i represents the start time of S_i's dispatch. Finally, the dispatch time vector $t = (t_1, \ldots, t_n)$ for this partition can be computed in linear time. Using this representation, we can model SAD as a mixed-integer linear program (MILP) with the following variables:

x_S: Indicates if batch $S \subseteq N$ is dispatched.

t_i: Start time for dispatch $i \in N$.

z: Makespan.

Proposition 2. *The following MILP solves SAD:*

$$\min_{t,x,z \geq 0} \; z \tag{2a}$$

$$\text{s.t. } t_i \geq r_i \qquad i \in N \tag{2b}$$

$$t_{i+1} \geq t_i + \sum_{S \in \mathcal{N}_i} x_S f(S) \qquad i \in N \setminus n \tag{2c}$$

$$z \geq t_n + \sum_{S \in \mathcal{N}_n} x_S f(S) \tag{2d}$$

$$\sum_{S \subseteq N, S \ni i} x_S = 1 \qquad i \in N \tag{2e}$$

$$x_S \in \{0,1\} \qquad \emptyset \neq S \subseteq N. \tag{2f}$$

The proof can be found in Appendix A.1. Next, we address the complexity of solving SAD.

Proposition 3. *SAD is strongly NP-Hard, even in the special case where f is a Steiner TSP in a generalized "star" graph, a tree where a depot node has arbitrary degree but all other nodes have degree one or two.*

This result is in addition to Yuan et al. (2006), who show that SAD is strongly NP-Hard when f represents a batch scheduling function with family setups. The proof can be found in Appendix A.2, and relies on a reduction from the 3-partition problem. Nevertheless, SAD has a straightforward 2-approximation algorithm.

Proposition 4. *A heuristic that groups all orders into a single batch, resulting in a solution with makespan $r_n + f(N)$, is a 2-approximation for SAD. This approximation guarantee is tight.*

Proof. Let z^* be the optimal makespan; then $r_n \leq t_n \leq z^*$. Similarly, $f(N) \leq z^*$, because $f(N)$ is the optimal makespan if all release times are zero. Thus, $r_n + f(N) \leq 2z^*$. It is simple to construct an instance with $n = 2$ in which the heuristic solution has a makespan that is twice the optimum: take a modular f with $f(\{1\}) = 1$, $f(\{2\}) = 0$ and $r_1 = 0$, $r_2 = 1$. The optimal solution has makespan 1, whereas the heuristic has makespan 2. □

This heuristic gives the simplest possible solution, as it uses only one dispatch. In general, solutions with a small number of dispatches are appealing, as they may be easier to compute and offer operational simplicity. The next proposition gives a lower bound on the approximation ratio of such solutions.

Proposition 5. *Consider a heuristic for SAD that, for any instance, generates a solution with at most $d < n$ dispatches. The heuristic's multiplicative approximation guarantee cannot be smaller than $1 + 1/d$, even when constraining f to be a modular set function.*

The family of instances proving this proposition is given in Appendix A.3. We next turn our attention to solving the linear relaxation of (2).

Lemma 6. *Consider the linear relaxation of* (2)*, where we relax the integrality of each x_S variable* (2f) *to non-negativity. This relaxation can be solved in polynomial time if condition (B) holds, that is, if* $\min_{S \subseteq N}\{f(S) - \sum_{i \in S} \gamma_i\}$ *can be solved in polynomial time for all $\gamma \in \mathbb{R}^n$.*

The proof can be found in Appendix A.4, and relies on linear programming duality and the equivalence of separation and optimization. Condition (B) essentially translates to the separation problem induced by variables x_S in the LP's dual.

4 Linear Relaxation-Based Analysis

4.1 Heuristics Based on an Optimal LP Solution

Lemma 6 states that the linear relaxation of (2) can be solved efficiently if condition (B) holds. We leverage this to construct heuristic solutions that take as input an extreme point optimal solution of the linear relaxation. These heuristics obtain guarantees that match the best possible limits given by Proposition 5 for two and three dispatches. We begin with Algorithm 1, which creates the fractional analogue of a schedule for a fractional solution by ordering batches with positive weight and computing their start and end times.

Proposition 7. *Let (t^{LP}, x^{LP}, z^{LP}) be a feasible extreme point of the LP relaxation of* (2)*. Using that solution as input, Algorithm 1 returns an ordered list (by increasing maximum index, then decreasing cardinality) of the* fractional dispatches*, batches $S \subseteq N$ with $x_S^{LP} > 0$, and returns their corresponding schedule (start, end and idle times) in $O(n \log n)$ time.*

The proof is in Appendix B.1, and is illustrated in Figures 1a and 1b. Based on this schedule of the solution, the next result slightly relaxes the instance to remove idle time; intuitively, removing the idle time makes the heuristic analysis easier, and the performance guarantees then translate to the original instance.

Proposition 8. *Let (t^{LP}, x^{LP}, z^{LP}) be an optimal extreme point solution of the LP relaxation of* (2) *for some instance I_0. Let η and Δ respectively be the vectors of batch schedule positions and idle times computed by*

Algorithm 1 Schedule for an extreme point of the LP relaxation of (2)

Notation: η_S: position of batch S in the schedule.
$\hat{t}_\eta$, $\hat{e}_\eta$: start and end time of dispatch for the batch in position η.
Δ_j: vehicle idle time before order j, for $j \in N$.

Input: Order set N, release time vector r, extreme point of the LP relaxation (t^{LP}, x^{LP}, z^{LP}).

Let $\mathcal{B}_j \leftarrow \emptyset$, $j \in N$ [$\mathcal{B}_j$ will be an ordered list of batches]
for all subsets $S \subseteq N$ with $x_S > 0$ **do**
    $j \leftarrow \max\{k \in S\}$
    Insert S in $\mathcal{B}_j$ in the earliest position where subsequent batches have smaller or equal cardinality
end for
$\ell \leftarrow 1$, $\hat{e}_0 \leftarrow 0$
for $j = 1, 2, \ldots, n$ **do**
    $\Delta_j \leftarrow 0$
    for $S \in \mathcal{B}_j$, according to $\mathcal{B}_j$'s ordering, **do**
        $\eta_S \leftarrow \ell$, $\hat{t}_\ell \leftarrow \max\{r_j, \hat{e}_{\ell-1}\}$
        if $\hat{t}_\ell - \hat{e}_{\ell-1} > 0$ **then**
            $\Delta_j \leftarrow \hat{t}_\ell - \hat{e}_{\ell-1}$
        end if
        $\hat{e}_\ell \leftarrow \hat{t}_\ell + x_S f(S)$, $\ell \leftarrow \ell + 1$
    end for
end for

Output: Vectors η, $\hat{t}$, $\hat{e}$ and Δ.

Algorithm 1. Suppose $\sum_{j \in N} \Delta_j > 0$; then Algorithm 2 creates an instance I_1 in $O(n)$ time for which x^{LP} induces an optimal solution and the schedule given by η has no idle time. Therefore, instance I_1's optimal fractional makespan is $z^{LP} - \sum_{j \in N} \Delta_j$.

This proof is in Appendix B.2, and the algorithm is illustrated in Figure 1c. The outputs of Algorithms 1 and 2 allow us to design two heuristics for SAD, leveraging the fact that the solution's schedule in the relaxed instance has no idle time.

The first heuristic uses two dispatches. Because the fractional solution's schedule has no idle time, some batch's fractional dispatch starts before $z^{LP}/2$, half of the fractional makespan, and ends at or after $z^{LP}/2$. Letting D be the union of orders included in any fractional dispatch up to this point, the heuristic first dispatches D as soon as its last order is released (which must be before $z^{LP}/2$), and then dispatches the remaining orders in a second batch $N \setminus D$ as soon as possible thereafter; see Figure 1d for an example. Algorithm 3 formalizes the procedure.

The second heuristic uses at most three dispatches, following intuition similar to the two-dispatch case, but with thirds of the fractional makespan rather than halves. Again relying on

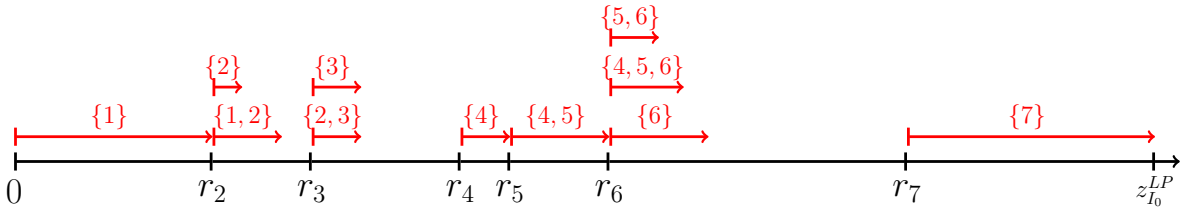
Algorithm 2 Transformation of an instance to remove idle time

Notation: η_S: position of batch S in the schedule.
$\hat{t}_\eta$, $\hat{e}_\eta$: start and end time of the dispatch for batch in position η.
Δ_j: vehicle idle time before order j, for $j \in N$.

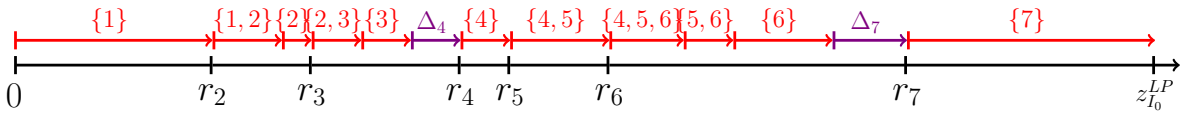
Input: Vectors $\Delta, \eta, \hat{t}, \hat{e}$, order set N and release time vector r.

$\hat{\Delta} \leftarrow 0$
for $j = 1, 2, \ldots, n$ **do**
  $\hat{\Delta} \leftarrow \hat{\Delta} + \Delta_j, \quad r_j \leftarrow r_j - \hat{\Delta}$
  for batches S with $j = \max\{k \in S\}$ **do**
    $\hat{t}_{\eta_S} \leftarrow \hat{t}_{\eta_S} - \hat{\Delta}, \quad \hat{e}_{\eta_S} \leftarrow \hat{e}_{\eta_S} - \hat{\Delta}$
  end for
end for

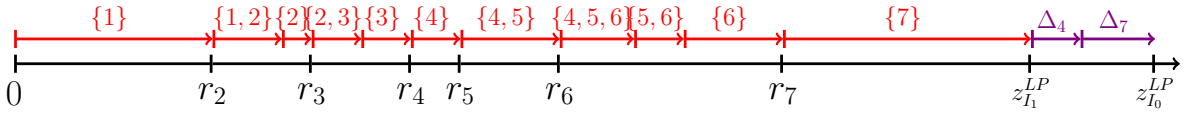
Output: Modified vectors r, $\hat{t}$, $\hat{e}$.



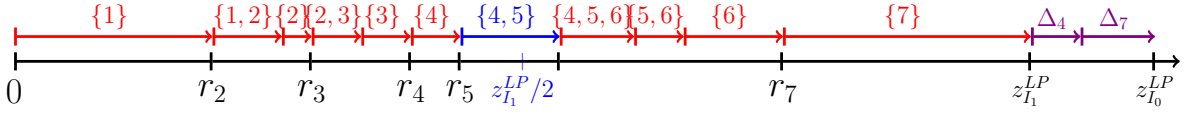
(a) Optimal fractional solution for some instance I_0.



(b) Schedule of the optimal fractional solution for I_0, including idle times.



(c) Modified instance I_1 obtained from Algorithm 2, with the same fractional dispatch schedule as I_0.



(d) Batch starting before $z^{LP}_{I_1}/2$ and ending at or after $z^{LP}_{I_1}/2$. This is the batch used in Algorithm 3.

Figure 1: Illustration of Algorithms 1, 2 and 3.

the fact that there is no idle time, some batch must be fractionally dispatched before $z^{LP}/3$ and return at or after $z^{LP}/3$; we let D_1 be the union of orders in all batches fractionally dispatched up to that point. Similarly, some batch is fractionally dispatched before $2z^{LP}/3$, and returns at or after $2z^{LP}/3$; we let D_2 be the set of orders fractionally served up to that point that are not present in D_1. Recalling that D is the set of orders used in the two-dispatch heuristic, the three-dispatch heuristic picks the best among five solutions:

(i) $\{N\}$ (one dispatch)

(ii) $\{D, N \setminus D\}$ (two dispatches)

(iii) $\{D_1, N \setminus D_1\}$ (two dispatches)

(iv) $\{D_1 \cup D_2, N \setminus (D_1 \cup D_2)\}$ (two dispatches)

(v) $\{D_1, D_2, N \setminus (D_1 \cup D_2)\}$ (three dispatches)

Notice that solutions (iii)-(v) differ only if $D_2 \neq \emptyset$. Algorithm 4 formalizes this description, and the next theorem gives the two heuristics' performance guarantees.

Algorithm 3 Two-dispatch heuristic

Notation: η_S: position of batch S in the schedule.
$\hat{t}_\eta$, $\hat{e}_\eta$: start and end time of the dispatch for batch in position η.
Δ_j: vehicle idle time before order j, for $j \in N$.

Input: Optimal extreme point (t^{LP}, x^{LP}, z^{LP}), order set N and release time vector r.

1: $\eta, \hat{t}, \hat{e}, \Delta \leftarrow$ Algorithm 1$((t^{LP}, x^{LP}, z^{LP}), N, r)$. ▷ [Values for original instance I_0]
2: $r, \hat{t}, \hat{e} \leftarrow$ Algorithm 2$(\Delta, \eta, \hat{t}, \hat{e}, N, r)$. ▷ [Values for modified instance I_1]
3: $z \leftarrow \max_\eta\{\hat{e}_\eta\}$ ▷ [Optimal fractional makespan of I_1, equal to $z^{LP} - \sum_j \Delta_j$]
4: $\hat{\imath} \leftarrow \min_\eta\{\hat{e}_\eta \geq z/2\}$ ▷ [Implies $\hat{t}_{\hat{\imath}} < z/2 \leq \hat{e}_{\hat{\imath}}$]
5: $D \leftarrow \bigcup\{S : \eta_S \leq \hat{\imath}\}$

Output: Solution with batches $D, N \setminus D$.

Theorem 9. *Let (t^{LP}, x^{LP}, z^{LP}) be an extreme point optimal solution of the LP relaxation of* (2)*, and define $\delta := z^{LP} - \sum_{S \subseteq N} x_S^{LP} f(S) = \sum_{j \in N} \Delta_j \geq 0$. If f is fractionally subadditive – condition (A2) – the two-dispatch solution obtained by Algorithm 3 has a tight $3/2 - \delta/(2z^{LP})$ approximation ratio. Furthermore, let $B = \{i \in N : r_i \leq \frac{z^{LP}-\delta}{3}\}$; if $f(B) \leq \frac{z^{LP}-\delta}{3}$, the solution obtained by Algorithm 4 has a tight $4/3 - \delta/(3z^{LP})$ approximation ratio. The algorithms run in $O(n \log n)$ time.*

Algorithm 4 Three-dispatch heuristic

Notation: η_S: position of batch S in the schedule.
$\hat{t}_\eta$, $\hat{e}_\eta$: start and end time of the dispatch for batch in position η.
Δ_j: vehicle idle time before order j, for $j \in N$.

Input: Optimal extreme point (t^{LP}, x^{LP}, z^{LP}), order set N and release time vector r

1: Perform Algorithm 3, retrieve D, $\hat{t}$, and $\hat{e}$.
2: $z \leftarrow \max_\eta\{\hat{e}_\eta\}$ ▷ [Optimal fractional makespan of I_1, equal to $z^{LP} - \sum_j \Delta_j$]
3: $\hat{\imath}_1 \leftarrow \min_\eta\{\hat{e}_\eta \geq z/3\}$, $\hat{\imath}_2 \leftarrow \min_\eta\{\hat{e}_\eta \geq 2z/3\}$
4: $D_1 \leftarrow \bigcup\{S : \eta_S \leq \hat{\imath}_1\}$, $D_2 \leftarrow \bigcup\{S : \eta_S \leq \hat{\imath}_2\} \setminus D_1$
5: Compute the makespan for the following five solutions: (i) $\{N\}$, (ii) $\{D, N\setminus D\}$, (iii) $\{D_1, N\setminus D_1\}$, (iv) $\{D_1 \cup D_2, N\setminus(D_1 \cup D_2)\}$, (v) $\{D_1, D_2, N\setminus(D_1 \cup D_2)\}$.

Output: Return the best solution from the last step

The proof is in Appendix B.3. When the LP relaxation has no idle time, $\delta = 0$, the approximation guarantees for Algorithms 3 and 4 (under the additional assumption) are $3/2$ and $4/3$, respectively; when $\delta > 0$, the slightly refined guarantees follow from the transformation of the original instance I_0 into a new instance I_1 that has no idle time. The tightness of both approximation ratios against the SAD optimal solution (rather than against the linear relaxation) follows from the family of instances used to prove Proposition 5; as $n \to \infty$ in the instance sequence, both heuristics' approximation ratios converge to their respective worst-case guarantees. Moreover, Proposition 5 also verifies that these guarantees are best possible for heuristics using at most two and three dispatches, respectively.

Intuitively, the additional condition required for Algorithm 4's guarantee means that most of the workload cannot be released at the beginning of the shift or workday. We may replace $z^{LP} - \delta$ respectively with an upper bound in the definition of B and with a lower bound in the inequality $f(B) \leq \frac{z^{LP}-\delta}{3}$ to obtain a condition we can check a priori.

Even if we expand Algorithm 4 to evaluate all three-dispatch solutions induced by Algorithm 1's fractional ordering, it is impossible to guarantee a ratio better than $3/2$ when the additional assumption does not hold. Without the assumption, it is possible to create adversarial instances that force the heuristic to use at most two dispatches, where the first dispatch ends after the last order arrives. This also applies to analogous solutions with at most $d > 3$ dispatches that are constructed from the linear relaxation using our techniques: an adversarial instance forces the solution to use just two dispatches, because the first dispatch is too long. Building on this insight and using a related family of instances, we prove next that no heuristic that relies only on the LP

relaxation of (2) can have an approximation ratio better than $(\sqrt{3}-1)^{-1} \approx 1.37 > 4/3$.

Proposition 10. *Let z_I^* be the optimal makespan of SAD for instance I, and z_I^{LP} be the optimal (fractional) makespan of the LP relaxation of* (2) *for instance I. There exists a family of instances $I_1, I_2, \ldots$ such that $\lim_{m\to\infty} z_{I_m}^*/z_{I_m}^{LP} = (\sqrt{3}-1)^{-1}$. Therefore, when $\zeta < (\sqrt{3}-1)^{-1}$ it is not possible to create a ζ-approximation algorithm for SAD based solely on the optimal solution of the LP relaxation of* (2).

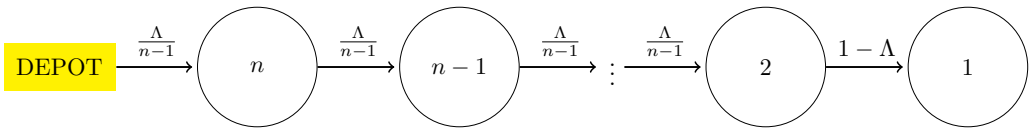


Figure 2: Family of instances where $z^{LP}(\sqrt{3}-1)^{-1}$ approaches z^*.

The family of instances used to prove Proposition 10 is shown in Figure 2, where $r_1 = 0$, $\tau_1 = 1$, $\Lambda = \sqrt{3-2/n}-1$, and $r_i = \Lambda\frac{i-1}{n-1}$, $\tau_i = \Lambda\frac{n-i+1}{n-1}$ for $i \geq 2$; the dispatch time function is $f(S) = \max_{i\in S}\{\tau_i\}$. The proof can be found in Appendix B.4. This family of instances has optimal solutions where batches are composed of intervals of consecutive indices. We discuss these solutions, how to compute them, and when they are optimal in Section 5.

As a final remark on the analysis of the LP relaxation and the heuristics, we note that the results in this section are independent of the complexity of solving the LP, i.e. they do not rely on condition (B). In particular, if an optimal extreme point solution of the LP relaxation is available, both algorithms can use it as input and provide solutions with the performance guarantees in polynomial time.

4.2 Valid Inequalities

We now discuss improving the lower bound given by the linear relaxation of (2) using valid inequalities.

Theorem 11. *Inequalities* (2b) *in* (2) *can be strengthened to*

$$t_i \geq r_i + \sum_{j<i}\sum_{S\in\mathcal{N}_j} \max\{0, r_j - r_i + f(S)\}x_S, \quad i = 2,3,\ldots,n. \tag{3}$$

The proof can be found in Appendix B.5. Consider the strengthened linear relaxation of (2)

where we replace (2b) with (3). Let $\beta \geq 0$ be the dual variables corresponding to (2c) and (2d), and γ to (2e). Let $\alpha \geq 0$ be the dual variables corresponding to the strengthened inequalities (3). The dual constraints corresponding to the (relaxed) x variables are

$$-\beta_i f(S) + \sum_{j \in S} \gamma_j - \sum_{j > i} \max\{0, r_i - r_j + f(S)\}\alpha_j \leq 0, \qquad i \in N, S \in \mathcal{N}_i.$$

For each $i \in N$, the separation problem for these constraints is then

$$\min_{S \in \mathcal{N}_i} \left\{ \beta_i f(S) - \sum_{j \in S} \gamma_j + \sum_{j > i} \max\{0, r_i - r_j + f(S)\}\alpha_j \right\}. \tag{4}$$

If (4) can be solved efficiently, the LP relaxation with strengthened inequalities can be solved efficiently. Unfortunately, even f being submodular does not guarantee that this can be solved efficiently, because of the coefficient multiplying each α_j. For generalized star graphs, we show in Appendix D.3 that (4) remains solvable in polynomial time. More generally, the separation problem can be solved with integer programming, and its efficiency will depend on the structure of the function f. We explore this computationally in Section 6.

5 First-In First-Out (FIFO) Algorithms and Functions

In the previous section, we study heuristics that produce a small number of dispatches. Next, we consider a different approach that does not limit the number of dispatches, but constrains their structure instead. A natural operating rule that arises in many applications is first-in, first-out (FIFO), the idea that orders should be dispatched in the sequence in which they are released. FIFO is appealing from an operational perspective, as it simplifies dispatching decisions, and may also offer customer service benefits. When combined with batching, FIFO implies that an order j cannot be dispatched before another order $i < j$; therefore, FIFO solutions only dispatch "interval" batches of the form $[i, j] := \{i, i+1, \ldots, j\}$ for $i \leq j$.

5.1 FIFO Algorithm

We next propose a dynamic program to compute the best FIFO solution for SAD. Introduce the notation $f_{i,j} := f([i,j])$, and compute $z_{i,j}$ for all $i, j \in N$ with $i \leq j$ using the following recursion:

$$z_{1,j} = r_j + f_{1,j} \qquad j \in N \tag{5a}$$

$$z_{i,j} = \max\left\{r_j, \min_{k<i}\{z_{k,i-1}\}\right\} + f_{i,j} \qquad 2 \leq i \leq j \leq n \tag{5b}$$

$$z^{\text{FIFO}} = \min_{i \in N}\{z_{i,n}\}. \tag{5c}$$

Intuitively, $z_{i,j}$ is the minimum makespan required to serve orders $1, \ldots, j$ in FIFO sequence when the last dispatched batch is $[i,j]$.

Proposition 12. *Let κ be the number of operations needed to compute all values $f_{i,j}$, for $1 \leq i \leq j \leq n$. For a SAD instance, the best makespan among FIFO solutions is given by z^{FIFO}, and can be computed in $\Theta(n^2 + \kappa)$ time.*

This proof is in Appendix C.1. As the result points out, for FIFO solutions the complexity bottleneck is actually κ, the complexity of calculating the $f_{i,j}$ values, which is $\Omega(n^2)$ (one computation per value) and may be larger for some functions.

Although it may perform well in many practical settings, in the worst case the FIFO algorithm cannot improve on the simple single-batch solution, which is itself FIFO; the next theorem formalizes this result.

Theorem 13. *The FIFO Algorithm is a 2-approximation for SAD, and this guarantee is tight.*

The theorem's proof is in Appendix C.2. Even though the theoretical guarantee of the FIFO Algorithm does not improve on the single-batch solution, it is indeed an optimal procedure for some functions, which we discuss in the next subsection.

5.2 FIFO-Optimal Functions and Generalizations

We say that f is FIFO-optimal if any instance of SAD with dispatch time function f has an optimal FIFO solution. Knowing that a function is FIFO-optimal allows us to optimize SAD in polyno-

mial time, and also guarantees the optimal solution will have the simple FIFO structure. Several important function classes turn out to be FIFO-optimal, as the next proposition explains.

Proposition 14. *Let $\tau_0 \geq 0$, $\tau_i > 0$ for $i \in N$, and let $g : \mathbb{R} \to \mathbb{R}$ be a concave non-decreasing function with $g(0) = 0$. The following functions are FIFO-optimal,*

1. $f(S) = \tau_0 + \sum_{i \in S} \tau_i$,
2. $f(S) = \tau_0 + \max_{i \in S}\{\tau_i\}$,
3. $f(S) = \tau_0 + g(|S|)$,

where in each case the function is defined for $S \neq \emptyset$, letting $f(\emptyset) = 0$.

The proof is in Appendix C.3. All three function classes in the proposition are non-decreasing and submodular; they include machine scheduling with batch setup times and either serial processing (1) or parallel processing (2), routing times along a single path (2), and continuous approximations of dispatch time (3), e.g. $f(S) = a + b|S| + c\sqrt{|S|}$.

Unfortunately, FIFO-optimality is not necessarily preserved by addition. In fact, even adding a modular function to a FIFO-optimal function may remove the FIFO-optimality property. To name one example, Figure 12 in the Appendix shows that the worst-case family of instances for the FIFO algorithm – in terms of approximation guarantee – has a dispatch time function f that represents routing times on a generalized "star" graph, a union of paths with a common depot. At the same time, $f(S) = \max_{i \in S}\{\tau_i\}$ represents routing time on a single path, and thus routing time on a generalized "star" graph is the sum of FIFO-optimal functions. Furthermore, Proposition 3 shows that SAD is strongly NP-Hard precisely for these dispatch time functions representing routing times on a generalized "star" graph. However, this proof also relies on a number of paths that grows with n. We next address the case in which the number of paths remains constant as n grows, and its generalization.

As a motivating example, consider an SAD instance in which dispatch times are given by routing on two disjoint paths with only a depot node in common. Even this function is not FIFO-optimal, but each path is individually "FIFO-optimal," i.e. should be dispatched in FIFO order, and an optimal solution can restrict itself to only dispatching batches containing orders from one path or the other. Intuitively, we can generalize the FIFO recursion (5) by tracking the last batch

dispatched on each path; this then leads to an optimal algorithm for the two-path instance with running time $\Theta(n^4 + \kappa)$, where κ now represents the time required to compute all values $f_{i,j}$ for interval batches from each path.

Formally, suppose the order set N is partitioned into Q subsets, $S_1, \ldots, S_Q$, and the dispatch function is given by

$$f(S) = \sum_{q=1}^{Q} f_q(S \cap S_q),$$

where each f_q is a FIFO-optimal function; we call such a function Q-FIFO-optimal.

Theorem 15. *Let Q be a fixed integer. For any Q-FIFO-optimal dispatch function, SAD can be optimized in $\Theta(n^{2Q} + \kappa)$ time, where κ is the time required to compute all interval dispatch times for each set S_q that partitions N.*

The proof can be found in Appendix C.4.

6 Computational Study

6.1 Routing on Restricted Topologies

In our first set of experiments, we study the computational performance of our proposed bounds and heuristics on routing problems in restricted topologies; we consider instances where the underlying graphs are generalized stars and trees with structure similar to generalized stars. To describe instances we use the notation (n, P, v), where n denotes to the number of orders, P is the number of paths, and v corresponds to the number of positions that can be visited on each path. We generate order release times using a Poisson process with an arrival rate of two per time unit, and order locations are sampled uniformly without replacement from the order positions in the respective network.

We consider trees with a central path containing the depot and P secondary paths stemming from the central one; we show an example in Figure 3, where $P = 12$ and $v = 4$. Black nodes represent potential order positions, light blue nodes represent the start of a path and the red node is the depot. Under this topology, paths are divided into two groups, with economies of scale when dispatching to orders within a group. Figure 4 shows the analogous generalized star graph

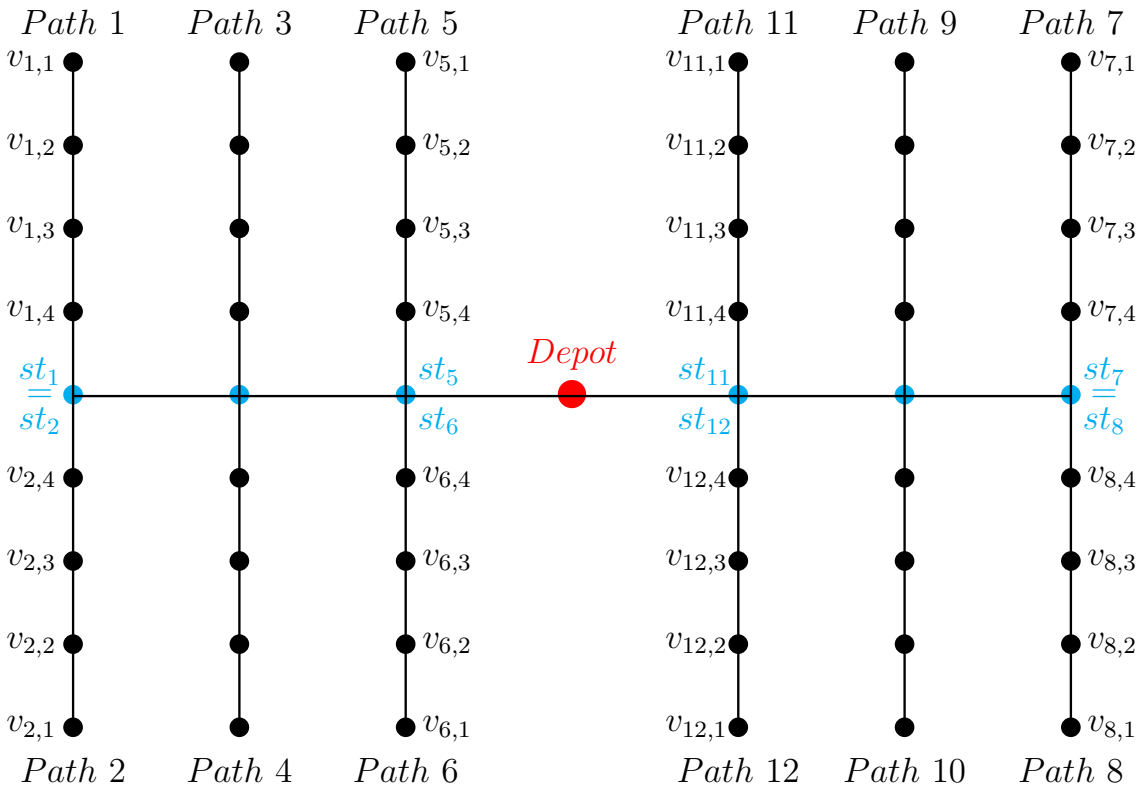


Figure 3: Tree topology used for our experiments when $(P, v) = (12, 4)$.

when $P = 12$ and $v = 4$. For both types of graphs, the dispatch time to a single position v_{ij} is the same; however, in generalized stars there are no economies of scale when dispatching to two or more different paths. For the same triples (n, P, v) defining a tree and generalized star instance, we use the same input vector of release times and the same order positions; the only difference is in f. Between any black node and its neighbors there is a round-trip distance of one time unit; there is a round-trip distance of two units between any two blue nodes, and between the depot and its neighbors.

We generate 25 instances for each parameter combination of (n, P, v), and use Python 3.11, Gurobi 10.0.1 and a Windows machine with 16 GB of RAM and an Intel Core i7-12650H processor for our experiments. We use the following notation to refer to the different bounds and solutions we evaluate:

- IP: MILP formulation (2).
- LPW: LP relaxation of (2), solved without column generation.

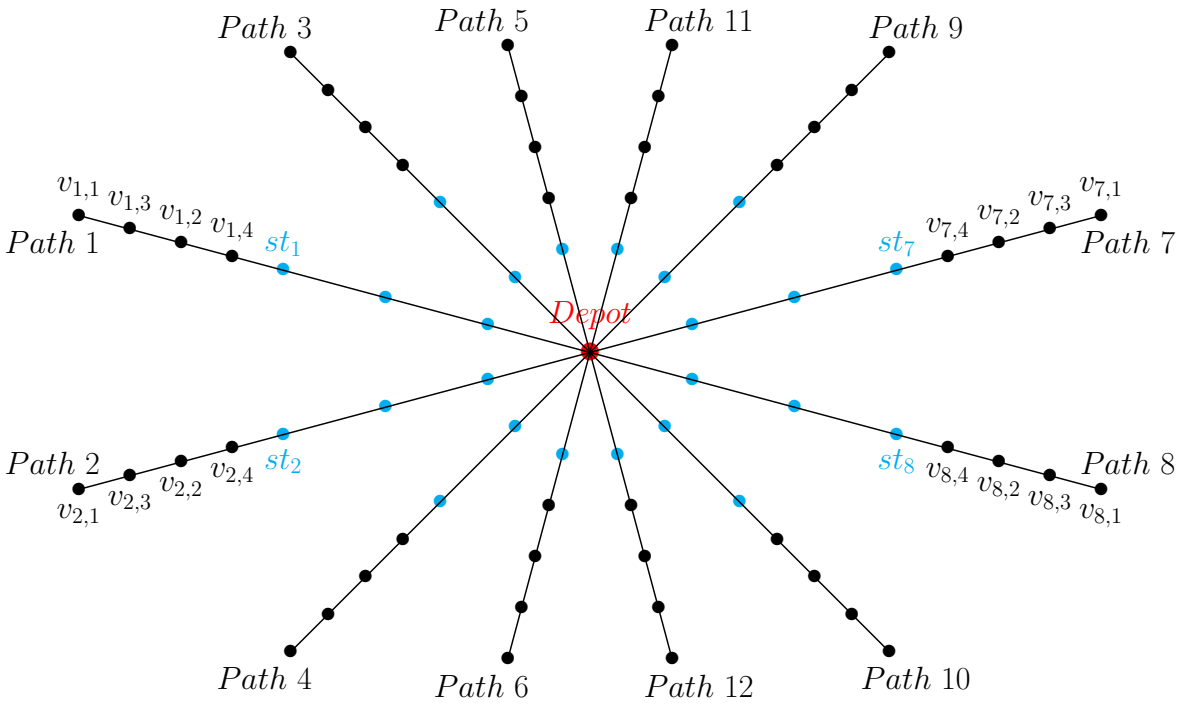


Figure 4: Generalized star topology used for our experiments when $(P, v) = (12, 4)$.

- CG: Column generation for the linear relaxation of (2).
- CGS: Column generation for the linear relaxation of (2) strengthened with (3).
- 3-DISP: Solution with at most three dispatches, based on Algorithm 4, but evaluating all two- and three-dispatch solutions consistent with the fractional ordering obtained from Algorithm 1. This enhancement of Algorithm 4 evaluates quadratically many solutions and can be carried out in polynomial time.
- CG IP: Heuristic solution of (2) restricted to columns generated by CG plus the columns corresponding to batches from Algorithm 4.
- FIFO: Best FIFO solution computed via the dynamic program (5).

For column generation methods, we use the acceleration technique from Ben-Ameur and Neto (2007), as explained in Appendix D.1. The pricing problems in CG and CGS for generalized star graphs and trees are detailed in Appendix D.2 and D.3, respectively. We do not present separate results for the two-dispatch heuristic, since these solutions are superseded by 3-DISP.

We also computed 4-DISP and 5-DISP, the four- and five-dispatch heuristics generalizing 3-DISP; they respectively require the evaluation of $O(n^3)$ and $O(n^4)$ candidate solutions. Our results indicate that 4-DISP requires significant additional computing time compared to 3-DISP, and only performs slightly better in a few extreme cases. Similarly, 5-DISP is dominated in performance and running time by CG IP. Therefore, we only report results for 3-DISP.

As detailed in Section 5, while generalized star instances are not FIFO-optimal, there are no economies of scale obtained by batching orders from different paths, and therefore we only need to consider interval batches within paths; this implies we can consider only a quadratic number of variables without loss of optimality. Therefore, the MILP formulation (2) and its relaxation can be directly optimized for relatively large values of n. Table 1 summarizes the relative performance of the tested methods for different generalized star instances, as compared to the optimal value given by (2). One set of instances considers a total of 400 orders and either 12, 20, 40 or 80 paths. The second set of instances considers 700 orders.

For all methods, the performance diminishes when the instance density n/P increases; however, the decrease is larger for CG compared to CGS, and for the 3-DISP and FIFO heuristics compared to CG IP. In particular, CG has optimality gaps between 99.81% and 97.05%, whereas CGS always has optimality gaps of at least 99.81%, which indicates that the valid inequalities significantly improve the optimality gap in high-density instances. On the heuristic side, CG IP obtains the optimal solution over 80% of the time in each of the instance classes. Furthermore, the worst optimality gap (over all instances) is within 0.3%, and the geometric means of this gap are always within 0.05%. On the other hand, 3-DISP and FIFO are significantly affected by the increase in density, reaching optimality gaps of up to 12.2%, and geometric means of around 10% and 11%, respectively.

With respect to running time, solving the MILP formulation becomes significantly more difficult when the density of the instances increases. For instances with low density, solving the MILP is faster than CG or the FIFO Heuristic; as density increases, the MILP solve time grows dramatically. For large values of n, FIFO is also slow compared to the LP relaxation methods.

In tree instances, there are economies of scale when batching orders in different paths, and therefore we must consider an exponential number of variables. This makes it impractical to solve the MILP (2) and even its LP relaxation directly even for very small instances. Also, in

		Baseline	Lower bounds		Heuristics		
(n, P, v)		IP	CG (LPW)	CGS	3-DISP	CG IP	FIFO
(400, 80, 100)	Gap geometric mean (%)	100	99.81	100.00	101.47	100.00	101.82
	Worst gap (%)	100	99.62	100.00	101.93	100.00	102.37
	Best gap (%)	100	100.00	100.00	100.68	100.00	101.30
	Same Obj. as IP (%)	-	4	100	0	100	0
	Beats ($\leq$) FIFO (%)	-	-	-	100	100	-
	Time per instance (s)	0.13	2.47 (0.07)	15.14	0.36	2.10	16.92
(400, 40, 100)	Gap geometric mean (%)	100	99.34	99.94	102.42	100.01	102.75
	Worst gap (%)	100	99.14	99.80	103.40	100.27	103.56
	Best gap (%)	100	99.84	100.00	100.89	100.00	101.79
	Same Obj. as IP (%)	-	0	36	0	88	0
	Beats ($\leq$) FIFO (%)	-	-	-	100	100	-
	Time per instance (s)	0.57	3.33 (0.12)	26.25	0.11	1.61	11.49
(400, 20, 100)	Gap geometric mean (%)	100	98.42	99.89	104.03	100.01	104.41
	Worst gap (%)	100	98.09	99.74	105.67	100.09	105.67
	Best gap (%)	100	98.83	100.00	100.94	100.00	103.46
	Same Obj. as IP (%)	-	0	8	0	92	0
	Beats ($\leq$) FIFO (%)	-	-	-	100	100	-
	Time per instance (s)	3.34	4.82 (0.17)	48.93	0.08	2.20	9.25
(400, 12, 100)	Gap geometric mean (%)	100	97.13	99.86	105.72	100.03	105.87
	Worst gap (%)	100	96.63	99.52	107.12	100.26	107.12
	Best gap (%)	100	97.82	100.00	103.80	100.00	103.80
	Same Obj. as IP (%)	-	0	4	0	80	0
	Beats ($\leq$) FIFO (%)	-	-	-	100	100	-
	Time per instance (s)	28.90	6.57 (0.27)	80.84	0.10	3.46	8.52
(700, 80, 100)	Gap geometric mean (%)	100	99.68	99.95	102.16	100.00	102.70
	Worst gap (%)	100	99.57	99.86	103.14	100.10	103.25
	Best gap (%)	100	99.80	100.00	100.76	100.00	102.28
	Same Obj. as IP (%)	-	0	20	0	96	0
	Beats ($\leq$) FIFO (%)	-	-	-	100	100	-
	Time per instance (s)	0.43	7.83 (0.22)	83.86	0.61	7.90	108.10
(700, 40, 100)	Gap geometric mean (%)	100	99.18	99.85	104.04	100.01	104.55
	Worst gap (%)	100	99.03	99.70	104.99	100.08	105.34
	Best gap (%)	100	99.37	99.95	102.06	100.00	103.81
	Same Obj. as IP (%)	-	0	0	0	84	0
	Beats ($\leq$) FIFO (%)	-	-	-	100	100	-
	Time per instance (s)	1.87	11.08 (0.31)	175.69	0.25	3.78	84.93
(700, 20, 100)	Gap geometric mean (%)	100	98.25	99.82	106.77	100.00	107.70
	Worst gap (%)	100	98.06	99.47	108.61	100.00	108.61
	Best gap (%)	100	98.66	99.95	104.34	100.00	106.89
	Same Obj. as IP (%)	-	0	0	0	100	0
	Beats ($\leq$) FIFO (%)	-	-	-	100	100	-
	Time per instance (s)	42.40	16.27 (0.61)	358.37	0.24	5.40	75.73
(700, 12, 100)	Gap geometric mean (%)	100	97.05	99.81	109.90	100.02	110.94
	Worst gap (%)	100	96.73	99.64	112.10	100.28	112.16
	Best gap (%)	100	97.34	99.98	106.03	100.00	109.80
	Same Obj. as IP (%)	-	0	0	0	88	0
	Beats ($\leq$) FIFO (%)	-	-	-	100	100	-
	Time per instance (s)	619.87	24.74 (1.12)	586.12	0.47	32.69	73.27

Table 1: Results for generalized star graph instances, compared to the optimal value.

		Baseline	Lower Bound	Heuristics		
(n, P, v)		CGS	CG	3-DISP	CG IP	FIFO
(150, 80, 100)	Gap geometric mean (%)	100	99.85	100.74	100.02	100.77
	Worst gap (%)	100	99.61	101.32	100.06	101.32
	Best gap (%)	100	99.96	100.01	100.00	100.14
	Same Obj. as Baseline (%)	-	0	0	44	0
	Beats ($\leq$) FIFO (%)	-	-	100	100	-
	Time per instance (s)	834.01	9.54	0.02	0.29	0.34
(150, 40, 100)	Gap geometric mean (%)	100	99.55	101.37	100.03	101.37
	Worst gap (%)	100	99.31	101.95	100.20	101.95
	Best gap (%)	100	99.73	100.72	100.00	100.72
	Same Obj. as Baseline (%)	-	0	0	64	0
	Beats ($\leq$) FIFO (%)	-	-	100	100	-
	Time per instance (s)	526.88	4.98	0.02	0.22	0.32
(150, 12, 100)	Gap geometric mean (%)	100	97.59	101.76	100.03	101.77
	Worst gap (%)	100	97.00	103.43	100.25	103.43
	Best gap (%)	100	98.32	100.78	100.00	100.78
	Same Obj. as Baseline (%)	-	0	0	76	0
	Beats ($\leq$) FIFO (%)	-	-	100	100	-
	Time per instance (s)	484.75	4.23	0.03	0.45	0.30

Table 2: Results for smaller tree instances, compared to the CGS lower bound as baseline.

tree instances we do not have an efficient pricing problem for the strengthened bound CGS, and must use an integer program. Therefore, we first use smaller instances to compare methods with CGS as baseline; Table 2 summarizes results for tree instances with 150 orders, and 80, 20 and 12 paths. With respect to performance, just as in the previous table, we see that the CG bound deteriorates somewhat when the density increases, and at a similar rate as for generalized stars. The three heuristics have slightly larger gaps, but not by more than around 1%; surprisingly, CG IP reaches optimality a higher proportion of the time when the density increases. Furthermore, 3-DISP is at least as good as FIFO in all instances. With respect to running time, we see a significant increase for CGS compared to generalized stars, given that the pricing problem requires solving integer programs. Both CGS and CG decrease their running time when density increases, and this indicates that CG and CG IP may be particularly useful in this case.

In larger tree instances, we use CG as baseline; Table 3 summarizes results for instances with $n = 400$ and $n = 700$. Just as in the previous tables, an increase in density decreases the heuristics' performance. Nonetheless, the previous experiments indicate that CG itself may be farther from optimality when density increases, and the 4% gap exhibited by CG IP against CG is in line with its gap in previous experiments. This suggests CG IP remains very close to optimal for tree instances. Furthermore, all the tested methods (excluding CGS) are scalable to these larger instances; in all

(n, P, v)		Baseline	Heuristics		
		CG	3-DISP	CG IP	FIFO
(400, 80, 100)	Gap geometric mean (%)	100	102.07	100.37	102.35
	Worst gap (%)	100	102.60	100.86	102.75
	Best gap (%)	100	101.17	100.08	101.85
	Beats ($\leq$) FIFO (%)	-	100	100	-
	Time per instance (s)	80.28	0.10	1.85	5.71
(400, 40, 100)	Gap geometric mean (%)	100	103.43	100.85	103.68
	Worst gap (%)	100	104.24	101.23	104.85
	Best gap (%)	100	101.23	100.29	102.89
	Beats ($\leq$) FIFO (%)	-	96	100	-
	Time per instance (s)	65.18	0.12	1.84	5.46
(400, 12, 100)	Gap geometric mean (%)	100	109.12	103.30	109.24
	Worst gap (%)	100	110.38	104.05	110.38
	Best gap (%)	100	107.62	102.30	107.62
	Beats ($\leq$) FIFO (%)	-	100	100	-
	Time per instance (s)	49.77	0.36	6.46	5.24
(700, 80, 100)	Gap geometric mean (%)	100	103.01	100.59	103.53
	Worst gap (%)	100	104.23	100.85	104.23
	Best gap (%)	100	101.35	100.24	102.98
	Beats ($\leq$) FIFO (%)	-	100	100	-
	Time per instance (s)	444.64	0.4	9.29	29.21
(700, 40, 100)	Gap geometric mean (%)	100	105.36	101.20	105.90
	Worst gap (%)	100	106.56	101.64	106.63
	Best gap (%)	100	103.12	100.85	104.86
	Beats ($\leq$) FIFO (%)	-	100	100	-
	Time per instance (s)	588.33	0.51	14.84	28.19
(700, 12, 100)	Gap geometric mean (%)	100	113.79	103.65	114.70
	Worst gap (%)	100	116.07	104.30	116.07
	Best gap (%)	100	109.40	103.16	113.15
	Beats ($\leq$) FIFO (%)	-	100	100	-
	Time per instance (s)	260.37	2.23	84.30	28.07

Table 3: Results for larger tree instances, compared to the CG lower bound as baseline.

classes, we can run CG and then CG IP on average in under 30 minutes.

In our final set of experiments in this section, we examine whether a smaller arrival rate may impact the heuristics' performance. Specifically, we modify the $(400, 80, 100)$ and $(400, 12, 100)$ tree instances by reducing the average arrival rate from 2 per time unit to $1, 1/2$ and $1/4$, keeping all other parameters constant, which translates to multiplying the original arrival times by $2, 4$ and 8. These modified instances can equivalently be interpreted as having sparser arrivals over time or a faster dispatch time function. Table 4 summarizes the results and includes the average number of dispatches generated by each heuristic. The results demonstrate the adaptability of CG IP, which maintains good performance and low gaps – under 4% on average in all cases. For 3-DISP and FIFO, gaps increase significantly as the arrival rate drops, but only until a point, after

			Baseline	Heuristics		
(n, P, v)	Arrival Rate		CG	3-DISP	CG IP	FIFO
(400, 80, 100)	2	Gap geometric mean (%)	100	102.07	100.37	102.35
		Average dispatch count	79.68	2.36	4.76	2.28
		Beats ($\leq$) FIFO (%)	-	100	100	-
	1	Gap geometric mean (%)	100	103.12	100.50	104.43
		Average dispatch count	87.40	2.48	6.40	2.44
		Beats ($\leq$) FIFO (%)	-	100	100	-
	1/2	Gap geometric mean (%)	100	104.00	100.86	108.35
		Average dispatch count	108.00	2.96	9.24	2.32
		Beats ($\leq$) FIFO (%)	-	100	100	-
	1/4	Gap geometric mean (%)	100	105.97	101.51	115.15
		Average dispatch count	138.00	3.00	13.96	2.88
		Beats ($\leq$) FIFO (%)	-	100	100	-
	1/8	Gap geometric mean (%)	100	111.49	102.25	125.91
		Average dispatch count	179.00	3.00	19.92	3.92
		Beats ($\leq$) FIFO (%)	-	100	100	-
(400, 12, 100)	2	Gap geometric mean (%)	100	109.12	103.30	109.24
		Average dispatch count	113.80	2.12	4.32	2.12
		Beats ($\leq$) FIFO (%)	-	100	100	-
	1	Gap geometric mean (%)	100	114.83	103.46	117.02
		Average dispatch count	127.56	2.40	6.36	2.56
		Beats ($\leq$) FIFO (%)	-	100	100	-
	1/2	Gap geometric mean (%)	100	119.83	103.55	128.71
		Average dispatch count	145.84	2.84	10.44	2.84
		Beats ($\leq$) FIFO (%)	-	100	100	-
	1/4	Gap geometric mean (%)	100	125.18	103.27	130.43
		Average dispatch count	193.12	3.00	18.92	3.88
		Beats ($\leq$) FIFO (%)	-	100	100	-
	1/8	Gap geometric mean (%)	100	110.34	100.72	107.25
		Average dispatch count	180.32	3.00	36.56	6.08
		Beats ($\leq$) FIFO (%)	-	4	100	-

Table 4: Impact of arrival rate on heuristic performance for $(400, 80, 100)$ and $(400, 12, 100)$ tree instances.

which the problem becomes easier. For 3-DISP, the gap increase likely stems from limiting the number of dispatches, whereas the increase for FIFO may be explained because it becomes more desirable to violate FIFO under a lower arrival density in order to create more efficient batches. We also see that 3-DISP outperforms FIFO by a significant amount until the sparsity of arrivals becomes too large.

6.2 SDD System with Non-Homogeneous Arrival Rates

In this experiment, we assess how time-varying order arrival rates affect planned dispatches in an SDD system when the decision maker seeks to minimize the makespan, which here corresponds

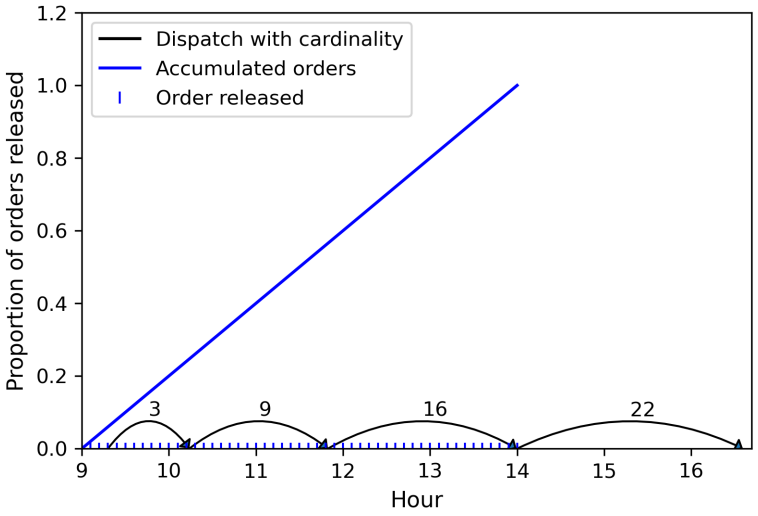


Figure 5: SDD tactical design results for uniform arrivals every six minutes.

to the delivery time guarantee that can be quoted to customers. From Proposition 14, the instance is FIFO-optimal, so we use the DP formulation (5).

Our case study uses an instance from Stroh et al. (2022), an SDD system with service area of around 26 square miles in northeastern metro Atlanta, including 22 census tracts and with a population of 92,198 as measured by the U.S. Census Bureau. The SDD system accepts orders between 9 AM and 2 PM; assuming 5% of the population in the region uses the SDD service once every two months, 50 people place orders each day on average. For this instance, Stroh et al. (2022) computed the continuous-time approximation of the dispatch-time function as $f(S) = 10 + 1.5|S| + 24\sqrt{|S|}$ minutes.

Fixing $r_1 = 9$ AM and $r_{50} = 2$ PM, we tested the following seven scenarios, illustrated in Figures 5 and 6a-6f: (i) constant order arrival rate, every six minutes; (ii) the first 30 orders arrive every four minutes, the next 20 orders arrive every nine minutes; (iii) the first 30 orders arrive every two minutes, the next 20 orders arrive every 12 minutes; (iv) the first 20 orders arrive every nine minutes, the last 30 orders arrive every four minutes; (v) the first 20 orders arrive every 12 minutes, the last 30 orders arrive every two minutes; (vi) the first 15 orders arrive every four minutes, the next 20 orders arrive every nine minutes, the last 15 orders arrive every four minutes; and (vii) the first 15 orders arrive every two minutes, the next 20 orders arrive every 12 minutes, the last 15 orders arrive every two minutes.

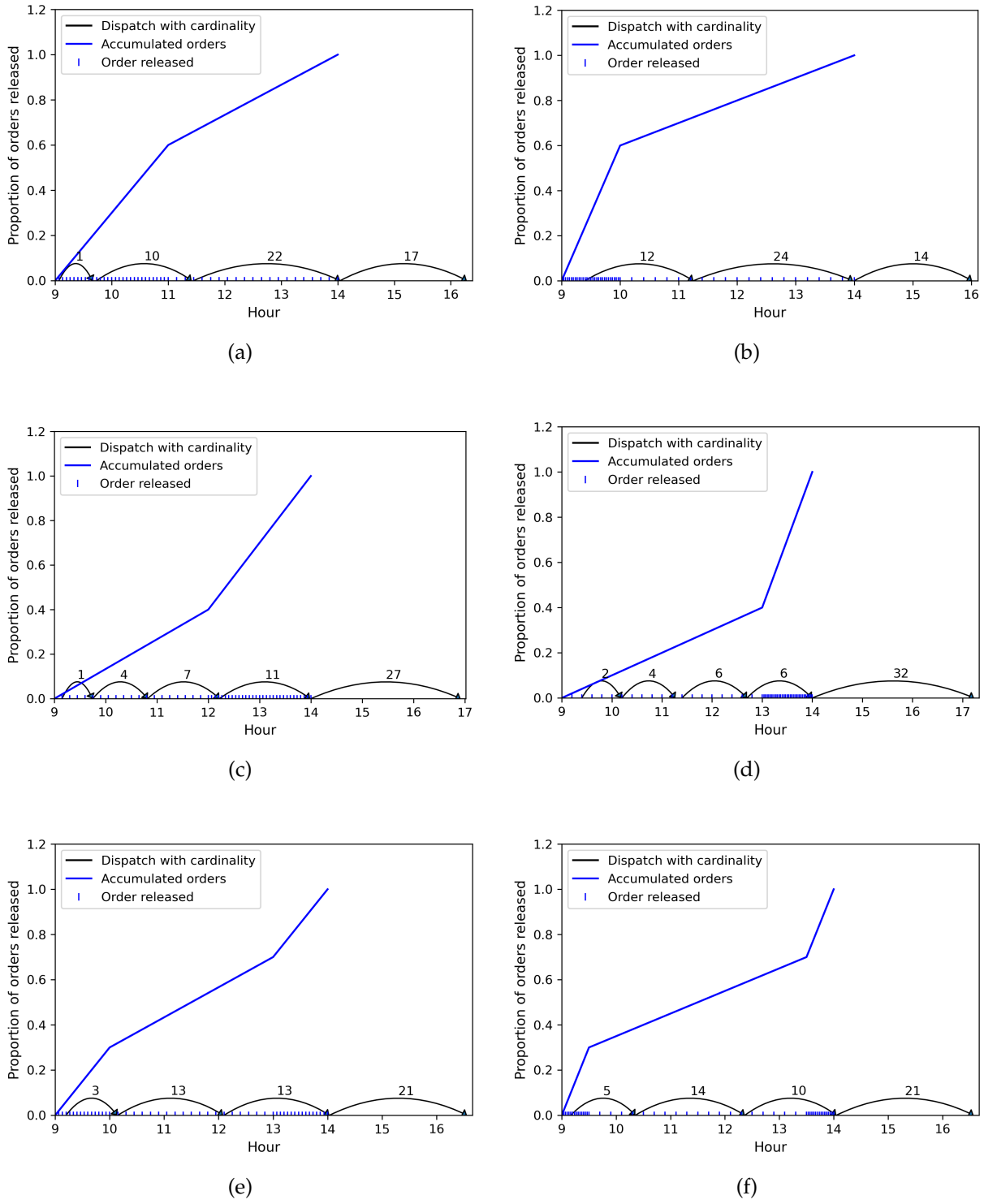


Figure 6: Dispatches for different order arrival patterns.

Assuming constant order arrivals from 9 AM until 2 PM (Figure 5), the solution has four planned dispatches with increasing quantities, and the last dispatch returns at roughly 4:30; this time corresponds to the delivery time guarantee that the SDD provider can quote to its customers. We next explore how varying arrival rates impact these conclusions. Figures 6a and 6b show that when more orders are expected at the start of the day, we can quote an earlier delivery guarantee and may see fewer dispatches; this suggests incentivizing SDD orders early in the day, perhaps via discounts or promotions. Conversely, if the system expects a surge in orders at the end of the window (Figures 6c and 6d), the delivery guarantee must be delayed and the vehicle may even make additional, inefficient dispatches; this suggests more conservative planning is merited if many SDD customers make last-minute orders at the end of the service window. Moreover, Figures 6e and 6f depict mixed scenarios with two order peaks at the start and end of the window. In this case, the two peaks somewhat cancel out and the final return time and number of dispatches are both similar to the case with constant arrivals. However, we do observe non-monotone order quantities in the dispatches when the peaks are pronounced (Figure 6f).

From the solution for scenario (i) when the arrival rate is constant, we can derive two different dispatch policies: the first dispatches batches with the same cardinalities, and is guaranteed to preserve total dispatch time; the second uses dispatch departure times, taking all orders that have arrived and have not been dispatched, leaving at those departure times if possible, or after the previous dispatch returns. We study the performance of the solutions generated by both policies under scenarios (ii)-(vii); the results are shown in Table 5.

Table 5 verifies that system performance gains significantly from directly optimizing the instance with non-homogeneous arrivals versus adapting a solution from a constant arrival scenario. The makespan of the optimal solution is at least 15 minutes shorter than either policy's solution in four of six scenarios, with a difference of at least 29 minutes in scenario (iii). We also observe that the departure time policy outperforms the cardinality policy in all scenarios except (ii), but it may introduce significant amounts of idle time, as in scenarios (iv) and (v).

Arrival pattern		Optimal solution	Cardinality policy	Departure time policy
Scenario (ii)	Makespan	4:17 PM	4:35 PM	4:40 PM
	Dispatch time	7h 6m	7h 17m	7h 22m
	Idle time	0m	6m	0m
Scenario (iii)	Makespan	4:00 PM	4:35 PM	4:29 PM
	Dispatch time	6h 36m	7h 17m	7h 11m
	Idle time	0m	12m	0m
Scenario (iv)	Makespan	4:55 PM	5:17 PM	5:13 PM
	Dispatch time	7h 45m	7h 17m	7h 0m
	Idle time	0m	33m	56m
Scenario (v)	Makespan	5:13 PM	6:01 PM	5:28 PM
	Dispatch time	7h 40m	7h 17m	6h 45m
	Idle time	0m	1h 8m	1h 25m
Scenario (vi)	Makespan	4:33 PM	4:42 PM	4:41 PM
	Dispatch time	7h 20m	7h 17m	7h 23m
	Idle time	0m	13m	0m
Scenario (vii)	Makespan	4:34 PM	4:51 PM	4:49 PM
	Dispatch time	7h 24m	7h 17m	7h 31m
	Idle time	0m	28m	0m

Table 5: Performance of the two solutions derived from scenario (i) when used for the non-homogeneous arrival rates from scenarios (ii)-(vii).

7 Conclusions

We introduced the subadditive dispatching problem (SAD), studied its complexity and devised different heuristic methods to obtain efficient solutions. SAD includes several important models as special cases, such as models for SDD system design, routing in restricted topologies and single-machine scheduling models with family setups and release times. We leveraged the LP relaxation of our MILP formulation to provide a polynomial-time 3/2-approximation algorithm with two dispatches for SAD with fractionally subadditive dispatch times, and a polynomial-time 4/3-approximation algorithm with at most three dispatches, under additional assumptions. Both approximation guarantees are best possible for heuristics with their respective number of dispatches; furthermore, we proved that without additional assumptions it is impossible to have an algorithm based on the LP relaxation with a guarantee below $(\sqrt{3}-1)^{-1} \approx 1.37 > 4/3$. We also discussed FIFO solutions, their approximation guarantee, and the special cases in which FIFO solutions are optimal. A computational study on routing in restricted topologies shows that the linear relaxation bound and our various heuristic solutions have reasonable gaps, and that they can be obtained in reasonable computational time even for large instances with up to 700 orders. We used our FIFO algorithm to computationally study SDD tactical design problems with non-

stationary order arrival rates, which could not be accommodated by previous work.

Our results motivate several avenues for future research, as SAD can be naturally generalized in several ways. We are currently studying the multi-vehicle version of SAD, which has many of the same applications as single-vehicle SAD. Even for modular dispatch times, the two-vehicle version of SAD is already NP-hard without release dates, as it generalizes the partition problem, and therefore presents significant additional challenges. Another interesting variant is a capacitated SAD, where batches are limited by cardinality, or more generally by a knapsack constraint. This variant is also already NP-hard even in very simple cases (Poon and Zhang, 2004). Other generalizations could include order deadlines, different objectives, or dispatch time functions that are not fractionally subadditive but have other appealing structural features. In general, the study of integrated logistics processes in the presence of varying order arrivals and economies of scale in dispatching or processing have significant application potential in e-commerce, and pose many interesting challenges for the research community.

Acknowledgments

The authors thank the area editor, associate editor and referees for their prompt feedback and constructive comments, which motivated significant extensions to the results and helped improve the exposition of the paper. The authors' work was partially supported by the U.S. Office of Naval Research, grant N00014-18-1-2075.

References

J. H. Ahmadi, R. H. Ahmadi, S. Dasu, and C. S. Tang. Batching and Scheduling Jobs on Batch and Discrete Processors. *Operations Research*, 40:750–763, 1992.

A. Allahverdi. The third comprehensive survey on scheduling problems with setup times/costs. *European Journal of Operational Research*, 246(2):345–378, 2015. ISSN 0377-2217. doi: https://doi.org/10.1016/j.ejor.2015.04.004. URL https://www.sciencedirect.com/science/article/pii/S0377221715002763.

J. Bai, Z.-R. Li, and X. Huang. Single-machine group scheduling with general deterioration and learning effects. *Applied Mathematical Modelling*, 36(3):1267–1274, 2012. ISSN 0307-904X. doi: https://doi.org/10.1016/j.apm.2011.07.068. URL https://www.sciencedirect.com/science/article/pii/S0307904X11004665.

D. Banerjee, A. Erera, and A. Toriello. Fleet Sizing and Service Region Partitioning for Same-Day Delivery Systems. *Transportation Science*, 56:1327–1347, 2022.

D. Banerjee, A. Erera, A. Stroh, and A. Toriello. Who has access to e-commerce and when? Time-varying service regions in same-day delivery. *Transportation Research Part B: Methodological*, 170:148–168, 2023.

K. Bangachev and S. M. Weinberg. q-partitioning valuations: Exploring the space between subadditive and fractionally subadditive valuations. arxiv.org/abs/2304.01451, 2023.

S. Barman and R. G. Sundaram. Uniform welfare guarantees under identical subadditive valuations. In C. Bessiere, editor, *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, pages 46–52. International Joint Conferences on Artificial Intelligence Organization, 7 2020. doi: 10.24963/ijcai.2020/7. URL https://doi.org/10.24963/ijcai.2020/7. Main track.

S. Barman, U. Bhaskar, A. Krishna, and R. G. Sundaram. Tight approximation algorithms for p-mean welfare under subadditive valuations. arxiv.org/abs/2005.07370, 2020.

J. Beardwood, J. Halton, and J. Hammersley. The shortest path through many points. *Mathematical Proceedings of the Cambridge Philosophical Society*, 55(4):299–327, 1959.

W. Ben-Ameur and J. Neto. Acceleration of cutting-plane and column generation algorithms: Applications to network design. *Networks*, 49:3–17, 01 2007. doi: 10.1002/net.20137.

I. Bogunovic, S. Mitrović, J. Scarlett, and V. Cevher. Robust submodular maximization: A non-uniform partitioning approach. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, page 508–516. JMLR.org, 2017.

O. Bondareva. Some applications of linear programming methods to the theory of cooperative games (in Russian). *Problemy Kybernetiki*, 10:119–139, 1963.

P. Brucker, A. Gladky, H. Hoogeveen, M. Y. Kovalyov, C. N. Potts, T. Tautenhahn, and S. L. van de Velde. Scheduling a batching machine. *Journal of Scheduling*, 1(1):31–54, 1998.

Y. Cai and M. Zhao. Simple mechanisms for subadditive buyers via duality. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2017, page 170–183, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450345286. doi: 10.1145/3055399.3055465. URL https://doi.org/10.1145/3055399.3055465.

Y. Cai and M. Zhao. Simple mechanisms for subadditive buyers via duality. *SIGecom Exch.*, 17(1):39–53, may 2019. doi: 10.1145/3331033.3331037. URL https://doi.org/10.1145/3331033.3331037.

A. Caldwell. 67 e-commerce stats and facts to know in 2021, 2021. URL https://www.netsuite.com/portal/resource/articles/ecommerce/ecommerce-statistics.shtml.

J. G. Carlsson and S. Song. Coordinated logistics with a truck and a drone. *Management Science*, 64(9):1–31, 2017.

J. G. Carlsson, S. Liu, N. Salari, and H. Yu. Provably good region partitioning for on-time last-mile delivery. In review, URL https://ssrn.com/abstract=3915544., 2021.

B. R. Chaudhury, J. Garg, and R. Mehta. Fair and efficient allocations under subadditive valuations. In *AAAI Conference on Artificial Intelligence*, 2020. URL https://api.semanticscholar.org/CorpusID:218628851.

C. Chekuri and A. Ene. Approximation algorithms for submodular multiway partition. In *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*, pages 807–816, 2011. doi: 10.1109/FOCS.2011.34.

Digital Commerce 360. E-commerce growth, 2022. URL https://www.digitalcommerce360.com/article/us-ecommerce-sales/.

G. Dobson and R. S. Nambimadom. The Batch Loading and Scheduling Problem. *Operations Research*, 49:52–65, 2001.

J. Edmonds. Submodular functions, matroids, and certain polyhedra. In *Combinatorial Structures and their Applications, Proceedings Calgary International Conference*, pages 69–87, 1970.

U. Feige. On maximizing welfare when utility functions are subadditive. *SIAM Journal on Computing*, 39(1): 122–142, 2009. doi: 10.1137/070680977. URL https://doi.org/10.1137/070680977.

J. W. Fowler and L. Mönch. A survey of scheduling with parallel batch (p-batch) processing. *European Journal of Operational Research*, 298(1):1–24, 2022. ISSN 0377-2217. doi: https://doi.org/10.1016/j.ejor.2021.06.012. URL https://www.sciencedirect.com/science/article/pii/S037722172100518X.

A. Franceschetti, O. Jabali, and G. Laporte. Continuous approximation models in freight distribution management. *TOP: An Official Journal of the Spanish Society of Statistics and Operations Research*, 25(3):413–433, October 2017. doi: 10.1007/s11750-017-0456-1. URL https://ideas.repec.org/a/spr/topjnl/v25y2017i3d10.1007_s11750-017-0456-1.html.

M. Garey and D. Johnson. "Strong" NP-Completeness Results: Motivation, Examples, and Implications. *Journal of the ACM*, 25(3):499–508, jul 1978. ISSN 0004-5411. doi: 10.1145/322077.322090. URL https://doi.org/10.1145/322077.322090.

M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., USA, 1979. ISBN 0716710447.

M. X. Goemans, M. Queyranne, A. S. Schulz, M. Skutella, and Y. Wang. Single machine scheduling with release dates. *SIAM J. Discret. Math.*, 15(2):165–192, Feb. 2002. ISSN 0895-4801. doi: 10.1137/S089548019936223X. URL https://doi.org/10.1137/S089548019936223X.

M. Grötschel, L. Lovász, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization*. Springer-Verlag, Berlin, 1993.

A. Hariri and C. Potts. An algorithm for single machine sequencing with release dates to minimize total weighted completion time. *Discrete Applied Mathematics*, 5(1):99 – 109, 1983. ISSN 0166-218X. doi: https://doi.org/10.1016/0166-218X(83)90019-7. URL http://www.sciencedirect.com/science/article/pii/0166218X83900197.

Y. Herer and M. Penn. Characterizations of naturally submodular graphs: A polynomially solvable class of the TSP. *Proceedings of the American Mathematical Society*, 123:673–679, 1995. doi: 10.1090/S0002-9939-1995-1260169-4.

T. Hirayama, Y. Liu, K. Makino, K. Shi, and C. Xu. *A Polynomial Time Algorithm for Finding a Minimum 4-Partition of a Submodular Function*, pages 1680–1691. 2023. doi: 10.1137/1.9781611977554.ch64. URL https://epubs.siam.org/doi/abs/10.1137/1.9781611977554.ch64.

S. Iwata, L. Fleischer, and S. Fujishige. A combinatorial strongly polynomial algorithm for minimizing submodular functions. *Journal of the Association for Computing Machinery*, 48:761–777, 2001.

M. R. Karimi, M. Lucic, H. Hassani, and A. Krause. Stochastic submodular maximization: The case of coverage functions. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 6856–6866, Red Hook, NY, USA, 2017. Curran Associates Inc. ISBN 9781510860964.

M. A. Klapp, A. L. Erera, and A. Toriello. The Dynamic Dispatch Waves Problem for same-day delivery. *European Journal of Operational Research*, 271(2):519–534, 2018a. doi: 10.1016/j.ejor.2018.05.03. URL https://ideas.repec.org/a/eee/ejores/v271y2018i2p519-534.html.

M. A. Klapp, A. L. Erera, and A. Toriello. The one-dimensional dynamic dispatch waves problem. *Transportation Science*, 52(2):402–415, Mar. 2018b. ISSN 1526-5447. doi: 10.1287/trsc.2016.0682. URL https://doi.org/10.1287/trsc.2016.0682.

M. A. Klapp, A. L. Erera, and A. Toriello. Request acceptance in same-day delivery. *Transportation Research Part E: Logistics and Transportation Review*, 143:102083, 2020.

A. Krause and D. Golovin. Submodular Function Maximization. In L. Bordeaux, Y. Hamadi, and P. Kohli, editors, *Tractability: Practical Approaches to Hard Problems*, pages 71–104. Cambridge University Press, 2014. doi: 10.1017/CBO9781139177801.004.

W.-C. Lee and C.-C. Wu. A note on optimal policies for two group scheduling problems with deteriorating setup and processing times. *Computers & Industrial Engineering*, 58(4):646–650, 2010. ISSN 0360-8352. doi: https://doi.org/10.1016/j.cie.2010.01.006. URL https://www.sciencedirect.com/science/article/pii/S0360835210000082.

S. Li, G. Li, X. Wang, and Q. Liu. Minimizing makespan on a single batching machine with release times and non-identical job sizes. *Operations Research Letters*, 33:157–164, 03 2005. doi: 10.1016/j.orl.2004.04.009.

S. Liu, L. He, and Z.-J. M. Shen. On-Time Last-Mile Delivery: Order Assignment with Travel-Time Predictors. *Management Science*, 67:4095–4119, 2020.

C. L. Monma and C. N. Potts. On the complexity of scheduling with batch setup times. *Operations Research*, 37(5):798–804, Oct. 1989. ISSN 0030-364X. doi: 10.1287/opre.37.5.798. URL https://doi.org/10.1287/opre.37.5.798.

G. Nemhauser and L. Wolsey. *Integer and Combinatorial Optimization*. John Wiley & Sons, Inc., 1999.

N. Nisan. Bidding and allocation in combinatorial auctions. In *Proceedings of the 2nd ACM Conference on Electronic Commerce*, EC '00, page 1–12, New York, NY, USA, 2000. Association for Computing Machinery. ISBN 1581132727. doi: 10.1145/352871.352872. URL https://doi.org/10.1145/352871.352872.

G. Owen. On the core of linear production games. *Mathematical Programming*, 9:358–370, 12 1975. doi: 10.1007/BF01681356.

C. K. Poon and P. Zhang. Minimizing makespan in batch machine scheduling. *Algorithmica*, 39(2): 155–174, feb 2004. ISSN 0178-4617. doi: 10.1007/s00453-004-1083-4. URL https://doi.org/10.1007/s00453-004-1083-4.

D. Reyes, A. L. Erera, and M. W. Savelsbergh. Complexity of routing problems with release dates and deadlines. *European Journal of Operational Research*, 266(1):29–34, 2018. ISSN 0377-2217. doi: https://doi.org/10.1016/j.ejor.2017.09.020. URL https://www.sciencedirect.com/science/article/pii/S037722171730841X.

A. Schrijver. A combinatorial algorithm minimizing submodular functions in strongly polynomial time. *Journal of Combinatorial Theory, Series B*, 80(2):346–355, 2000. ISSN 0095-8956. doi: https://doi.org/10.1006/jctb.2000.1989.

A. Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*. Springer, Berlin, 2003.

M. Schutten, S. Van De Velde, and W. Zijm. Single-machine scheduling with release dates, due dates and family setup times. *Management Science*, 42:1165–1174, 11 1996. doi: 10.1287/mnsc.42.8.1165.

L. Shapley. On balanced sets and cores. *Naval Research Logistics Quarterly*, 14(4):453–460, 1967. doi: https://doi.org/10.1002/nav.3800140404. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/nav.3800140404.

L. Shapley. Cores of convex games. *International Journal of Game Theory*, 1:11–26, 1971.

B. C. Shelbourne, M. Battarra, and C. N. Potts. The vehicle routing problem with release and due dates. *INFORMS Journal on Computing*, 29(4):705–723, 2017. doi: 10.1287/ijoc.2017.0756. URL https://doi.org/10.1287/ijoc.2017.0756.

A. Stroh, A. Erera, and A. Toriello. Tactical design of same-day delivery systems. *Management Science*, 68: 3444–3463, 2022.

X. Sun, K. Li, and W. Li. The vehicle routing problem with release dates and flexible time windows. *Engineering Optimization*, 2021. doi: 10.1080/0305215X.2021.1974853. Forthcoming.

T. Vanelslander, L. Deketele, and D. Hove. Commonly used e-commerce supply chains for fast moving consumer goods: comparison and suggestions for improvement. *International Journal of Logistics*, 16: 243–256, 06 2013. doi: 10.1080/13675567.2013.813444.

S. Voccia, A. Campbell, and B. Thomas. The same-day delivery problem for online purchases. *Transportation Science*, 53(1):167–184, 2019.

J.-Q. Wang, G. Fan, and Z. Liu. Mixed batch scheduling on identical machines. *Journal of Scheduling*, 23, 08 2020. doi: 10.1007/s10951-019-00623-9.

S. Wang, T. Zhou, C. Lavania, and J. A. Bilmes. Constrained robust submodular partitioning. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 2721–2732. Curran Associates, Inc., 2021. URL `https://proceedings.neurips.cc/paper_files/paper/2021/file/161882dd2d19c716819081aee2c08b98-Paper.pdf`.

S. Webster and K. R. Baker. Scheduling Groups of Jobs on a Single Machine. *Operations Research*, 43:692–703, 1995.

K. Wei, R. Iyer, S. Wang, W. Bai, and J. Bilmes. Mixed robust/average submodular partitioning: Fast algorithms, guarantees, and applications. NIPS'15, page 2233–2241, Cambridge, MA, USA, 2015. MIT Press.

M. Wölck and S. Meisel. Branch-and-Price Approaches for Real-Time Vehicle Routing with Picking, Loading, and Soft Time Windows. *INFORMS Journal on Computing*, 2022. doi: 10.1287/ijoc.2021.1151. Forthcoming.

S.-J. Yang and D.-L. Yang. Single-machine group scheduling problems under the effects of deterioration and learning. *Computers & Industrial Engineering*, 58(4):754–758, 2010. ISSN 0360-8352. doi: https://doi.org/10.1016/j.cie.2010.02.006. URL `https://www.sciencedirect.com/science/article/pii/S0360835210000306`.

J. Yuan, Z. Liu, C.-T. Ng, and T. Cheng. Single machine batch scheduling problem with family setup times and release dates to minimize makespan. *Journal of Scheduling*, 9:499–513, 12 2006. doi: 10.1007/s10951-006-8776-2.

A Proofs from Section 3

A.1 MILP solves SAD

Proposition 2. *The following MILP solves SAD:*

$$\min_{t,x,z\geq 0} \; z \tag{2a}$$

$$\text{s.t. } t_i \geq r_i \qquad i \in N \tag{2b}$$

$$t_{i+1} \geq t_i + \sum_{S\in\mathcal{N}_i} x_S f(S) \qquad i \in N \setminus n \tag{2c}$$

$$z \geq t_n + \sum_{S\in\mathcal{N}_n} x_S f(S) \tag{2d}$$

$$\sum_{S\subseteq N, S\ni i} x_S = 1 \qquad i \in N \tag{2e}$$

$$x_S \in \{0,1\} \qquad \emptyset \neq S \subseteq N. \tag{2f}$$

Proof. We show the equivalence between feasible solutions for the MILP and feasible solutions for SAD. First, we start from a solution for the MILP, and construct a solution for SAD. For each $i \in N$, there can be at most one subset $S_i \in \mathcal{N}_i$ such that $x_{S_i} = 1$, otherwise constraint (2e) would be violated. Construct a solution for SAD as follows: for $i \in N$, define the i-th dispatch as (t_i, S_i) if there exists some $S_i \in \mathcal{N}_i$ with $x_{S_i} = 1$, and as $(t_i, \emptyset)$ otherwise. This ordered set of dispatches is a feasible solution for SAD since (2e) ensures the batches partition N, and (2b) and (2c) correspond exactly to the feasibility constraints for SAD in the proof of Proposition 1. Finally, constraint (2d) ensures the makespan is also feasible. Therefore, the MILP solution and the solution for SAD have the same makespan.

Now assume we have feasible solution for SAD, (t_i, S_i) for $i \in N$. By Proposition 1, we may assume without loss of optimality that the batches are dispatched following their indexing, and thus $t_1 \leq \cdots \leq t_n$. We construct a solution for the MILP by keeping the same values t_i, initializing vector x to be zero, and then setting $x_{S_i} = 1$ for $S_i \neq \emptyset$. Because the original solution for SAD is feasible, the vector x satisfies (2e) and (2b) is satisfied by t. There is at most one non-zero x_S in any of the constraints (2c) and (2d), implying those constraints reduce to the feasibility constraints of SAD, and so they are also satisfied by t. Finally, by setting z as the value that achieves equality at constraint (2d), we get a feasible solution for the MILP with the same makespan as the SAD solution. □

A.2 Strong NP-Hardness Proof

Proposition 3. *SAD is strongly NP-Hard, even in the special case where f is a Steiner TSP in a generalized "star" graph, a tree where a depot node has arbitrary degree but all other nodes have degree one or two.*

Proof. Consider the 3-partition problem (3-PP), which is known to be strongly NP-hard (Garey and Johnson, 1979). An instance of 3-PP is described by two positive integers a and h and a set $S = \{a_1, a_2, \ldots, a_{3h}\}$, where each a_i is a positive integer with $a/4 < a_i < a/2$ for all $i = 1, \ldots, 3h$. The question is whether it is possible to find a partition of S into subsets $S_1, S_2, \ldots, S_h$ such that $\sum_{i \in S_\ell} a_i = a$, for all $\ell = 1 \ldots, h$.

A problem is strongly NP-Hard if there is a pseudo-polynomial transformation to it from another strongly NP-hard problem. In our case, we prove that SAD on generalized "star" graphs is strongly NP-Hard by exhibiting a pseudo-polynomial transformation from 3-PP, which requires the four following conditions (Garey and Johnson, 1978):

(a) For all instances of 3-PP, the 3-PP instance has a "Yes" answer if and only if the corresponding transformed instance in SAD has a "Yes" answer.

(b) The transformation can be computed in time bounded by a polynomial of the instance input length, $3h + 1$, and the largest number found in the instance, a.

(c) There is a polynomial function $p(\cdot)$ such that, for every 3-PP instance, the length of the input in 3-PP is smaller or equal to $p(\cdot)$ applied to the length of the input in the SAD instance.

(d) There is a polynomial function $q(\cdot)$ such that the largest number in the SAD instance is smaller or equal to $q(\cdot)$ applied to the tuple (size, largest number) of the 3-PP instance.

We reduce 3-PP to the decision version of SAD with dispatch times given by a generalized "star" graph using the following transformation. Consider an arbitrary 3-PP instance; for each number a_i, create a path in the SAD instance with exactly h nodes, which we denote as $v_{i,j}$ for $j = 1, \ldots, h$. The second index j represents the position of the node in the path i: $v_{i,h}$ is the farthest node from the depot in path i and $v_{i,1}$ is the closest node to the depot. Each node $v_{i,h}$ has a release time of 0 and a round-trip from the depot takes a_i time units; node $v_{i,j}$ has a release time $(h-j)a$ and the round-trip from the depot takes ja_i/h time. Intuitively, every a units of time a new order appears in each path and that order is closer to the depot than the previous one. At time $(h-1)a$, the last set of $3h$ orders are released (nodes $v_{i,1}$). The dispatch time to a set of nodes in different paths is just the sum of the times incurred by visiting each path individually, and therefore the dispatch time function is the routing time in a generalized "star" graph.

We prove now that this is a valid pseudo-polynomial transformation from 3-PP to SAD on generalized "star" graphs. We start by showing that a 3-PP instance has a "Yes" answer if and only if the corresponding generalized "star" SAD instance has makespan smaller or equal to $a(h + \frac{h-1}{2})$.

First, consider a solution for the 3-PP problem, given by the partition $S_1, \ldots, S_h$. We construct a solution for SAD as follows. The first dispatch leaves the depot at time 0 and includes nodes $v_{i,h}$ from the three paths corresponding to $i \in S_1$. The dispatch time to those three nodes is $\sum_{i \in S_1} a_i = a$, and this first dispatch ends at time a. The second dispatch includes all available nodes from paths $i \in S_2$, $v_{i,h}$ and $v_{i,h-1}$. This dispatch also takes a time because visiting $v_{i,h}$ in each path takes a_i time units, and $\sum_{i \in S_2} a_i = a$. We keep selecting paths in $S_3, \ldots, S_h$ for the third through h-th dispatch, respectively, and each of those dispatches takes a time and ends at time $3a, \ldots, ah$, respectively, covering all paths since the sets S_j are a partition of S. When the h-th dispatch ends, all remaining uncovered nodes are available; in paths $i \in S_1$ this means we need to dispatch up to the three nodes $v_{i,h-1}$, for $i \in S_2$ to the three nodes $v_{i,h-2}$, and so on, ending in paths $i \in S_{h-1}$ with the three nodes $v_{i,1}$. A dispatch that includes all these nodes has a dispatch time equal to $\sum_{j=1}^{h-1} \frac{h-j}{h} a$, because the farthest remaining node in path $i \in S_j$ is the node whose round-trip takes $\frac{(h-j)}{h} a_i$ time units. Thus, the solution's makespan is $ah + \sum_{j=1}^{h-1} \frac{h-j}{h} a = a(h + \frac{h(h-1)}{2h}) = a(h + \frac{h-1}{2})$. It follows that if the 3-PP problem is feasible, there exists a solution with the desired makespan for SAD.

Consider now a solution for SAD with makespan less than or equal to $a(h + \frac{h-1}{2})$. We claim that this makespan can only be achieved by instances corresponding to feasible 3-PP instances, and that it is actually the minimum makespan for such instances. Notice that it is always feasible to wait until time $a(h-1)$ and then include all nodes in each path in a single dispatch, with a total dispatch time of $\sum_{i=1}^{3h} = ah$, inducing a makespan of $a(h-1) + ah$. Any solution with a makespan of $a(h + \frac{h-1}{2})$ must therefore "save" $a\frac{h-1}{2}$ dispatch time between time 0 and $a(h -$

1), because starting at time $a(h-1)$ we can dispatch all orders that have been not dispatched yet. Furthermore, any dispatch starting during interval $[0, a)$ contains only nodes $v_{i,h}$, the only nodes released before time a; therefore, after any such dispatch is finished, paths i visited by that dispatch have $v_{i,h-1}$ as the farthest uncovered node, requiring a round-trip of $\frac{h-1}{h}a_i$ time. Hence, the total time saved by that dispatch with respect to the solution that serves all orders together is its dispatch time multiplied by $1/h$. Similarly, any dispatch starting during interval $[a, 2a)$ can save at most its dispatch time multiplied by $2/h$, which occurs only when the new dispatch does not repeat a path from the previous dispatch. In general, any dispatch starting in the interval $[ai, a(i+1))$ can save at most its dispatch time multiplied by $(i+1)/h$, for $i = 0, \ldots, (h-2)$, and that saving is obtained only if the paths visited by that dispatch are visited for the first time.

The maximum total time saved by these possible dispatches is obtained by saving the maximum possible time in each interval, which is exactly $a\sum_{i=1}^{h-1}\frac{i}{h} = a\frac{h-1}{2}$. This implies that a solution with makespan $a(h+\frac{h-1}{2})$ must precisely save the maximum possible time in each interval $[ai, a(i+1)]$: it must have no idle time (no time can be saved while the vehicle idles); it must have a dispatch finishing exactly at each of the respective times ai, for $i = 1, \ldots, h-1$; and each of these dispatches must include paths that have not been visited before. We can thus describe each interval $[ai, a(i+1)]$ by the paths the dispatch visits during this time, and as these paths have not been visited previously, the total dispatch time equals the sum of the round-trip times to the farthest unvisited node in each of those paths; since this time equals a, the sum of the a_i values for these paths must be exactly a. As each path is visited only once during the interval $[0, a(h-1)]$, we can describe the paths served during interval $[a(i-1), ai]$ for $i = 1, \ldots h-1$ as subsets S_i, and by denoting the unserved paths as S_h, we form a partition for S. Furthermore, because $a/4 < a_j < a/2$ for all $j = 1, \ldots, 3h$, each those subsets S_i has exactly three elements, and so the partition S_i is a solution for 3-PP. Therefore, if there exists a solution for SAD with makespan equal to $a(h+\frac{h-1}{2})$, the 3-PP instance has a partition meeting the desired requirements.

This concludes the proof of condition (a). The transformation generates a SAD instance defined using $3h^2$ numbers, and all these numbers are bounded by $(h-1)a$; therefore (b), (c) and (d) hold. We conclude that the transformation is pseudo-polynomial. □

For illustration, we present an example where the 3-PP instance has $h = 2$. Figure 7 presents the graph used for the SAD instance, where all nodes in the outer layer have their orders released at time 0 and the orders corresponding to the nodes in the inner layer are released at time a. The goal is a set of order batches with corresponding makespan equal to $2.5a$. The makespan when batching all orders together is equal to $3a$, with the dispatch leaving at time a; to get the desired solution at least one dispatch must start earlier, and during the time interval $[0, a)$, $a/2$ units of time must be "saved" compared to the one-dispatch solution.

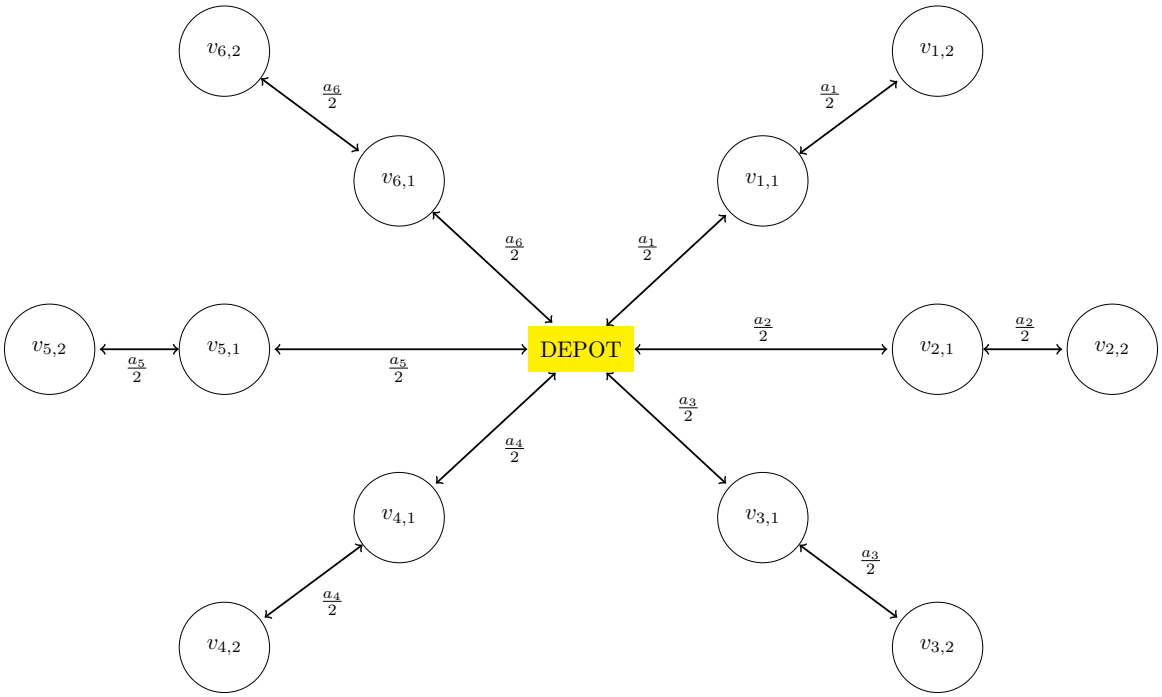


Figure 7: Example of SAD instance derived from 3-PP with $h = 2$.

A.3 Approximation Guarantee with Limited Dispatches

Proposition 5. *Consider a heuristic for SAD that, for any instance, generates a solution with at most $d < n$ dispatches. The heuristic's multiplicative approximation guarantee cannot be smaller than $1 + 1/d$, even when constraining f to be a modular set function.*

Proof. Consider an instance with n orders, modular dispatch time function $f(S) = |S|$ and release times $r_i = i - 1$ for $i = 1, \ldots, n$. We may assume without loss of generality that the solution has no idle time after the first dispatch leaves the depot. Because of this fact and the modular dispatch time function, the best solution with $d < n$ dispatches is equivalent to finding the minimum time t such that we can do d consecutive dispatches that cover all orders.

At any time t before the first dispatch, the number of released orders is $\lfloor t + 1 \rfloor$. Furthermore, after dispatching $|S|$ orders, the number of orders that were released during the dispatch is exactly $|S|$, as long as the dispatch returns by time $n - 1$. It follows that we can get a lower bound on the solution's makespan by assuming that for any t before the first dispatch, the number of orders that have been released is $t + 1$ instead of $\lfloor t + 1 \rfloor$, and by then precisely arranging the d dispatches so that they incur no idle time: the departure time of each dispatch is equal to the release time of the last order that has been released up to that point. This translates to the equation $t + (d - 1)(t + 1) = n - 1$, where t is the start time of the first dispatch, then $d - 1$ dispatches each with dispatch time $(t + 1)$ are executed (as that is the number of orders released between dispatches), and the last dispatch starts when the last order is released. This yields $t = n/d - 1$, and the lower bound on the makespan is $n + n/d - 1$ because n is the total dispatch time of all dispatches.

The optimal solution uses n consecutive dispatches, starting at time zero and with no idle time between dispatches, giving an optimal makespan of n. Therefore, the approximation ratio of a heuristic that executes at most d dispatches cannot be better than $1 + 1/d - 1/n$, which goes to $1 + 1/d$ as n goes to infinity. □

A.4 Solving the LP relaxation

Lemma 6. *Consider the linear relaxation of* (2)*, where we relax the integrality of each x_S variable* (2f) *to non-negativity. This relaxation can be solved in polynomial time if condition (B) holds, that is, if $\min_{S \subseteq N}\{f(S) - \sum_{i \in S} \gamma_i\}$ can be solved in polynomial time for all $\gamma \in \mathbb{R}^n$.*

Proof. The dual of the LP relaxation of (2) is

$$\max_{\alpha,\beta,\gamma} \sum_{i \in N} (\alpha_i r_i + \gamma_i) \tag{6a}$$

$$\text{s.t. } \alpha_1 - \beta_1 \leq 0 \tag{6b}$$

$$\alpha_i + \beta_{i-1} - \beta_i \leq 0 \qquad i = 2, \ldots, n \tag{6c}$$

$$\beta_n \leq 1 \tag{6d}$$

$$-\beta_i f(S) + \sum_{j \in S} \gamma_j \leq 0 \qquad i \in N, S \in \mathcal{N}_i \tag{6e}$$

$$\alpha, \beta, \gamma \in \mathbb{R}^N,\ \alpha, \beta \geq 0. \tag{6f}$$

Here, α are the dual variables for constraints (2b), β for (2c) and (2d), and γ for (2e). Because the number of variables x_S in the primal is exponential, the dual has an exponential number of constraints (6e).

The separation problem for (6e) is

$$\min_{i \in N} \left\{ \min_{S \in \mathcal{N}_i} \left\{ \beta_i f(S) - \sum_{j \in S} \gamma_j \right\} \right\}.$$

From the equivalence of separation and optimization (Grötschel et al., 1993), it follows that the LP relaxation of (2) is solvable in polynomial time if and only if this separation problem is solvable in polynomial time for any $\beta, \gamma \in \mathbb{R}^N$ with $\beta \geq 0$. We conclude the proof by showing that if condition (B) holds, the separation problem is polynomially solvable.

It suffices to show that the inner minimization, $\min_{S \in \mathcal{N}_i} \{\beta_i f(S) - \sum_{j \in S} \gamma_j\}$ for a given $i \in N$, is polynomially solvable under condition (B), as we can enumerate over all $i \in N$. We may assume $\beta_i > 0$, as otherwise the problem is trivial, and thus we may divide by β_i to obtain the equivalent problem $\min_{S \in \mathcal{N}_i} \{f(S) - \sum_{j \in S} \hat{\gamma}_j\}$, where $\hat{\gamma}_j = \gamma_j / \beta_i$ for $j \leq i$.

In the problem described in condition (B), the monotonicity of f implies that if $\gamma_j < 0$, j is not in the minimizing set. Similarly, if $\gamma_j = M > 0$ is large enough, we ensure j is in the minimizing

set; one such value is $M = f(N) + 1$. Define $\hat{\gamma}_j = -1$ for $j > i$ and redefine $\hat{\gamma}_i = M$; we then have

$$\operatorname*{argmin}_{S \in \mathcal{N}_i} \left\{ \beta_i f(S) - \sum_{j \in S} \gamma_j \right\} = \operatorname*{argmin}_{S \subseteq N} \left\{ f(S) - \sum_{j \in S} \hat{\gamma}_j \right\}. \qquad \square$$

B Proofs from Section 4

B.1 Proof of Proposition 7

Proposition 7. *Let (t^{LP}, x^{LP}, z^{LP}) be a feasible extreme point of the LP relaxation of* (2)*. Using that solution as input, Algorithm 1 returns an ordered list (by increasing maximum index, then decreasing cardinality) of the* fractional dispatches*, batches $S \subseteq N$ with $x_S^{LP} > 0$, and returns their corresponding schedule (start, end and idle times) in $O(n \log n)$ time.*

Proof. Because (t^{LP}, x^{LP}, z^{LP}) is an extreme point, there are at most $2n$ variables x_S^{LP} with non-zero value. We can group the corresponding batches according to their highest index in $O(n)$ time, obtaining a (possibly empty) list of batches $\mathcal{B}_j$ for each $j \in N$; each batch $S \in \mathcal{B}_j$ corresponds to a non-zero x_S^{LP} variable with $j = \max\{k \in S\}$. See Figure 1a for an example. Furthermore, for each $j \in N$ we can sort the batches in $\mathcal{B}_j$ according to their cardinality in non-increasing fashion, leading to an ordering (and indexing) of the fractional dispatches; this requires $O(n \log(n))$ time (lines 1 to 5 in Algorithm 1).

With this ordering of the batches, we can determine the start and end time of each "fractional dispatch" by using the dispatch times $f(S)$ and the values x_S^{LP}, which takes $O(n)$ time (lines 7-16 of Algorithm 1). This leads to a schedule for the associated solution, including the idle time Δ_j before the release time of order j (line 12); idle time can only occur once per order $j \in N$, if $\hat{t}_\ell = r_j > \hat{e}_{\ell-1}$. Figure 1b shows an example of the ordering of the fractional dispatches and the respective idle times. $\square$

B.2 Proof of Proposition 8

Proposition 8. *Let (t^{LP}, x^{LP}, z^{LP}) be an optimal extreme point solution of the LP relaxation of* (2) *for some instance I_0. Let η and Δ respectively be the vectors of batch schedule positions and idle times computed by Algorithm 1. Suppose $\sum_{j \in N} \Delta_j > 0$; then Algorithm 2 creates an instance I_1 in $O(n)$ time for which x^{LP} induces an optimal solution and the schedule given by η has no idle time. Therefore, instance I_1's optimal fractional makespan is $z^{LP} - \sum_{j \in N} \Delta_j$.*

Proof. Algorithm 2 creates a new instance I_1 by keeping the same order set N, but modifying the vector r, and as it iterates once for each subset S with $x_S^{LP} > 0$, the overall complexity is $O(n)$.

Let k be the last index such that $\Delta_k > 0$. From monotonicity and the definition of Δ_k, all orders $1, \ldots, k-1$ must be completely dispatched by r_k, otherwise the solution is not optimal, as we could use the idle time to add or increase a fractional dispatch. Therefore, in the original instance I_0, all fractional dispatches starting at or after r_k must constitute an optimal fractional solution for

the sub-instance defined by orders $k, \ldots, n$; otherwise, the makespan for I_0 could be decreased, a contradiction.

When creating the new instance I_1 we make sure that:

1. All orders $k, \ldots, n$ have their release time decreased exactly by $\sum_j \Delta_j$; combined with the optimality of x^{LP} starting at time r_k, this implies that the makespan cannot be reduced by more than $\sum_{j \in N} \Delta_j$, i.e. the new instance's makespan cannot be less than $z^{LP} - \sum_{j \in N} \Delta_j$.

2. The vector x^{LP} and the ordering of fractional dispatches η remain feasible for the new instance, as the release times were not increased, and the sequence of order arrivals was preserved (order i in I_0 remains order i in I_1).

3. The modified vectors $\hat{t}, \hat{e}$ define a fractional schedule for x^{LP} with the updated release time vector r. In this schedule, dispatches including orders $1, \ldots, k-1$ end precisely at the updated release time r_k, and the schedule has no idle time.

The solution x^{LP} and updated schedule are feasible for I_1 and the fractional makespan is precisely $z^{LP} - \sum_{j \in N} \Delta_j$. □

B.3 Proof of Theorem 9

Theorem 9. *Let (t^{LP}, x^{LP}, z^{LP}) be an extreme point optimal solution of the LP relaxation of (2), and define $\delta := z^{LP} - \sum_{S \subseteq N} x_S^{LP} f(S) = \sum_{j \in N} \Delta_j \geq 0$. If f is fractionally subadditive – condition (A2) – the two-dispatch solution obtained by Algorithm 3 has a tight $3/2 - \delta/(2z^{LP})$ approximation ratio. Furthermore, let $B = \{i \in N : r_i \leq \frac{z^{LP} - \delta}{3}\}$; if $f(B) \leq \frac{z^{LP} - \delta}{3}$, the solution obtained by Algorithm 4 has a tight $4/3 - \delta/(3z^{LP})$ approximation ratio. The algorithms run in $O(n \log n)$ time.*

Appendix B.3.1 establishes the approximation ratio for Algorithm 3, and Appendix B.3.2 for Algorithm 4.

B.3.1 Approximation Ratio of Algorithm 3

Proof. Consider an arbitrary instance of SAD, and let (t^{LP}, x^{LP}, z^{LP}) be an extreme point optimal solution for the linear relaxation of (2). We focus first on the case where the solution has no idle time, $\delta = 0$, and afterwards extend the result to $\delta > 0$.

By using Algorithm 1 we can get an ordered (fractional) schedule for the dispatches performed in the solution in $O(n \log n)$ time. Figure 8 shows the ordered fractional schedule for the dispatches after the indexing, where the length of the arrow is the interval where the fractional dispatch is executed, and the set on top is the batch being dispatched. For the rest of the proof, we refer to the first batch according to this sequence as S_1, the second batch is S_2, and so on. Because we assume the solution has no idle time, we can define the fractional starting time of each batch as ξ_j, where $\xi_1 = 0$ and $\xi_j = \sum_{k<j} x_{S_k}^{LP} f(S_k)$. Furthermore, there is some batch S_h satisfying $\xi_h < z^{LP}/2 \leq \xi_{h+1}$; in Figure 8 it is the blue dispatch.

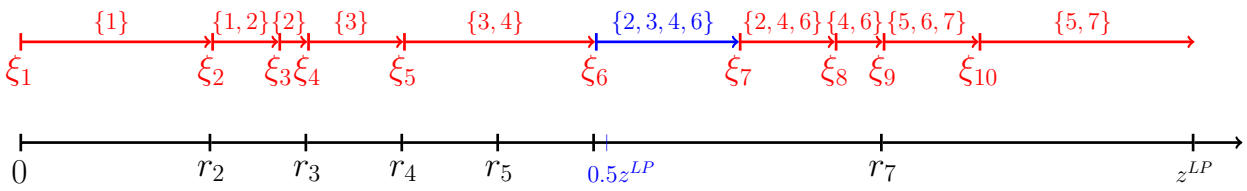


Figure 8: Sequenced fractional dispatches; the blue dispatch corresponds to batch S_h.

We define $D = \bigcup_{k=1}^{h} S_k$, and claim that the two-dispatch solution defined by batches D and $N \setminus D$ has makespan bounded above by $3z^{LP}/2$. Let $r_D = \max_{i \in D}\{r_i\}$ be the release time of the latest order in batch D; from the feasibility of the LP's solution we get $r_D \leq \xi_h$. To prove the claim, we use the non-negativity, monotonicity and fractional subadditivity of f; we consider two separate cases:

1. Suppose $r_D + f(D) \leq z^{LP}$. By construction, all orders in $N \setminus D$ have not been fractionally dispatched before $z^{LP}/2$. From the fractional subadditivity and monotonicity of f, we obtain $f(N \setminus D) \leq \sum_{k>h} x_{S_k}^{LP} f(S_k) \leq z^{LP}/2$, and thus the solution's makespan is bounded above by $3z^{LP}/2$.

2. Now assume $r_D + f(D) > z^{LP}$, so the solution defined by batches D and $N \setminus D$ has makespan of $r_D + f(D) + f(N \setminus D)$. For contradiction purposes, suppose $r_D + f(D) + f(N \setminus D) > 3z^{LP}/2$, which implies

$$f(D) > \frac{3}{2}z^{LP} - r_D - f(N \setminus D) \geq \frac{3}{2}z^{LP} - r_D - \left(z^{LP} - \sum_{k \leq h} x_{S_k}^{LP} f(S_k)\right)$$
$$= \frac{1}{2}z^{LP} + \sum_{k \leq h} x_{S_k}^{LP} f(S_k) - r_D > \sum_{k \leq h} x_{S_k}^{LP} f(S_k) = \xi_{h+1}.$$

The second inequality is a consequence of fractional subadditivity, monotonicity, and the fact that $\delta = 0$. The last inequality follows from the fact that $r_D \leq \xi_h < z^{LP}/2$. We claim that a solution that fractionally dispatches batch D from time 0 up to time ξ_{h+1} (with $x_D = \xi_{h+1}/f(D) < 1$), then fractionally dispatches all orders with $x_N = 1 - x_D$ and finally fractionally dispatches $N \setminus D$ with $x_{N \setminus D} = x_D$ yields a lower bound for z^{LP}. To prove the claim, let I_2 be the instance where orders not included in D with release time earlier than ξ_h have their release time delayed to ξ_{h+1}. Compared to the original instance $I_0 = I_1$ (recall that the solution has no idle time), I_2 cannot have a lower makespan because its release times are equal or later than those in I_1, and so the dispatches from the original instance are also an optimal solution for this new modified instance. We depict an example in Figures 9a, 9b and 9c. Now consider instance I_3, where the release times for orders in D are set to 0, and release times for orders in $N \setminus D$ are set to ξ_{h+1}, as shown in Figure 9d. All the release times in I_3 are smaller or equal to the ones in instance I_2, which means an optimal solution for I_3 provides a lower bound for I_2 (and thus also I_1). Furthermore, fractional subadditivity implies that

our proposed solution is optimal for I_3; Figure 9e shows this for our example. Therefore,

$$z^{LP} \geq \frac{\xi_{h+1}}{f(D)}(f(D) + f(N \setminus D)) + \left(1 - \frac{\xi_{h+1}}{f(D)}\right) f(N)$$
$$\geq \frac{\xi_{h+1}}{f(D)}(f(D) + f(N \setminus D)) + \frac{f(D) - \xi_{h+1}}{f(D)} f(D) = f(D) + \frac{\xi_{h+1}}{f(D)} f(N \setminus D), \qquad (7)$$

where the second inequality follows from monotonicity. For fixed $f(N \setminus D)$ and ξ_{h+1}, the right-hand side of (7) is minimized at $f(D) = \sqrt{\xi_{h+1} f(N \setminus D)}$; however, recalling our assumption that $f(D) > \xi_{h+1} > f(N \setminus D)$, we can actually lower-bound the right-hand side by assuming $f(D) = \xi_{h+1}$, and thus $z^{LP} \geq f(D) + f(N \setminus D)$. Finally, as $r_D \leq \xi_h < \frac{1}{2} z^{LP}$, $r_D + f(D) + f(N \setminus D) \leq 3z^{LP}/2$, which is our desired contradiction.

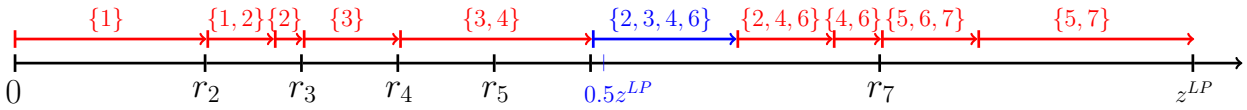
The argument above shows that when $\delta = 0$, the two-dispatch solution given by D and $N \setminus D$ has makespan bounded above by $3z^{LP}/2$. Proposition 5 verifies that this approximation ratio is tight, as we cannot do better with a two-dispatch solution. We now extend the result to when $\delta > 0$. From the original instance I_0, Algorithm 2 constructs an instance I_1 for which the optimal solution x^{LP} has no idle time, with makespan $z^{LP} - \delta$. By using the procedure described above to get D^{I_1}, we can now obtain a solution with makespan $3/2(z^{LP} - \delta)$ for instance I_1. By construction, for all orders $i \in N$ the release times have the following property: $r_i^{I_0} \leq r_i^{I_1} + \delta$, and so the solution given by dispatching D^{I_1} and $N \setminus D^{I_1}$ is feasible for I_0 if the first dispatch starts δ units of time later, and has a makespan of at most $3/2(z^{LP} - \delta) + \delta$ in I_0, which is $3/2 z^{LP} - \delta/2$ and gives the desired result. Finally, Algorithm 1 requires $O(n \log n)$ time, Algorithm 2 requires $O(n)$ time, and finding batch S_h and then finding subset D requires $O(n)$ time; therefore the overall complexity is $O(n \log n)$. □

B.3.2 Approximation Ratio of Algorithm 4

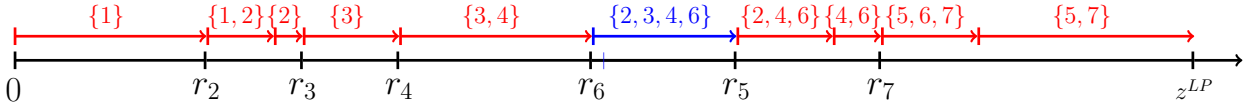
Proof. Consider an arbitrary instance I_0 of SAD, and let (t^{LP}, x^{LP}, z^{LP}) be an extreme point optimal solution for the linear relaxation of (2). By executing Algorithms 1 and 2, we transform I_0 to a modified instance I_1, and get as output an ordered schedule of (fractional) dispatches without idle time described by x^{LP}, with makespan $z_{I_1}^{LP} = z^{LP} - \delta$.

For the rest of the proof we refer to vector r as the vector describing instance I_1, and we refer to the first batch (of the fractional schedule) as S_1, the second batch is S_2, and so on. As in the previous proof, because this solution has no idle time we can define the fractional starting time of each batch as ξ_j, where $\xi_1 = 0$ and $\xi_j = \sum_{k<j} x_{S_k}^{LP} f(S_k)$. Furthermore, there is some batch S_{h_1} satisfying $\xi_{h_1} < z_{I_1}^{LP}/3 \leq \xi_{h_1+1}$, and also some batch S_{h_2} satisfying $\xi_{h_2} < 2z_{I_1}^{LP}/3 \leq \xi_{h_2+1}$; Figure 10a shows a fractional schedule for instance I_1 and Figure 10b highlights batches S_{h_1} and S_{h_2} in blue and violet, respectively.

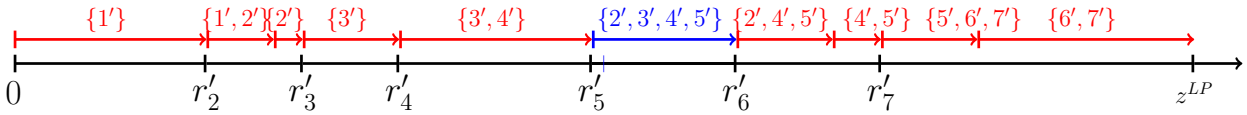
We again use the order subset D defined in the previous proof, and also define $D_1 = \bigcup_{k=1}^{h_1} S_k$, $D_2 = \left(\bigcup_{k=1}^{h_2} S_k\right) \setminus D_1$. We consider the five following solutions:



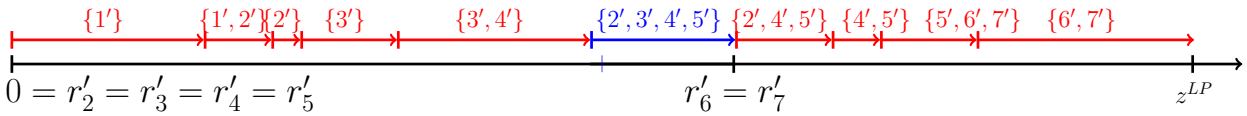
(a) Optimal fractional solution for instance I_1.



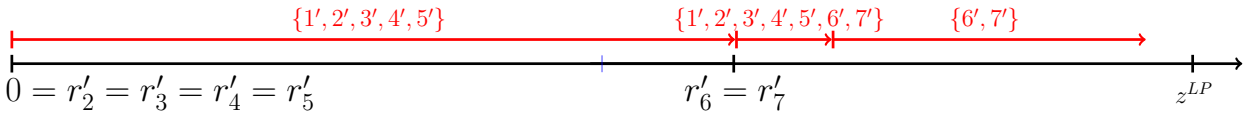
(b) Moving the arrival of order 5 to ξ_{h+1}.



(c) Optimal solution for instance I_2 (after adjusting the indices).



(d) Instance I_3 with the original fractional solution remaining feasible.



(e) Optimal solution for instance I_3.

Figure 9: Visual representation of the lower bound for z^{LP}.

(i) The one-dispatch solution of all orders, N.

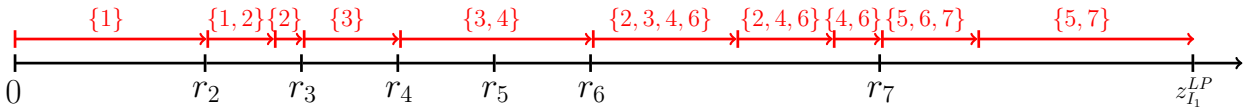
(ii) The two-dispatch solution with batches D and $N \setminus D$ from Algorithm 3.

(iii) The two-dispatch solution with D_1 and $N \setminus D_1$.

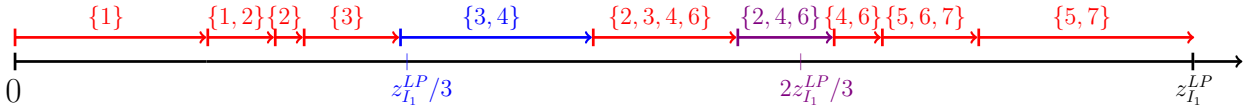
(iv) The two-dispatch solution with $D_1 \cup D_2$ and $N \setminus (D_1 \cup D_2)$.

(v) The three-dispatch solution with D_1, D_2 and $N \setminus (D_1 \cup D_2)$.

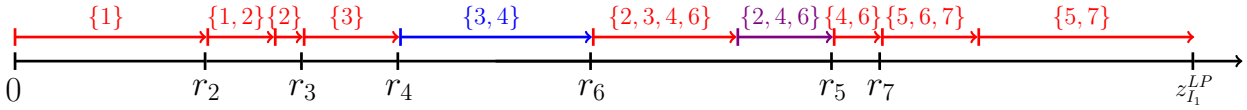
We claim that the solution with the smallest makespan among the five choices has a 4/3



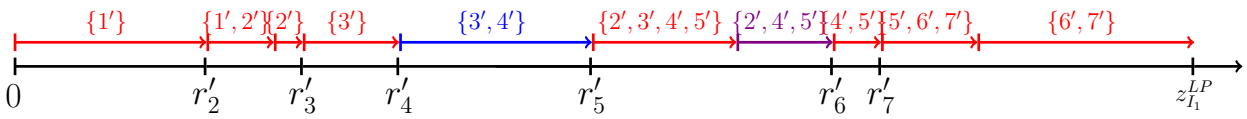
(a) Optimal fractional solution for instance I_1.



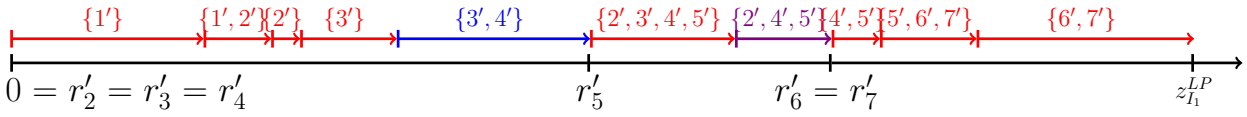
(b) Batch S_{h_1} highlighted in blue and batch S_{h_2} in violet. In this example $D_1 = \{1, 2, 3, 4\}$ and $D_2 = \{6\}$.



(c) Instance I_3 with same optimal solution as I_1. Instances I_1 and I_2 are identical in this case; I_3 increases the release time of order 5.



(d) Instance I_3 with relabeled order indices.



(e) Relaxed instance I_4 for which the original solution is feasible.

Figure 10: Visual representation of batches S_{h_1} and S_{h_2} (blue and violet, respectively), then construction of relaxed instance I_4 based on those batches.

approximation ratio. We assume solution (i) has a makespan greater than $(4/3)z^{LP}_{I_1}$, implying $r_n > z^{LP}_{I_1}/3 > \xi_{h_1}$, so solutions (iii) and (i) are different. We also assume solution (ii) has a makespan greater than $(4/3)z^{LP}_{I_1}$, otherwise the proof is finished. Note that solutions (iii), (iv) and (v) only differ if $D_2 \neq \emptyset$.

Let $r_{D_1} = \max_{i \in D_1}\{r_i\}$, $r_{D_2} = \max_{i \in D_2}\{r_i\} \geq r_{D_1}$; we complete the proof by considering the following cases:

(a) $r_n \leq z^{LP}_{I_1}/3 + \xi_{h_1+1}$.

(b) $r_{D_2} + f(D_1 \cup D_2) \leq z^{LP}_{I_1}/3 + \xi_{h_2+1}$.

(c) $r_{D_2} + f(D_1 \cup D_2) > z^{LP}_{I_1}/3 + \xi_{h_2+1}$ and $r_n > z^{LP}_{I_1}/3 + \xi_{h_1+1}$. We divide this case into multiple sub-cases, with the following conditions:

(c1) $r_{D_1} + f(D_1) + f(D_2) \leq z^{LP}_{I_1}/3 + \xi_{h_2+1}$ and $r_{D_1} + f(D_1) \geq r_{D_2}$.

(c2) $r_{D_1} + f(D_1) + f(D_2) \leq z^{LP}_{I_1}/3 + \xi_{h_2+1}$, $r_{D_1} + f(D_1) < r_{D_2}$ and $r_{D_2} + f(D_2) \leq z^{LP}_{I_1}/3 + \xi_{h_2+1}$.

(c3) $r_{D_1} + f(D_1) + f(D_2) \leq z^{LP}_{I_1}/3 + \xi_{h_2+1}$, $r_{D_1} + f(D_1) < r_{D_2}$ and $r_{D_2} + f(D_2) > z^{LP}_{I_1}/3 + \xi_{h_2+1}$.

(c4) $r_{D_1} + f(D_1) + f(D_2) > z^{LP}_{I_1}/3 + \xi_{h_2+1}$.

We use the following facts in our proof. First, our assumption that $f(B) \leq (z^{LP}_{I_0} - \delta)/3$ implies $f(D_1) \leq z^{LP}_{I_1}/3$. Second, using fractional subadditivity and the definition of D_1 and D_2, $f(N \backslash D_1) \leq z^{LP}_{I_1} - \xi_{h_1+1}$ and $f(N \backslash (D_1 \cup D_2)) \leq z^{LP}_{I_1} - \xi_{h_2+1}$.

For case (a), the makespan of solution (iii) is

$$\max\{r_{D_1} + f(D_1), r_n\} + f(N \setminus D_1) \leq (z^{LP}_{I_1}/3 + \xi_{h_1+1}) + (z^{LP}_{I_1} - \xi_{h_1+1}) = (4/3) z^{LP}_{I_1},$$

where we use $r_{D_1} \leq \xi_{h_1+1}$.

For case (b), we have $z^{LP}_{I_1} < z^{LP}_{I_1}/3 + \xi_{h_2+1}$ by the definition of ξ_{h_2+1}; it follows that solution (iv) has a makespan of

$$\max\{r_{D_2} + f(D_1 \cup D_2), r_n\} + f(N \setminus (D_1 \cup D_2)) \leq (z^{LP}_{I_1}/3 + \xi_{h_2+1}) + (z^{LP}_{I_1} - \xi_{h_2+1}) = (4/3) z^{LP}_{I_1},$$

where we also use $r_n \leq z^{LP}_{I_1}$.

For case (c1), the makespan of solution (v) is

$$\max\{r_{D_1} + f(D_1) + f(D_2), r_n\} + f(N \setminus (D_1 \cup D_2)) \leq (z^{LP}_{I_1}/3 + \xi_{h_2+1}) + (z^{LP}_{I_1} - \xi_{h_2+1}) = (4/3) z^{LP}_{I_1}.$$

Similarly, for case (c2) the makespan of solution (v) is

$$r_{D_2} + f(D_2) + f(N \setminus (D_1 \cup D_2)) \leq (z^{LP}_{I_1}/3 + \xi_{h_2+1}) + (z^{LP}_{I_1} - \xi_{h_2+1}) = (4/3) z^{LP}_{I_1}.$$

For case (c3), $r_{D_2} + f(D_1 \cup D_2) > z^{LP}_{I_1}/3 + \xi_{h_2+1}$ implies $D_2 \neq \emptyset$, because otherwise $D_1 = D_1 \cup D_2$. Solution (v) has makespan $r_{D_2} + f(D_2) + f(N \backslash (D_1 \cup D_2))$; to derive a contradiction, we assume this makespan exceeds $(4/3) z^{LP}_{I_1}$. Similarly to the previous proof, we construct a modified instance based on I_1. In instance I_2, orders with release time earlier than ξ_{h_1+1} and not in D_1 have their release time delayed to ξ_{h_1+1}; I_2 has the same optimal LP solution as I_1. Then, in instance I_3 orders that have a release time in I_2 between ξ_{h_1+1} (inclusive) and $\min\{\xi_{h_1+1} + z^{LP}_{I_1}/3, \xi_{h_2+1}\}$

(exclusive) and are not in D_2 have their release time delayed to $\min\{\xi_{h_1+1} + z^{LP}_{I_1}/3, \xi_{h_2+1}\}$; see Figures 10c and 10d to continue with the example of Figure 10. I_3 also has the same optimal LP solution as I_1. To obtain instance I_4, we relax I_3 by reducing the release time of all orders in D_1 to time 0, all the orders in D_2 to time ξ_{h_1+1}, and all remaining orders to time $\min\{\xi_{h_1+1} + z^{LP}_{I_1}/3, \xi_{h_2+1}\}$; see Figure 10e. As I_4 is a relaxation of I_3, the optimal LP solution of I_4 has a fractional makespan smaller or equal to $z^{LP}_{I_1}$. For this new instance, $f(D_1) \leq z^{LP}_{I_1}/3 \leq \xi_{h_1+1}$ implies the first dispatch is finished before D_2 is released. Furthermore, $f(D_2) > z^{LP}_{I_1}/3 + \xi_{h_2+1} - r_{D_2}$, and therefore $f(D_2) > \min\{\xi_{h_1+1} + z^{LP}_{I_1}/3, \xi_{h_2+1}\} - \xi_{h_1+1}$. The optimal LP solution for I_4 has D_1 dispatched first, then D_2 is fractionally dispatched, then $N \setminus D_1$, and finally $N \setminus (D_1 \cup D_2)$, as depicted in Figure 11.

Let $x_{D_2} = \frac{\min\{\xi_{h_1+1} + z^{LP}_{I_1}/3, \xi_{h_2+1}\} - \xi_{h_1+1}}{f(D_2)}$; then

$$\begin{aligned} z^{LP}_{I_1} \geq z^{LP}_{I_4} &= \xi_{h_1+1} + x_{D_2} f(D_2) + (1 - x_{D_2}) f(N \setminus D_1) + x_{D_2} f(N \setminus (D_1 \cup D_2)) \\ &\geq \xi_{h_1+1} + f(D_2) + \frac{\min\{\xi_{h_1+1} + z^{LP}_{I_1}/3, \xi_{h_2+1}\} - \xi_{h_1+1}}{f(D_2)} f(N \setminus (D_1 \cup D_2)). \end{aligned}$$

The inequality follows from monotonicity and the definition of x_{D_2}. For fixed $f(N \setminus (D_1 \cup D_2))$, ξ_{h_1+1}, ξ_{h_2+1}, $z^{LP}_{I_1}/3$, and by considering the right hand side as a function of $f(D_2)$, the expression is convex and minimized when $f(D_2)$ equals

$$\sqrt{(\min\{\xi_{h_1+1} + z^{LP}_{I_1}/3,\ \xi_{h_2+1}\} - \xi_{h_1+1}) f(N \setminus (D_1 \cup D_2))};$$

however, $f(D_2) > \min\{\xi_{h_1+1} + z^{LP}_{I_1}/3,\ \xi_{h_2+1}\} - \xi_{h_1+1} \geq f(N \setminus (D_1 \cup D_2))$, so by substituting $f(D_2) = \min\{\xi_{h_1+1} + z^{LP}_{I_1}/3,\ \xi_{h_2+1}\} - \xi_{h_1+1}$, we obtain

$$z^{LP}_{I_1} \geq z^{LP}_{I_4} \geq \xi_{h_1+1} + f(D_2) + f(N \setminus (D_1 \cup D_2)).$$

This would imply $(4/3) z^{LP}_{I_1} \geq z^{LP}_{I_4} + z^{LP}_{I_1}/3 \geq z^{LP}_{I_1}/3 + \xi_{h_1+1} + f(D_2) + f(N \setminus (D_1 \cup D_2))$, and as $z^{LP}_{I_1}/3 + \xi_{h_1+1} > r_{D_2}$, we arrive at our desired contradiction.

For case (c4), we again must have $D_2 \neq \emptyset$. As $f(D_1) \leq z^{LP}_{I_1}/3$, $f(D_2) > \xi_{h_2+1} - r_{D_1}$ and hence also $f(D_2) > \min\{\xi_{h_1+1} + z^{LP}_{I_1}/3, \xi_{h_2+1}\} - \xi_{h_1+1}$. Therefore, we can use the same construction and argument from case (c3).

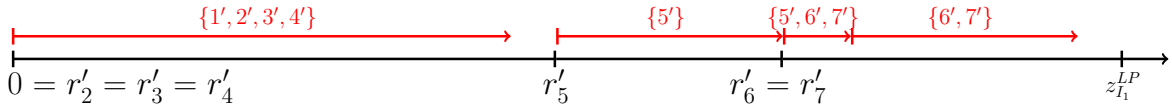


Figure 11: Visual representation of the structure of the optimal solution of I_4

Cases (a), (b) and (c) together cover all possible scenarios in which $f(D_1) \leq z^{LP}_{I_1}/3$, and show that at least one of the solutions from Algorithm (4) meets the desired approximation ratio. It follows that when $f(D_1) \leq (z^{LP} - \delta)/3$, the algorithm returns a solution that has a makespan of

at most $(4/3)(z^{LP} - \delta)$ in I_1. By Proposition 8, if we delay the start of such a solution by δ, it is feasible for the original instance I_0 and has a makespan of $(4/3)(z^{LP} - \delta) + \delta = (4/3)z^{LP} - \delta/3$, which completes the proof of the approximation ratio.

Finally, running Algorithm 3 has complexity $O(n \log n)$, finding batches S_{h_1} and S_{h_2} and computing D_1 and D_2 has complexity $O(n)$, and computing the makespan of the five possible solutions has complexity $O(1)$, yielding a total complexity of $O(n \log n)$. □

B.4 Gap between LP relaxation and MILP

Proposition 10. *Let z_I^* be the optimal makespan of SAD for instance I, and z_I^{LP} be the optimal (fractional) makespan of the LP relaxation of* (2) *for instance I. There exists a family of instances $I_1, I_2, \ldots$ such that $\lim_{m\to\infty} z_{I_m}^*/z_{I_m}^{LP} = (\sqrt{3}-1)^{-1}$. Therefore, when $\zeta < (\sqrt{3}-1)^{-1}$ it is not possible to create a ζ-approximation algorithm for SAD based solely on the optimal solution of the LP relaxation of* (2).

Proof. As a reminder, the family of instances is characterized by: $\Lambda = \sqrt{3 - 2/n} - 1$, $r_1 = 0$, $\tau_1 = 1$, and $r_i = \Lambda\frac{i-1}{n-1}$, $\tau_i = \Lambda\frac{n-i+1}{n-1}$ for $i \geq 2$, with $f(S) = \max_{i\in S}\{\tau_i\}$.

Because $\tau_1 > r_n$, any batch containing the first order returns to the depot after r_n. Furthermore, f is FIFO-optimal, so we may assume the first batch includes the first order; therefore, an optimal solution consists of at most two batches. All such solutions have makespan $r_i + \tau_1 + \tau_{i+1} = 1 + \Lambda\frac{i-1}{n-1} + \Lambda\frac{n-(i+1)+1}{n-1} = \Lambda + 1$ if the first batch is $\{1, \ldots, i\}$ for $i < n$, or $r_n + \tau_1 = \Lambda + 1$ if $i = n$ and the solution consists of a single batch; hence all such solutions are optimal and $z^* = 1 + \Lambda$.

Since $\tau_1 > r_n$, the optimal solution of the LP relaxation fractionally dispatches batch $\{1\}$ in the interval $[0, r_2]$, $\{1, 2\}$ in the interval $[r_2, r_3]$ and so forth, until it dispatches N in interval $[r_n, 1]$, and then it dispatches $\{2, 3, \ldots, n\}$ in interval $[1, 1 + r_2\tau_2]$, $\{3, \ldots, n\}$ in interval $[1 + r_2\tau_2, 1 + r_2\tau_2 + (r_3 - r_2)\tau_3]$, and so forth until $\{n\}$. By grouping terms, it follows that the makespan of the LP relaxation is

$$\begin{aligned} z^{LP} &= 1 + r_2(\tau_2 - \tau_3) + \cdots + r_{n-1}(\tau_{n-1} - \tau_n) + r_n\tau_n \\ &= 1 + \sum_{i=2}^{n-1} \frac{i-1}{n-1}\Lambda^2\left[\frac{n-i+1}{n-1} - \frac{n-(i+1)+1}{n-1}\right] + \frac{\Lambda^2}{n-1} \\ &= 1 + \frac{\Lambda^2}{(n-1)^2}\sum_{i=2}^{n-1}(i-1) + \frac{\Lambda^2}{n-1} = 1 + \frac{(n-1)(n-2)\Lambda^2}{2(n-1)^2} + \frac{\Lambda^2}{n-1} = 1 + \frac{n\Lambda^2}{2(n-1)}. \end{aligned}$$

The second equality comes from definition of vectors r, τ. It follows that $z^*/z^{LP} = \frac{1+\Lambda}{1+\frac{n\Lambda^2}{2(n-1)}}$, and when substituting for the value of Λ and letting $n \to \infty$, we get the desired $(\sqrt{3}-1)^{-1}$ ratio. □

B.5 Strengthened Inequalities

Theorem 11. *Inequalities* (2b) *in* (2) *can be strengthened to*

$$t_i \geq r_i + \sum_{j<i} \sum_{S \in \mathcal{N}_j} \max\{0, r_j - r_i + f(S)\} x_S, \quad i = 2, 3, \ldots, n. \tag{3}$$

Proof. We prove the theorem by induction on i. Recall the notation $\mathcal{N}_i = \{S \subseteq \{1, \ldots, i\} : i \in S\}$, and let (t, x, z) be any feasible solution to (2). When $i = 2$, we have

$$t_2 \geq \max\{r_2, r_1 + x_{\{1\}} f(\{1\})\} = r_2 + x_{\{1\}} \max\{0, r_1 - r_2 + f(\{1\})\}.$$

Now suppose $i \geq 3$. If no dispatch occurs before i is released, the inequality is valid trivially. Similarly, if the coefficient in the inequality evaluates to zero for all previous dispatches, the inequality is again trivially valid. Assume this is not the case, and let $k = \max\{j < i : \sum_{S \in \mathcal{N}_j} x_S \max\{0, r_j - r_i + f(S)\} > 0\}$ be the largest index before i for which there is a dispatch with positive coefficient in the inequality. By feasibility, we have

$$t_i \geq t_k + \sum_{S \in \mathcal{N}_k} x_S f(S) = r_i + \left(-r_i + t_k + \sum_{S \in \mathcal{N}_k} x_S f(S) \right).$$

The term inside the parenthesis can be further expanded and bounded below as follows:

$$\begin{aligned}
-r_i + t_k + \sum_{S \in \mathcal{N}_k} x_S f(S) &\geq -r_i + r_k + \sum_{S \in \mathcal{N}_k} x_S f(S) + \sum_{j<k} \sum_{S \in \mathcal{N}_j} x_S \max\{0, r_j - r_k + f(S)\} \\
&\geq -r_i + r_k + \sum_{S \in \mathcal{N}_k} x_S f(S) + \sum_{j<k} \sum_{S \in \mathcal{N}_j} x_S \max\{0, r_j - r_i + f(S)\} \\
&= -r_i + r_k + \sum_{S \in \mathcal{N}_k} x_S f(S) + \sum_{j<i,\ j \neq k} \sum_{S \in \mathcal{N}_j} x_S \max\{0, r_j - r_i + f(S)\} \\
&= \sum_{j<i} \sum_{S \in \mathcal{N}_j} x_S \max\{0, r_j - r_i + f(S)\}.
\end{aligned}$$

The first inequality follows from induction, the second because $r_i \geq r_k$, while the first equation follows from the assumption that any dispatch between k and i has a zero coefficient; the last equation simply groups terms. □

C Proofs from Section 5

C.1 FIFO Algorithm

Proposition 12. *Let κ be the number of operations needed to compute all values $f_{i,j}$, for $1 \leq i \leq j \leq n$. For a SAD instance, the best makespan among FIFO solutions is given by z^{FIFO}, and can be computed in $\Theta(n^2 + \kappa)$ time.*

Proof. First, we claim that $z_{i,j}$ is the minimum makespan required to dispatch orders in N_j in FIFO fashion when the last batch dispatched is $[i, j]$. We prove the claim by induction, and the base case is straightforward as all $z_{1,j}$ are the optimal makespan of dispatching orders $\{1, 2 \ldots, j\}$ in one batch. As induction hypothesis, suppose there exists some $i_0 \in N$ such that for all $i \leq i_0$ and all $j \geq i$, $z_{i,j}$ represents the minimum makespan to dispatch orders in $[1, j]$ in FIFO fashion when the last batch dispatched is $[i, j]$.

Now we consider the values $z_{i_0+1,j}$ for $j \geq i_0 + 1$. The dispatch time of the batch $[i_0 + 1, j]$ is given by $f_{i_0+1,j}$, and the dispatch cannot leave the depot earlier than both r_j and the optimal makespan needed to dispatch orders in $[1, i_0]$; the latter is $\min_{k \leq i_0}\{z_{k,i_0}\}$ because of the induction hypothesis and the fact that orders need to be dispatched in FIFO fashion. Thus $z_{i_0+1,j} = \max\{r_j, \min_{k \leq i_0}\{z_{k,i_0}\}\} + f_{i_0+1,j}$ is the makespan to dispatch orders in $[1, j]$ in FIFO fashion when the last batch is $[i_0 + 1, j]$. It follows that $z_{i,n}$ represents the minimum makespan to dispatch all n orders in a FIFO manner when the last batch dispatched is $[i, n]$, and then the makespan for the best FIFO solution is given by $\min_{i \leq n}\{z_{i,n}\}$.

Assuming we perform κ operations to compute, the $f_{i,j}$ values, the recursion's complexity is dominated by equations (5b), which take quadratic time (for each $i \geq 2$ there are roughly $i - 1$ operations), yielding the complexity. □

C.2 Approximation ratio of the FIFO Algorithm

Theorem 13. *The FIFO Algorithm is a* 2*-approximation for SAD, and this guarantee is tight.*

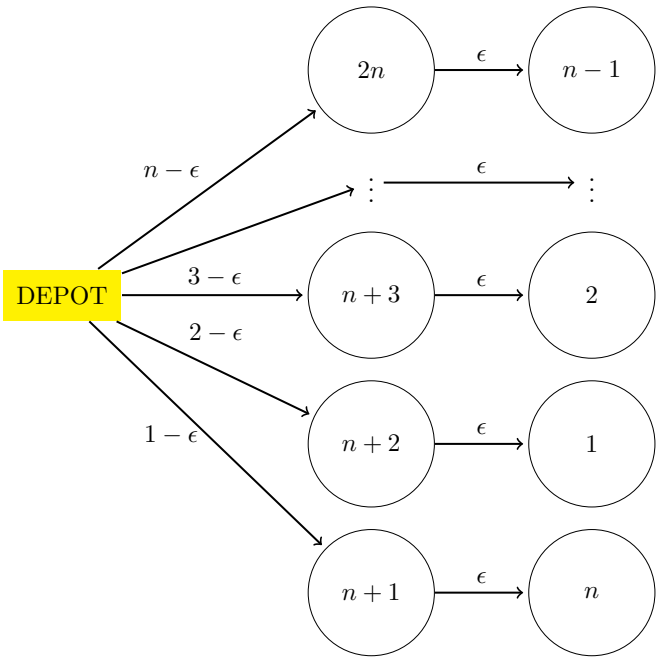


Figure 12: Family of instances in which the best FIFO makespan approaches twice the optimum. In the instance, $r_1 = r_2 = \ldots = r_n = r_{n+1} = 0$, and $r_{n+i} = \sum_{j=1}^{i-1} j = \frac{i(i-1)}{2}$ for $i = 2, 3, \ldots, n$.

Proof. The single-batch solution is a FIFO solution, so the FIFO algorithm inherits the guarantee from Proposition 4. To show that the approximation ratio is tight, we construct a family of instances as depicted in Figure 12. In the figure, the nodes represent orders, the dispatch times correspond to Steiner TSP routing times in the graph and orders have the following release times: $r_1 = r_2 = \ldots = r_n = r_{n+1} = 0$ and $r_{n+i} = \sum_{j=1}^{i-1} j = \frac{i(i-1)}{2}$ for $i = 2, 3, \ldots, n$. We define path 1 as nodes $\{n,\ n+1\}$ and for $i \geq 2$ path i is the set of nodes $\{i-1,\ n+i\}$. It is clear that the optimal solution dispatches to each path only once and so the optimal solution is given by batches $\{n, n+1\}, \{1, n+2\}, \ldots, \{n-1, 2n\}$, dispatched consecutively with no idle times, and has a makespan of $\sum_{i=1}^{n} i = \frac{(n+1)n}{2} = \frac{n^2+n}{2}$.

When constructing a FIFO solution, only one dispatch is needed for path i if and only if the dispatch for both orders in the path leaves the depot at least at time r_{n+i} and includes all orders in between $i-1$ and $n+i$. Because all orders in between are included, FIFO solutions may only have one batch that has both nodes of a path being dispatched together (resulting in only one visit to that respective path). Therefore, to find an optimal FIFO solution we must select the subset of paths that will be visited only once in that solution, and clearly path 1 is always included, as nodes n and $n+1$ would always be between the nodes of other paths. For illustration purposes, consider the solution where only path 1 is served once, which dispatches the batches $[1,\ n+1]$ and $[n+2,\ 2n]$ and has a makespan that is only $(1-\epsilon)$ units of time less than the single-batch makespan. We claim that the optimal FIFO solution is described by waiting until time 1, then consecutively dispatching batches $[1,\ n-2]$ and $[n-1,\ 2n]$ (with no idle time in between), with a makespan of $1 + \sum_{i=2}^{n-1} i + \sum_{i=1}^{n} i - \epsilon(n-2) = \frac{n(n-1)}{2} + \frac{(n+1)n}{2} - \epsilon(n-2) = n^2 - \epsilon(n-2)$. Note that this solution visits paths n and 1 only once and all other paths twice.

To prove the claim, consider adding path $1 < i < n$ to the set of paths visited only once. In order to accommodate that requirement there are two options:

i) *Path n continues to be visited only once:* Under this scenario the departure time of the last dispatch (that must include all paths served only once) is still $n(n+1)/2$, and now as path i is visited only once, then the batches that are previously dispatched cannot include the node $i-1$ (the node farthest away in path i), so the vehicle will incur extra idle time to leave at the required time for the last dispatch, and as the dispatch time of the last batch will have its value increased by ϵ, the makespan increases.

ii) *Path n is now visited twice in the solution:* By letting $i \geq 2$ be the largest index of a path that is visited only once, then paths $2, \ldots, i-1$ are the only other paths that may be visited only once. Note that the batch that includes all paths that are visited only once must leave the depot at least at time r_i, but for that to happen the vehicle will have an idle time of at least one unit even if all the orders $1, \ldots, i-2$ are previously dispatched to (recalling that node $i-1$ is in the batch that includes path i). This implies that for each path $2, \ldots, i-1$ we can actually reduce the makespan by ϵ if we decide to visit that path twice. It follows that in this case the best solution is the one that only has 1 and i as the subset of paths visited once, and is given by the vehicle departing at time 1 and consecutively dispatching the batches

$[1,\ i-2], [i-1,\ n+i], [n+i+1,\ 2n]$, with a cost of $r_i + \sum_{j=1}^{n} j - \epsilon(n-2) + \sum_{j=i+1}^{n} j - \epsilon(n-i) = \frac{(n+1)n}{2} + \frac{(n+1)n}{2} - i - \epsilon(2n-2-i) = n^2 + n - i - \epsilon(2n-2-i)$, which is minimized at $i = n$.

Therefore, the makespan of the best FIFO solution is $n^2 - \epsilon(n-2)$, whereas the makespan of the optimal solution is $(n^2+n)/2$, which gives the tight 2-approximation ratio when n goes to infinity. □

C.3 FIFO-Optimal Functions

Proposition 14. *Let $\tau_0 \geq 0$, $\tau_i > 0$ for $i \in N$, and let $g : \mathbb{R} \to \mathbb{R}$ be a concave non-decreasing function with $g(0) = 0$. The following functions are FIFO-optimal,*

1. $f(S) = \tau_0 + \sum_{i \in S} \tau_i$,
2. $f(S) = \tau_0 + \max_{i \in S}\{\tau_i\}$,
3. $f(S) = \tau_0 + g(|S|)$,

where in each case the function is defined for $S \neq \emptyset$, letting $f(\emptyset) = 0$.

Proof. We prove the claim for each of the functions separately:

1. Consider a non-FIFO optimal solution having batches $S_k = \{i_1, \ldots, i_h\}$ and $S_{k+1} = \{j_1, \ldots, j_k\}$ with a pair (a, b) such that $i_a > j_b$. Move all such orders from batch $k+1$ to k. Notice that batch k continues to start at the same time as in the previous solution, while requiring a larger dispatching time, implying that any idle time before batch $k+1$ cannot increase. Also, the sum of the dispatch times of both batches is equal, and so the makespan cannot be increased by performing this transformation. By executing this procedure sequentially for all such pairs of batches, starting with the smallest indices, we obtain a FIFO solution, and as it cannot have a greater makespan than the optimal solution, it is also optimal.

2. We prove $f(S) = \max_{i \in N}\{\tau_i\}$ is FIFO-optimal, which implies the claim, as we can add the constant τ_0 to all terms τ_i to put the function in this form.

   First, we may assume without loss of optimality that the first dispatched batch includes the order $i = \max\{\operatorname{argmax}_{j \in N}\{\tau_i\}\}$. Suppose this order appears instead in batch $k > 1$; then simply add all orders from preceding batches to this batch. The change cannot increase k's dispatch time, and it can still be dispatched at the same time or earlier.

   Now construct a subset $N' \subseteq N$ as follows. Initialize $N' = \emptyset$, and auxiliary set $C = N$. While C is not empty, add order $i = \max\{\operatorname{argmax}_{j \in C}\{\tau_j\}\}$ to N', then redefine $C \leftarrow [i+1, n]$; N' always includes n. By applying the preceding argument inductively, it follows that we can construct an optimal solution for N by solving SAD for N'. Specifically, suppose $N' = \{i_1, \ldots, i_h\}$, with indexes following the ordering in N; then if a batch in a solution for set N'

contains i_a, the corresponding batch in the solution for N contains $[i_{a-1}+1, i_a]$, or $[1, i_a]$ if $a = 1$.

It follows that we may focus on optimizing SAD for the set N', where orders i in this set are indexed in non-decreasing order of r_i and decreasing order of τ_i. Suppose we have an optimal non-FIFO solution; then in two consecutive batches S_k and S_{k+1} we have $j \in S_k$ and $i \in S_{k+1}$ with $i < j$. As before, we can move j to S_{k+1} without changing the start time of either dispatch or increasing either dispatch time; by applying this operation as many times as necessary, we can convert the solution into a FIFO solution with an equal makespan.

3. Suppose we have a non-FIFO optimal solution, with batches S_k, S_{k+1} including orders $j \in S_k$ and $i \in S_{k+1}$, where $i < j$. By swapping these orders, placing i in the k-th batch and vice versa, we maintain the cardinality of both batches and thus their dispatch times. Furthermore, the starting times of the dispatches are still feasible, because $r_i \leq r_j \leq t_k$ and $t_{k+1} \geq t_k \geq r_j$. Applying this operation as many times as necessary, we obtain a FIFO optimal solution. □

C.4 Q-FIFO-Optimal Algorithm

Theorem 15. *Let Q be a fixed integer. For any Q-FIFO-optimal dispatch function, SAD can be optimized in $\Theta(n^{2Q} + \kappa)$ time, where κ is the time required to compute all interval dispatch times for each set S_q that partitions N.*

Proof. We denote the Q non-intersecting sets of orders defining the Q-FIFO-optimal function as $S_1, S_2, \ldots, S_Q$, with respective release time vectors $r^1, r^2, \ldots, r^Q$. Furthermore, we let $n_1, n_2, \ldots, n_Q$ be the number of orders in each of the Q sets, and we denote the pre-computed dispatch times within the Q sets as $f^1, \ldots, f^Q$, with $f^q_{i,j}$ being the dispatch time of batch $[i, j]$ from set S_q, where $1 \leq i \leq j \leq n_q$. To solve this problem, we use an array z with $2Q+1$ dimensions, where the first index indicates the last set S_q that had a dispatch, whereas the other index pairs represent the order intervals dispatched last for each of the sets; as an example, $z(2, i_1, j_1, i_2, j_2, i_3, j_3)$ represents the minimum makespan when the last dispatch corresponding to set S_q was $[i_q, j_q]$ for $q = 1, 2, 3$, and the very last dispatch was $[i_2, j_2]$. We also use zeroes to indicate no dispatch for that set has yet occurred. The recursion defining z is then

$$z\Big(\sigma, (i_q, j_q)_{q=1}^{Q}\Big) = f^{\sigma}_{i_\sigma, j_\sigma} + \max \begin{cases} r^{\sigma}_{j_\sigma}, \\ \min\big\{z\big(\sigma', (i_q, j_q)_{q\neq\sigma} \cup (k_\sigma, i_\sigma - 1)\big) : k_\sigma \in [1, i_\sigma - 1]; \sigma' \in \{q : i_q \geq 1\}\big\}, \end{cases} \tag{8a}$$

$$\text{if } i_\sigma \geq 2;$$

$$z\Big(\sigma, (i_q, j_q)_{q=1}^{Q}\Big) = f^{\sigma}_{i_\sigma, j_\sigma} + \max\big\{r^{\sigma}_{j_\sigma}, \min\big\{z\big(\sigma', (i_q, j_q)_{q\neq\sigma} \cup (0,0)\big) : \sigma' \in \{q : i_q \geq 1\} \setminus \{\sigma\}\big\}\big\}, \quad \text{if } i_\sigma = 1. \tag{8b}$$

The boundary condition is simply $z(0, \ldots, 0) = 0$. The notation $(i_q, j_q)_{q\neq\sigma} \cup (k_\sigma, i_\sigma - 1)$ means the index pair $(k_\sigma, i_\sigma - 1)$ is appended to the previous vector in the σ-th position. Intuitively, the cur-

rent state identifies that the last dispatch had orders from set S_σ, and thus the recursion minimizes over which previous set had the next-to-last dispatch, and what the previous dispatch to S_σ contained. If the dispatch to S_σ was the first one, in which case $i_\sigma = 1$, there is no previous dispatch to this set, indicated by the previous state having the zero indexes. The optimal makespan of SAD is then

$$\min\Big\{z\Big(\sigma, (i_q, n_q)_{q=1}^{Q}\Big) : \quad i_q = 1, \dots, n_q,\ q = 1, \dots Q; \quad \sigma = 1, \dots, Q\Big\}. \tag{8c}$$

Generalizing the argument from the proof of Proposition 12, the array z tracks the optimal makespan for each partial state. There are $O(n^{2Q})$ states, each with $O(n)$ terms in its recursion, yielding a running time of $O(n^{2Q+1})$.

We can reduce the running time by an order of magnitude by noting that the minimization in (8a) can be computed beforehand. Specifically, for $\sigma \in \{1, \dots, Q\}$, $i_\sigma \geq 2$ and $(i_q, j_q)_{q \neq \sigma}$, define

$$\check{z}\big(\sigma, i_\sigma, (i_q, j_q)_{q\neq\sigma}\big) = \min\big\{z\big(\sigma', (i_q, j_q)_{q\neq\sigma} \cup (k_\sigma, i_\sigma - 1)\big) : k_\sigma \in [1, i_\sigma - 1], \sigma' \in \{q : i_q \geq 1\}\big\}. \tag{8d}$$

We can then replace the minimization in (8a) with the appropriate term from the auxiliary array $\check{z}$. This reduces the number of terms in the original recursions to a constant number (either two or Q), and adds $O(n^{2Q-1})$ auxiliary states with $O(n)$ terms in each minimization, yielding the desired complexity. □

D Details for Column Generation and Pricing Problems

D.1 Column Generation Acceleration

We implement the acceleration procedure from Ben-Ameur and Neto (2007) when solving the linear relaxations of (2) and its strengthening with (3). At every iteration, this procedure uses an incumbent feasible dual solution, denoted δ_f here for convenience, which can be initialized with any feasible solution, such as $\delta_f = 0$, and the infeasible dual solution obtained by the master solve, denoted δ_m. Instead of attempting to separate δ_m, we solve the dual separation problem for the convex combination $\hat{\delta} = \lambda\delta_f + (1 - \lambda)\delta_m$. Intuitively, this solution is likelier to be feasible or at least closer to the dual feasible region. If $\hat{\delta}$ is dual feasible, it necessarily has a larger objective value than δ_f, so it becomes the new feasible incumbent. Otherwise, it is dual infeasible, so we add dual cutting planes (i.e. columns in the primal) and perform another master solve. After preliminary calibration, we used $\lambda = 0.8$ in our experiments.

D.2 Pricing for the LP Relaxation in Experiments

To price the linear relaxation of (2), we need to solve the following problem for each $i \in N$:

$$\min_{S \in \mathcal{N}_i} \left\{ f(S) - \sum_{j \in S} \gamma_j \right\}.$$

We first describe the pricing problem for generalized stars, then extend this to the tree instances described in Section 6.

D.2.1 Generalized Stars

As discussed in Sections 5 and 6, for generalized stars it suffices to consider only batches contained in a single path. The pricing problem thus focuses on the path containing order i, and on other orders $j \in \mathcal{N}_i$ with $\gamma_j > 0$ contained in the same path; we denote this set $P_i \subseteq \mathcal{N}_i$ here. Orders $j \in P_i$ are each associated with a number $\tau_j > 0$, such that $f(S) = \max_{j \in S}\{\tau_j\}$ for $S \subseteq P_i$. We can order the elements of P_i as $j_1, \ldots, j_\ell$ with $\tau_{j_1} < \cdots < \tau_{j_\ell}$, where $i = j_k$ for some $k \leq \ell$. It follows from the structure of f and the assumption that $\gamma_j > 0$ for $j \in P_i \setminus i$ that the optimal set is an interval, $S = [j_1, j_m]$, where $m \geq k$ because i must be included in S. By evaluating the sets sequentially, the minimizer can be computed in time proportional to $\ell = |P_i|$.

D.2.2 Trees from Section 6

Recall that the tree topology we use in the experiments consists of a primary paths, and secondary paths with order positions connected to nodes in the primary path; see Figure 3. Similarly to the generalized star case, for the pricing problem it suffices to focus on the sub-tree rooted at the depot that contains i, and on orders $j \in \mathcal{N}_i$ with $\gamma_j > 0$ contained in the same sub-tree. Following the notation in Figure 3, we denote the non-depot nodes in the primary path as st (for "path start").

The pricing for this topology consists of recursive applications of the algorithm we use for a single path above. For the secondary path P_i in the sub-tree that contains i, we compute its reduced cost ρ_i as above and record the minimizing interval $S_i \subseteq P_i$. For every other secondary path P_k, we use the same algorithm but consider all intervals including the empty one (equivalent to not visiting the path at all), and again record the minimizing interval $S_k \subseteq P_k$ and reduced cost $\rho_k \leq 0$. Each node st_k is the start of two secondary paths, say P_{k_1} and P_{k_2}; we assign st_k a value $\tilde{\gamma}_k = -\rho_{k_1} - \rho_{k_2}$, the negated sum of reduced costs for the two corresponding paths. We also note which node is the start of path P_i; call this node st_i. Finally, we apply the path algorithm again on the primary path, using the values $\tilde{\gamma}_k$, and requiring that the solution contain node st_i. Suppose the subset of nodes st in this solution is denoted T; the batch that minimizes the reduced cost corresponds to $\bigcup_{k \in T} S_k$. This algorithm requires a number of operations proportional to the number of nodes considered in the sub-tree.

D.3 Pricing for the Strengthened LP Relaxation in Experiments

For the linear relaxation of (2) strengthened with (3), the pricing problem is equivalent to

$$\min_{S \in \mathcal{N}_i} \left\{ f(S) + \sum_{j>i} \max\{0, r_i - r_j + f(S)\}\alpha_j - \sum_{j \in S} \gamma_i \right\},$$

for each $i \in N$, where $\alpha_j \geq 0$ for all j. Because of the terms multiplying the α_j values, this minimization is no longer submodular, and is not guaranteed to be solvable in polynomial time.

D.3.1 Generalized Stars

The simpler pricing algorithm still applies, with a modification. As before, it suffices to focus on the path containing i, and on orders $j \in \mathcal{N}_i$ in this path with $\gamma_j > 0$. We again call this set $P_i = \{j_1, \ldots, j_\ell\}$, where $\tau_{j_1} < \cdots < \tau_{j_\ell}$. Once we fix the farthest order visited in the path, this fixes $f(S)$, and in this case also fixes the sum that includes the α terms. Thus, given a farthest included order, we choose to include all closer orders, and therefore consider only intervals $[j_1, j_m]$ that include i, for some m. We evaluate all such intervals and choose the minimizer.

D.3.2 Trees from Section 6

We again consider the sub-tree rooted at the depot that contains i, and all orders $j \in \mathcal{N}_i$ in the same sub-tree with $\gamma_j > 0$. The sub-tree consists of secondary paths $P_1, \ldots, P_\ell$, where each P_k starts at node st_k in the primary path. We solve this pricing problem with an MILP:

$$\begin{aligned}
\min_{d,D,F,v,y,\theta} \quad & F + \sum_{j>i} \theta_j - \sum_{j=1}^{i} y_j \gamma_j \\
\text{s.t. } & y_i = 1 \\
& v_P \geq y_j && j \in \mathcal{N}_i \cap P \\
& D \geq v_{P_k} f(\{st_{P_k}\}) && k = 1, \ldots, \ell \\
& d_P \geq y_j \big(f(\{j\}) - f(\{st_P\}) \big) && j \in \mathcal{N}_i \cap P \\
& F = D + \sum_{k=1}^{\ell} d_{P_k} \\
& \theta_j \geq (r_i - r_j + F)\alpha_j && j > i \\
& \theta \geq 0, v \in \{0,1\}, y \in \{0,1\}.
\end{aligned}$$

Variable y_j indicates whether order j is selected to be in the optimal subset. Variable v_P indicates if path P is visited. Variable D accounts for the routing time in the primary path, and variables d_P account for the routing time in each path P. Variable F represents the total routing time, and finally variables θ account for the non-linear term in the pricing objective multiplying the α values.

D.4 Pricing of the LP Relaxation for Linear Production Functions

Recall the linear production function,

$$f(S) = \min_{y \geq 0} \left\{ \sum_{\ell=1}^{m} y_\ell : Ay \geq \sum_{i \in S} b^i \right\}.$$

Condition (B) requires us to solve $\min_{S \subseteq N} \{ f(S) - \sum_{i \in S} \gamma_i \}$ in polynomial time. Assume without loss of generality that each b^i is integral, and let $B_k = \sum_{i \in N} b^i_k$ denote the total amount of product k when including the full set of orders N. We claim (B) holds when m and each B_k is polynomial in n, and p is constant. This follows from a dynamic programming formulation that has polynomially many states and actions under these conditions; the dynamic program is characterized by:

- Stages $0, 1, \ldots, n$, where in stage $i-1$ we consider adding order i, and in stage n we simply compute the time for the selected batch. Stages have states of type $(i, \psi_1, \psi_2, \ldots, \psi_p)$, with ψ_k denoting the amount of product k required so far, for $k = 1, \ldots, p$ and $\psi_k = 0, \ldots, B_k$.
- Actions $(i-1, \psi_1, \psi_2, \ldots, \psi_p) \to (i, \psi_1, \psi_2, \ldots, \psi_p)$ with cost 0 represent the decision to not include i in the batch, for $i \in N$ and all corresponding states.
- Actions $(i-1, \psi_1, \psi_2, \ldots, \psi_p) \to (i, \psi_1 + b^i_1, \psi_2 + b^i_2, \ldots, \psi_p + b^i_p)$ with cost $-\gamma_i$ represent the decision to include i in the batch, for all $i \in N$ and corresponding states.
- There is a dummy terminal sink node T; actions $(n, \psi_1, \psi_2, \ldots, \psi_p) \to T$ with cost

  $$\min_{y \geq 0} \left\{ \sum_{\ell=1}^{m} y_\ell : Ay \geq \psi \right\}$$

  compute the dispatch time for the selected batch. The initial state is the zero vector.

Under our assumptions, the number of states and actions is polynomial. Computing the costs in the last stage requires solving a polynomial number of linear programs of polynomial size. It follows that condition (B) holds.