Thomas **Bosman**

RELAX ROUND REFORMULATE

Near-Optimal Algorithms for Planning Problems in Network Design and Scheduling

VRIJE UNIVERSITEIT

Relax, Round, Reformulate

Near-Optimal Algorithms for Planning Problems in Network Design and Scheduling

ACADEMISCH PROEFSCHRIFT

ter verkrijging van de graad Doctor
aan de Vrije Universiteit Amsterdam,
op gezag van de rector magnificus
prof.dr. V. Subramaniam,
in het openbaar te verdedigen
ten overstaan van de promotiecommissie
van de School of Business and Economics
op dinsdag 26 november 2019 om 09.45 uur
in de aula van de universiteit,
De Boelelaan 1105

door

Thomas Nascimento Bosman

geboren te Apeldoorn

promotor:	prof.dr. L. Stougie
copromotor:	dr. N.K. Olver

"When I use a word," Humpty Dumpty said, in rather a scornful tone, "it means just what I choose it to mean—neither more nor less."
"The question is," said Alice, "whether you can *make* words mean so many different things."
"The question is," said Humpty Dumpty, "which is to be master—that's all."

— Lewis Caroll *Through the Looking-Glass*

Contents

Introduction

Note: a less technical introduction — covering similar topics, but aimed at the non-expert reader — can be found at the end of this thesis.

This thesis bundles a collection of papers on planning problems in network design and scheduling. The papers have quite diverse application areas, which we will discuss in more detail in subsequent sections, but they all share a common perspective: the study of theoretically efficient algorithms. The overlap between the techniques used in these algorithms provides the cohesive bond between the chapters, though each subfield has its own emphasis. In fact, cross-pollination of ideas picked up from different fields was central in cracking some of the problems in this thesis. The *Relax, Round or Reformulate* in the title alludes to the most common paradigms found among these techniques.

Relaxing the problem, or weakening some of the constraints, is a bread and butter technique that has the linear programming (LP) relaxation as its primary example. This type of relaxation is very useful since it can be mechanically applied once an integer programming formulation is known, and is therefore often a sensible first starting point. Though potentially stronger (nonlinear) convex relations are known, linear relaxations can be easier to work with while providing sufficient power for approximation purposes. An interesting departure from this rule occurs in Chapter 2, where we use a nonlinear convex relaxation[1] for convenience, even though it turns out the be equivalent to a more direct linear programming relaxation.

[1] Namely, Relaxation (2.11) based on the Lovász extension.

We also use more *custom made* relaxations, that are are conceived with a particular algorithmic idea in mind, instead of following from relaxing an integrality constraint, say. For example, in Chapter 2 we use a relaxed formulation whose solutions interpolate between a fractional solution of the LP relaxation and an integral solution.

Rounding refers to two different concepts. One of the most prominent techniques in approximation algorithms is the rounding of a fractional relaxation.[2] In Chapter 4, we use a randomized rounding approach inspired by approximation algorithms, but end up with an approximation ratio of 1, thus proving the underlying LP is exact.

[2] The excellent introductory book "The design of approximation algorithms", by Shmoys and Williamson [78] is filled for a large part with such methods.

In Chapter 2 we use the (randomized) iterative rounding paradigm, characterized by rounding steps interspersed with shrinking of the instance and/or re-solving the relaxation. This method has fueled many of the breakthroughs in approximation algorithms for network design since the early 2000s[3], and the current title holder for the best approximation for Steiner tree problem [17],

[3] See Lau, Ravi and Singh "Iterative methods in combinatorial optimization" [60] for an introduction to the topic. Apart from explaining some of the important approximation algorithms, such as Jain's landmark algorithm for the generalized Steiner network problem [58], it show how iterative rounding can be used to reproduce known exactness results and other facts about LP relaxations in completely different way.

a fundamental problem in combinatorial optimization, uses this technique.

A completely different type of rounding, is rounding the parameters of the problem in some way to reduce the number of unique values and/or make them well separated, for example by rounding to a geometric series. This type of rounding is very common in the design of polynomial time approximation schemes.[4] We use rounding of the input heavily to derive approximation algorithms in Chapters 3 and 4. In the latter chapter, we use also a more exotic type of rounding to derive a QPTAS. Here we really round the internal variables in the execution of the algorithm, thus reducing the state space and speeding up computation.

[4] Rounding of this type is especially useful in scheduling, where approximation schemes are ubiquitous. The idea of distinguishing rounding the input, the execution of the algorithm, or the output, is taken from Schuurman and Woeginger in "Approximation Schemes – A Tutorial" [72].

Reformulate captures modifications to the problem, other than relaxing it. One way is dualizing parts of the problem. We use LP duality between fractional matching and fractional vertex cover in Chapter 1 to derive a completely different problem which we then attack directly. In Chapters 2, we use a path based formulation for a problem where we fundamentally want tours. This formulation turns out to be much easier to work with and allows us to pull off the (otherwise tricky) analysis of the randomized iterative rounding procedure we use.

Another way is the opposite of relaxing: adding constraints to the problem, under which it is easier to find the optimal solution. In Chapter 4 we impose strong constraints on the structure of a solution. This is not without loss of generality, but loses only a constant in the approximation factor while making the problem a lot simpler. After rounding the instance and relaxing the objective function by dropping some terms, we end up with a problem that can be exactly solved using randomized rounding of an LP relaxation (as mentioned above). Hence, Chapter 4 provides a nice showcase for all the techniques we just discussed.

We will now review the chapters in greater detail.

Chapter 1

A virtual private network (VPN) provider offers a service to clients that allows digital devices in multiple locations — servers, laptops, smartphones etc. — to *virtually* connect as if they were part of the same local network. To guarantee quality of service it needs to install connections with sufficient bandwidth to carry data between all devices.

A crude solution would be to reserve some bandwidth for every connection a client might potentially make, but this does not take into account some basic restrictions. Every device has a maximum download and upload rate (based on hardware limitations, or their internet subscription for example) and this means that if a device sends a lot of data to A, it cannot send as much data to B.

By making clever use of such knowledge, capacity can be reused for multiple connections. This reduces the amount of bandwidth required (which is expensive) and therefore the cost of the network. Chapter 1 studies algorithms for calculating how to do this most effectively.

Technical outline of Chapter 1 In particular, we study algorithms for the masked VPN problem. This problem deals with routing uncertain and varying traffic patterns in a network. We are given a set of terminals W in a network G, together with a second graph H on the terminals (the *mask* graph) that encodes which terminal pairs might communicate with each other. Every terminal has a unit capacity connection with the network it may use to communicate with other terminals at any time. For every communicating terminal pair (according to H), we need to pick a connecting path through the network G. Subsequently we must install sufficient capacity in the network such that every traffic pattern that is feasible under these constraints can be routed via the predetermined paths. The masked VPN problem asks for a minimum cost solution to this problem.

The masked VPN problem generalizes the VPN problem, which can be seen as a special case where every terminal pair communicates, i.e. H is a clique. This problem has an elegant polynomial time solution that looks like a star embedded into the network. We explore generalizations of this result to other classes of mask graph H.

Most of the chapter is dedicated to a proof of a structure theorem for the case where H is a cycle. This states that there is always an optimal solution that looks like a cycle with spokes embedded in the network (Theorem 1.3.1), see Figure 1 for an illustration.

As the optimal structure can be found efficiently by dynamic programming, the cycle case is solvable in polynomial time (Theorem 1.1.1). We also show that when H has bounded degree, the optimal topology can be found in polynomial time using dynamic programming (Theorem 1.1.2). The bounded degree is necessary, as the special case of H a star is APX-hard — it corresponds exactly to Steiner tree.

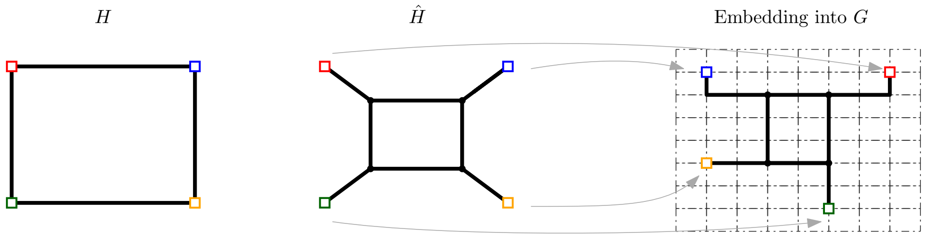


Figure 1: The embedding algorithm for H a cycle; in this example, G is a grid. Every path between pairs of communicating terminals follows the shortest route on the embedded cycle.

The technique that yield the structure theorems for both cases starts out with a reformulation of the problem. A straightforward formulation uses the paths connecting terminals as decision variables, which completely determines the necessary capacity in G. The required capacity of an edge e is equal to a maximum fractional matching in a restricted subgraph of H containing only edges between terminals that use e on their connecting path.

By dualizing the capacity computation — yielding a fractional vertex cover problem — we find a new viewpoint of the masked VPN problem. Every terminal fractionally pays for a set of edges in G, such that each pair of terminals that need to be connected, together fully pay for a connecting path. Finally, we change the problem slightly by requiring every terminal to buy an integral set of edges.

We then apply uncrossing techniques to show this problem has a structurally simple solution (like in Figure 1). Once we have this result, it is not much work to show that the integrality requirement can be added without loss of generality.

Chapters 2 and 3

In inventory management — think stocking supermarket warehouses with enough soup cans to serve demand — there is an interesting dynamic between the two quantities that largely drive expenses: square meters of storage space, and litres of fuel.

Ideally, inventory would arrive just as it is about to sell out, going from truck to shelf directly. But this would require deliveries to be made constantly. From a transportation point of view sending everything at once and far ahead of time would be more sensible, but this raises the required storage space, an expensive proposition in today's urban areas with their ever rising property prices. Chapters 2 and 3 contain algorithms to balance these forces and generate cheaper restocking schedules.

The former chapter studies a more general model, providing high level procedures usable in a wider array of inventory management problems and other settings. The latter chapter considers a specific scenario, where inventory needs to be replenished within a predictable number of days after each delivery, motivated by an application in cash replenishment at ATM's.

Technical outline of Chapter 2 In Chapter 2 we look at subadditive inventory problems, which contain the well-studied inventory routing problem (IRP) and the submodular joint replenishment problem (SJRP). In the general setting, N nodes need to be served over some finite time horizon $1, ..., T$. Multiple nodes may be served simultaneously, at a cost given by a subadditive ordering cost function $f : 2^N \to \mathbb{R}$. Demand can be served exactly on time, or at any point prior to that, in which case a holding cost needs to be paid for the intervening period. The goal is to balance these cost factors so as to minimize the total cost of the schedule.

The IRP is the special case where nodes are situated in a metric and ordering cost is given by the minimum cost route visiting all nodes from the depot. The submodular joint replenishment problem is the case where the ordering cost function is submodular. We provide the first sublogarithmic approximation algorithm for these two problems, both attaining an approximation ratio of $O(\log\log\min(N, T))$ (Theorems 2.1.1 and 2.1.2).

This improves the previous best known $O(\log N)$, $O(\log T)$ approximation ratios (using two different algorithms), as well as the $O(\log T/\log\log T)$ approximation algorithm for the special case of polynomial holding cost. We also provide general purpose results for the subadditive inventory problem, bounding the length of the time horizon in terms of the number of nodes N (Theorem 2.2.14), unifying approximation ratios for both parameters in a single algorithm.

The approximation algorithm for the IRP uses randomized iterative rounding of a relaxation we call a *fractional path solution*. This solution consists for each day of a tree, and a weighted collection of paths fractionally connecting the nodes to the trees. In each step we sample these paths, connecting more nodes to the trees. Next, we show we can delete a large fraction of edges from the paths, reducing the cost of the relaxation by a constant factor.

The approximation algorithm for the SJRP uses (deterministic) iterative rounding of a relaxation based on the Lovász extension. In each iteration, and for each day, we either find a set of items to buy that can be paid for by an immediate reduction in the cost of the relaxation, or we conclude that we can buy a small set of items whose cost can be charged against the cost of the relaxation a logarithmic number of times without losing too much. We then collapse adjacent time units and repeat the process.

A key element in both these algorithms is a reduction to a problem where each node needs to be visited in a certain time window (the holding costs are eliminated). Though this approach is used in previous work as well, we significantly increase its power by introducing the concept of *left aligned* families. Intuitively, this is a family of intervals that enjoys many of the nice properties of laminar families. However, we show that they are general enough that we can assume all time windows lay in a left aligned family at the loss of constant factor.

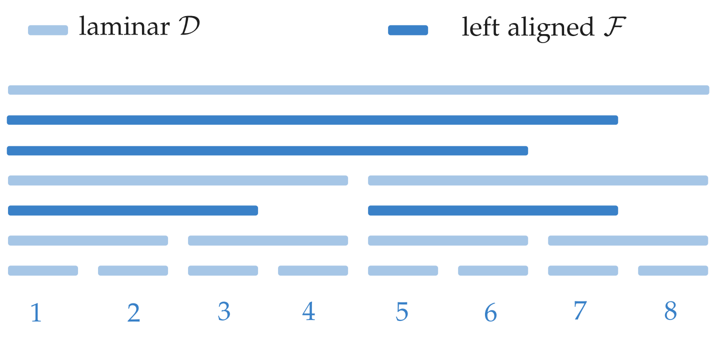


Figure 2: Example of laminar and left aligned families of intervals on $\{1, \ldots, 8\}$.

Both algorithms were initially conceived with laminar time windows in mind, which makes them significantly simpler. The techniques were then generalized using the left aligned family paradigm, working out the technical complications with the benefit of intuition provided by the result for the laminar case. We suspect that this approach may prove useful in designing approximation algorithms for other subadditive inventory problems.

Technical outline of Chapter 3 In the replenishment problem with fixed turnover times (RFTT), we have a metric with clients that need to be replenished (repeatedly) over an infinite time horizon. After each replenishment, a node must be revisited within a fixed time period (call the turnover time). Consequently, the effect of visiting a client early propagates, i.e. the next visit will also need to happen earlier. This property differentiates the RFTT from models typically studied in the literature. The study of the RFTT is motivated by applications in practice for which the fixed turnover time is more natural, for example the replenishment of ATMs. After every replenishment an ATM has a full stock of cash, which starts depleting at a steady rate immediately. The ATM will need to be revisited before it runs out again.

We consider the objectives of minimizing the average distance per day as well as the maximum distance per day. We provide constant factor approximation for both objectives when distance is given by a tree metric (Theorems 3.3.4 and 3.3.15), and logarithmic approximations for the general case under maximum distance objective (Theorems 3.4.1, 3.4.2) and a sublogarithmic approximation algorithm for the average distance objective (Theorem 3.4.5).

An important first step in deriving these results is rounding the instance such that all turnover times are powers of 2. This simplifies the problem enough that the min sum objective becomes polynomial time solvable on trees. The max approximation on general metrics relies on a decomposition technique, using properties of the objective to show we can layer the nodes into a logarithmic number of 'easy' instances. An approximation for the average distance objective is then derived by reducing the problem to the inventory routing problem, for which an $O(\log \log n)$-approximation is given in Chapter 2.

The most interesting result is the constant factor approximation for the min max objective on trees, however. We show how we can iteratively split a TSP tour and distribute it among the days, in each step fixing the nodes with a progressively higher turnover time. If we do this carefully, we can balance the workload assigned to each sequence of days. We complete the result by showing that on any given day there exists a cheap path connecting all the nodes assigned to it by the algorithm.

A technical challenge underlying all these problems is that they contain the so-called pinwheel problem as a special case (Theorem 3.2.3). This is essentially the replenishment setting where one tries to visit at most one client a day while staying feasible (though the motivation comes from a different point of view). It is not known whether this problem is in NP, nor if it is NP-hard, though it is conjectured to be PSPACE-complete.

Chapter 4

Imagine a small workshop with multiple production lines, that just received a batch of job orders. The orders have different priorities, and ideally the high priority orders get completed first. However, job orders arrive sequentially at the workshop and there is no time/space to reorder them before sending them to the machines, causing low priority orders that arrive early to delay high priority orders that come in later.

We can assign the jobs to the different production lines as we see fit, though. Is there a good way to exploit this? A reasonable sounding approach is to reserve some production lines for high priority jobs, trading off idle time for more flexibility. Our analysis shows that this is indeed a good idea and distills the intuition in an algorithm for computing efficient production schedules in Chapter 4.

Technical outline of Chapter 4 The optimization problem we look at is fixed order scheduling on parallel machines. In this setting, jobs with processing times and weights need to be processed on identical machines so as to minimize the total weighted completion time. The caveat is that the jobs arrive with a predetermined order, and this order must be respected on every machine (but not across machines). This prevents us from processing the jobs according to Smith's ratio on the individual machines, as would have been optimal otherwise.

We derive a constant factor approximation for this problem (Theorem 4.4.1), and QPTAS for the special case of unit processing times (Theorem 4.6.2).

These results build upon a strong structural characterization of (near) optimal solutions, developed in the first part of the chapter. A powerful concept here is a natural partial order we define on the jobs, found as the intersection of the input ordering and the order of the Smith ratios. We show that there is always a schedule whose machine assignment is monotone with respect to this partial order that is optimal (for unit processing times, Lemma 4.3.3) or within a factor $\frac{3}{2}$ of optimal (for general processing times, Lemma 4.3.6). We call such schedules *Smith-monotone*.

Restricting the solution to Smith-monotone schedules drastically reduces its degrees of freedom. Not only does this make reasoning about the problem much simpler, we also show that by rounding the instance and dropping some judiciously chosen terms from the objective function (at the loss of a constant factor, Theorem 4.4.3), the problem becomes polynomial time solvable. Additionally we provide some evidence that techniques previously successful on other scheduling problems with completion time objective are bound to fail on this problem.

1
The masked VPN problem

The contents of this chapter appeared in:
[15] Thomas Bosman and Neil Olver. "Exploring the Tractability of the Capped Hose Model".
Proceedings of the 25th Annual European Symposium on Algorithms (ESA), 2017.

Abstract Robust network design concerns the design of networks to support uncertain or varying traffic patterns. An especially important case is the *VPN problem*, where the total traffic emanating from any node is bounded, but there are no further constraints on the traffic pattern. Recently, Fréchette et al. [39] studied a generalization of the VPN problem where in addition to these so-called hose constraints, there are individual upper bounds on the demands between pairs of nodes. They motivate their model, give some theoretical results, and propose a heuristic algorithm that performs well on real-world instances.

Our theoretical understanding of this model is limited; it is APX-hard in general, but tractable when either the hose constraints or the individual demand bounds are redundant. In this work, we uncover further tractable cases of this model; our main result concerns the case where each terminal needs to communicate only with two others. Our algorithms all involve *optimally embedding* a certain auxiliary graph into the network, and have a connection to a heuristic suggested by Fréchette et al. for the capped hose model in general.

1.1 Introduction

Robust network design (RND) [11] is concerned with designing networks that can efficiently handle uncertain or varying utilization. The motivation comes primarily (though not exclusively) from communication networks. Let $G = (V, E)$, be a graph with edge costs that describes an existing, high-capacity network. A set of *terminals* $W \subseteq V$ is required to communicate over the network, and to enable this, we must reserve capacity on the edges of G for our exclusive use (this is in order to guarantee reliable performance). On each edge, we may buy multiple units of capacity (measured, say, in Mb/s); the cost of the edge represents the per-unit cost of capacity. In the RND framework, demand uncertainty is described by a *demand universe* $\mathcal{U}$. It is simply a set containing all of the demands that need to be routed; the choice of this set will be determined by operational needs or historical data. More precisely, each $D \in \mathcal{U}$ is a matrix where entry D_{ij} describes the demand (measured again, say, in Mb/s) from terminal i to terminal j. It turns out that the universe can always be taken to be a convex set, and will frequently be a polytope.

We will consider only single-path, oblivious routing (other routing schemes are possible, but less relevant to practice). This means that a solution to the RND problem must specify, for each pair of terminals $i, j \in W$, a path P_{ij} that will be used to route the demand between this pair. This path must be fixed ahead of time, and cannot be adjusted as a function of the current demand. Once all these paths have been fixed, a capacity reservation must be made on the network. For any edge $e \in E$, the capacity $u(e)$ must be chosen so that no matter which demand matrix $D \in \mathcal{U}$ is instantiated, the total amount of traffic traversing e does not exceed $u(e)$.

The most studied case of universe is the *hose model* [29, 36]. Here, each terminal $i \in W$ has an associated marginal b_i, and the universe $\mathcal{H}(\boldsymbol{b})$ consists of all demand matrices D for which $\sum_{j \in W} D_{ij} \le b_i$ for all i, and $D_{ij} = D_{ji}$[1]. The optimization problem for the hose model is called the VPN problem, and it was shown by Goyal et al. [46] to be polynomially solvable.

[1] This is the symmetric hose model; an asymmetric variant which does not require $D_{ij} = D_{ji}$ is also possible, and different [50],[30],[49].

While the hose model is particularly appealing, especially given its exact solvability, it is very natural to consider generalizations with more expressive modelling power. A number of such generalizations have been considered in the literature [67, 39, 37, 32]. One such generalization, the *capped hose model* was introduced by Fréchette, Shepherd, Thottan and Winzer [39]. It is very natural: in addition to the hose constraints b_i for each $i \in W$, there is an upper bound d_{ij} on the demand between a given pair $i, j \in W$. This leads to the capped hose polytope $\mathcal{H}^{\text{cap}}(\boldsymbol{b}, \boldsymbol{d})$. If $d_{ij} = \infty$ for all pairs $i, j \in W$, then this recovers the hose model; and if $b_i = \infty$ for all $i \in W$, then this recovers the *pipe model*, the somewhat trivial case where the problem is to route a single fixed demand matrix. We refer to Fréchette et al. [39] for further discussion and motivation.

As Fréchette et al. [39] observed, the problem of finding the cheapest solution in the capped hose model generalizes Steiner tree, and hence is APX-hard. Simply consider, for an arbitrarily chosen root $r \in W$, the choice $b_i = 1$ for all $i \in W$, and $d_{ir} = d_{ri} = 1$ for all $i \in W$, $d_{ij} = 0$ otherwise. Beyond this, the complexity and approximability of this problem is poorly understood. In particular, it is open as to whether there is a constant factor approximation algorithm. (The general robust network design problem is hard to approximate within polylog factors [67], but this construction does not apply to this restricted setting.) Moreover, the RND problem under $\mathcal{H}^{\text{cap}}(\boldsymbol{b}, \boldsymbol{d})$ is polynomially solvable for some choices of $\boldsymbol{b}$ and $\boldsymbol{d}$, for example when $\boldsymbol{d}$ is sufficiently large (recovering the hose model), or $\boldsymbol{b}$ is sufficiently large (recovering the pipe model). Our goal in this work is to expand the class of exactly solvable cases.

We focus on the setting where $b_i = 1$ for all $i \in W$ and $d_{ij} \in \{0, \infty\}$ (or equivalently, $d_{ij} \in \{0, 1\}$) for all $i, j \in W$, which we call the *masked hose model*. Instead of parametrizing an instance with $\boldsymbol{d}$ and $\boldsymbol{b}$, we can describe it via the *mask graph* H, which has vertex set W and an edge between each pair of terminals which may communicate. In other words, the universe is the set of all fractional matchings in H. The resulting *masked VPN problem* is a clean generalization of the standard VPN problem (the case where H is the complete graph), and is already very rich from a theoretical standpoint. Again, it is not known if a constant approximation factor is possible for arbitrary mask graphs, and the case where H is a star is APX-hard. It is harmless to restrict to connected mask graphs, since otherwise the problem can be solved separately on each connected component, and the resulting solutions overlaid in G.

Our main result is the following

Theorem 1.1.1. *The masked VPN problem is polynomially solvable if H is a cycle.*

The algorithm is based on embedding an appropriate auxiliary graph (see Figure 1.1). Let $\hat{H}$ denote the graph obtained by replacing each terminal i by a new node $\hat{\imath}$, and then adding back the terminal i along with the edge $\{i, \hat{\imath}\}$. We give each edge e of $\hat{H}$ a capacity of 1. An *embedding* of $\hat{H}$ into G is simply a mapping ϕ satisfying the following. Each node of $\hat{H}$ is mapped to a node of G, with $\phi(i) = i$ for all $i \in W$; and each edge $\{u, v\} \in E(\hat{H})$ maps to a path in G between $\phi(u)$ and $\phi(v)$. Any embedding implies a path in G between any adjacent pair of adjacent terminals $\{i, j\} \in E(H)$; simply the image under the embedding of the path $(i, \hat{\imath}, \hat{\jmath}, j)$. After assigning a capacity reservation $u(e) = |\{f \in E(\hat{H}) : e \in \phi(f)\}|$, it is easy to see that this yields a feasible solution to the masked VPN problem for H. Our algorithm simply finds the cheapest possible embedding of $\hat{H}$ into G; this can easily be done by dynamic programming. We show that this is optimal; an overview of the proof strategy can be found in Section 1.3.1

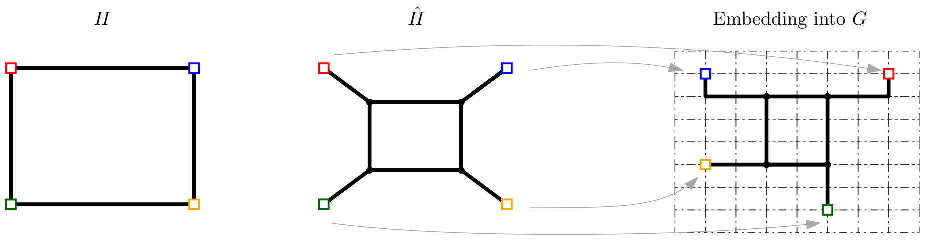


Figure 1.1: The embedding algorithm for H a cycle; in this example, G is a grid.

The cycle may seem like a very specific and restricted case. But understanding cycles has historically been an important stepping stone in the area towards more general results. The VPN Conjecture on the polynomial solvability of the hose model was first solved for the case where the network is a cycle [55, 48], and ideas from [48] were crucial for the resolution of the full conjecture.

We also prove the following.

Theorem 1.1.2. *The masked VPN problem is exactly solvable if H is a tree with bounded degree.*

Technically, this result is much more straightforward, and we give the proof in Section 1.4. It exploits the well-known Dreyfus-Wagner algorithm for Steiner tree on a fixed number of terminals [28], which corresponds to the case where H is a bounded degree star. We make heavy use of the dual viewpoint, discussed in Section 1.2, in order to argue that the solution can be efficiently decomposed into Steiner tree problems. And while our focus is on exactly solvable cases, we remark that an $O(1)$-approximation without any degree bound can readily be obtained (see Theorem 1.4.2 in Section 1.4).

While this result does not require major technical novelty, it yields an interesting message. The algorithm can *also* be interpreted as an embedding algorithm. This time, however, there are multiple options regarding which graph to embed, and we have to choose the best. Begin by constructing $\hat{H}$ in the same fashion as above, splitting out each terminal. But now we go further; for each node $v \in V(\hat{H})$ with degree 4 or more, consider all possible ways of "blowing up" v into a tree with only degree 3 nodes (see Figure 1.2). Each possible way of blowing up each node v yields a graph whose embedding yields a solution. The algorithm computes the cheapest possible embedding from all of these possibilities; by using dynamic programming, combined with the assumption of bounded degree, this can be done in polynomial time. Again, this is precisely the idea of the Dreyfus-Wagner algorithm for Steiner tree [28], extended from H a star to H a tree. We also observe that if H was a path, then $\hat{H}$ has maximum degree 3, and so only $\hat{H}$ itself needs to be embedded.

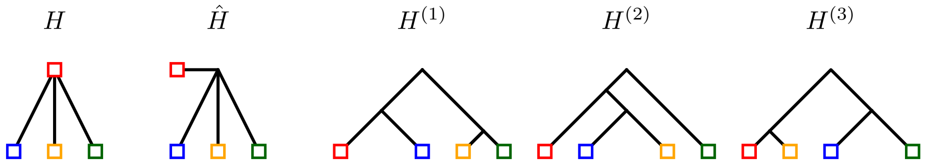


Figure 1.2: Potential graphs to embed in the case where H is a star.

Further, embedding algorithms of this form have been used before in RND, though only embeddings of *trees*. For the standard VPN problem, the optimal solution is simply the optimal embedding of a star with a leaf for each terminal [46]. This is very natural when one considers that the demand universe for the hose model—fractional matchings on the complete graph—is nothing more than the set of demands that are routable on such a star. A generalization of the hose model (different to the one discussed here) defines the universe to be the set of demands routable on a given capacitated tree (with leaf set equal to W) [67]. It has been conjectured that the optimal algorithm is given by the optimal embedding of this tree [66]; it is only known that this yields a constant factor approximation [67].

Partially motivated by this, Fréchette et al. proposed a tree embedding algorithm as a heuristic for the capped hose model. The tree they embed is chosen carefully, albeit heuristically, and they show that this performs well in practice. Our work can be seen as providing an initial theoretical basis for their approach, while suggesting that extending beyond trees may be beneficial.

1.2 *Problem definition and preliminaries*

An instance of the masked VPN (MVPN) problem consists of a graph $G = (V, E)$, with edge costs $c : E \to \mathbb{R}_+$ (where $\mathbb{R}_+$ denotes the nonnegative reals), a set of terminals $W \subseteq V$, and a second graph H which has W as its vertex set.

We use $\binom{W}{2}$ to denote the collection of unordered pairs of distinct terminals. The demand universe is defined as

$$\mathcal{H}^{\text{mask}}(H) := \left\{ D \in \mathbb{R}_+^{\binom{W}{2}} : \sum_j D_{ij} \leq 1 \; \forall i \text{ and } D_{ij} = 0 \text{ unless } \{i,j\} \in E(H) \right\}.$$

Our goal is to specify a routing template $\mathcal{P} = \{P_{ij} \; : \{i,j\} \in E(H)\}$, where P_{ij} is a fixed (possibly non-simple) i-j-path used for traffic between terminal i and j. (P_{ij} and P_{ji} refer to the same path.) Given a set of routing paths, we are required to make a capacity reservation $u : E \to \mathbb{R}$, sufficient to route any traffic vector in $\mathcal{H}^{\text{mask}}(H)$. The minimum capacity requirement on edge e is therefore

$$u(e) = \max_{D \in \mathcal{H}^{\text{mask}}(H)} \sum_{\{i,j\} \in \binom{W}{2} : e \in P_{ij}} D_{ij}. \tag{1.1}$$

The resulting solution has cost $\sum_e c(e)u(e)$, and this we wish to minimize.

We will often take a dual viewpoint of (1.1). This viewpoint has been exploited before, see [4, 55, 46]. Since (1.1) is a fractional matching problem, its dual is a fractional vertex cover problem:

$$\begin{aligned} u(e) = \quad \min \quad & \sum_{i \in W} y_i(e) \\ \text{s.t.} \quad & y_i(e) + y_j(e) \geq 1 \qquad \forall \{i,j\} \in E(H),\ e \in P_{ij} \\ & y_i(e) \geq 0. \end{aligned} \tag{1.2}$$

So, we may rephrase the problem as follows. Each terminal i buys a capacity vector y_i, with the property that $\{e \in E : y_i(e) + y_j(e) \geq 1\}$ contains an i-j-path, for each $\{i,j\} \in E(H)$. The goal is to minimize the total cost $\sum_i c(y_i)$, where $c(y_i) = \sum_{e \in E} c(e) y_i(e)$.

Let $\boldsymbol{y}$ denote the vector whose i'th component y_i is the capacity vector purchased by $i \in W$. At this point we note that, since (1.2) is a fractional vertex cover problem, $\boldsymbol{y}$ can always assumed to be half-integral. In fact, we will mainly be able to restrict ourselves to integral capacity vectors. In such a case it is convenient to express the solution as a collection of edge sets $\boldsymbol{Y} = (Y_i)_{i \in W}$ where $Y_i = \{e : y_i(e) = 1\}$.

Remark. Through this dual viewpoint, a connection can be made with the work of Iglesias et al. [56]. With a completely different motivation, they consider essentially this problem, explicitly requiring integrality but also *connectivity* of the set of edges purchase by each terminal. Since we show that the optimal solutions satisfy these properties, our results apply in their setting as well.

1.3 *The cycle case*

We consider the case where H is a cycle. Let k denote the number of terminals, and assume for convenience that $W = \{1, 2, \ldots, k\}$, with the ordering corresponding to the cycle structure of H. We will interpret all references to terminals modulo k; so terminal 0 refers to terminal k, and terminal $k+1$ to terminal 1.

Our main technical theorem shows that there is always an optimal solution to the mVPN problem that satisfies a simple structure.

Theorem 1.3.1 (Hubbed solution)**.** *There exists an optimal solution to the* mVPN *problem on cycles such that:*

- *for each terminal i there exists a* hub *vertex h_i; and*
- *the routing path $P_{i,i+1}$ is given by concatenating shortest paths from i to h_i, from h_i to h_{i+1}, and from h_{i+1} to $i+1$.*

The optimal location of the hub vertices minimizes the cost

$$\sum_{i \in W} \mathrm{sp}_c(i, h_i) + \mathrm{sp}_c(h_i, h_{i+1}),$$

where sp_c denotes the shortest path distance with respect to the edge costs c. Since H is a cycle the optimal location of hubs $h_{i+1}, \ldots, h_{j-1}$ are independent of hubs $h_{j+1}, \ldots, h_{i-1}$ given the location of h_i and h_j, so we can find these hubs in polynomial time with dynamic programming, yielding Theorem 1.1.1.

1.3.1 *Overview*

The proof of Theorem 1.3.1 involves first showing that there is always an optimal solution of a certain nice form, albeit not yet of the hubbed form we are looking for. We first argue that we may restrict our focus to solutions $\boldsymbol{y}$ that are integral. Next, we show that there is an integral solution satisfying a certain structure theorem. Roughly speaking, this structure is similar to the hubbed structure we are looking for, but instead of a single hub h_i, there is an odd-cardinality set T_i; instead of a path between i and h_i, we have a $(\{i\} \triangle T_i)$-join; and instead of a path between h_i and h_{i+1}, we have a $(T_i \triangle T_{i+1})$-join. The final step of the argument is then to show that we can take $|T_i| = 1$ for each i, which is then precisely a hubbed solution. This step uses a rather non-obvious "rotation" of the solution to reduce the cardinality of the T_i's.

1.3.2 *Integrality*

It will be convenient to work with integral solutions. If k is even, so that H is bipartite, then each fractional matching problem in (1.2) has an integral optimum. We have to work a bit harder in the case where k is odd.

Lemma 1.3.2. *There exists an integral optimal solution to the cycle* MVPN *problem.*

Proof. Since any strict subgraph of a cycle is bipartite, the only case where (1.2) does not have an integral optimal solution, is if it corresponds to a vertex cover problem on the complete cycle. This only happens if the routing path between *every* pair of neighbours uses the edge. So suppose $e = \{u, v\}$ is used on every routing path in a solution $\boldsymbol{y}$. We claim the integral solution $\boldsymbol{Z}$, given by taking Z_i to be the edges of a shortest i-v-path for all terminals i, costs no more than $\boldsymbol{y}$.

Let D be a traffic vector with $D_{i,i+1} = \frac{1}{2}$ for all i. Now if we route D according to the solution $\boldsymbol{y}$, the flow between i and $i+1$ can be split into half a unit of i-v flow and half a unit of v-$(i+1)$-flow, since every routing path passes through e, and thus v.

So D induces a unit i-v flow for each $i \in W$. So $\boldsymbol{y}$ has sufficient capacity to route 1 unit of flow from each $i \in W$ simultaneously. But this costs at least as much as the sum of the shortest paths from each terminal to v, as required. □

For the remainder of the proof we will therefore assume that each terminal buys a set of edges. It will be useful to partition these edges into a different collection of sets based on the routing paths they support.

Definition 1.3.3. A *feasible solution* $\bar{\boldsymbol{X}}$ consists of edge sets $\bar{X}_i$ and $\bar{X}_{i,i+1}$ for each i, such that all edge sets are disjoint and for each i there exists a path $P_{i,i+1}$ connecting terminal i to $i+1$ with

$$P_{i,i+1} \subseteq \bar{X}_i \cup \bar{X}_{i,i+1} \cup \bar{X}_{i+1}.$$

The cost of the solution is $\sum_i c(\bar{X}_i) + \sum_i c(\bar{X}_{i,i+1})$.

One should think of $\bar{X}_i$ as the edges on both $P_{i,i+1}$ and $P_{i-1,i}$ and $\bar{X}_{i,i+1}$ as the remaining edges on $P_{i,i+1}$. Such a solution can be transformed into a feasible solution in original form by setting $X_i = \bar{X}_i \cup \bar{X}_{i,i+1}$ for all i. Indeed, the definition does not confer any advantage to choosing any of the sets $\bar{X}_{i,i+1}$ to be nonempty. But as we will see in the next section, this formulation provides a natural way to express the structure of a feasible solution.

1.3.3 *Structure theorem*

For the remainder we will assume that G is a complete graph satisfying the triangle inequality. We may simply replace G with its metric completion to ensure this, and it allows us to bypass some substantial technical awkwardness.

Definition 1.3.4. Let $\boldsymbol{T}$ be a collection consisting of an odd cardinality sets $T_i \subseteq V \setminus W$ for each terminal i. Then a $\boldsymbol{T}$*-solution* $\bar{\boldsymbol{X}}$ consists of a collection of edge sets $\bar{X}_i$ and $\bar{X}_{i,i+1}$ for each terminal i satisfying:

1. $\bar{X}_i$ is a perfect matching on $T_i \triangle \{i\}$

2. $\bar{X}_{i,i+1}$ is a perfect matching on $T_i \triangle T_{i+1}$

The cost of the solution is $\sum_i c(\bar{X}_i) + \sum_i c(\bar{X}_{i,i+1})$.

It is good to observe that Property 1 and 2 of this definition imply that $\bar{\boldsymbol{X}}$ is a feasible solution. The odd degree vertices of $\bar{X}_i \,\dot{\cup}\, \bar{X}_{i,i+1} \,\dot{\cup}\, \bar{X}_{i+1}$ are precisely

$$T_i \triangle \{i\} \triangle T_i \triangle T_{i+1} \triangle T_{i+1} \triangle \{i+1\} = \{i, i+1\},$$

implying that i and $i+1$ are in the same component.

The restriction in the above definition that $T_i \cap W = \emptyset$ is without loss of generality, since we may always modify any given instance by replacing each terminal i in the instance and the solution with a new dummy node $\bar{\imath}$, and then replacing i in the instance at the same location, attaching the terminal to this dummy node by an edge of zero cost, and including $\{i, \bar{\imath}\}$ in $\bar{X}_i$.

The following lemma shows that we do not lose anything if we restrict ourselves to $\boldsymbol{T}$-solutions.

Lemma 1.3.5 (Weak Structure Lemma). *Any feasible solution $\bar{\boldsymbol{X}}$ may be transformed into a $\boldsymbol{T}$-solution $\bar{\boldsymbol{Y}}$ of no higher cost for some $\boldsymbol{T}$ satisfying $T_i \subseteq V(\bar{X}_i) \setminus \{i\}$ for all $i \in W$.*

Proof. Define:

- $\bar{Y}'_i = \bar{X}_i \cap E(P_{i-1,i}) \cap E(P_{i,i+1})$, and

- $\bar{Y}'_{i,i+1} = E(P_{i,i+1}) \setminus (\bar{X}_i \cup \bar{X}_{i+1})$.

We then obtain $\bar{\boldsymbol{Y}}$ from $\bar{\boldsymbol{Y}}'$ by shortcutting paths, so that $\bar{Y}_i$ is a collection of vertex disjoint edges for each $i \in W$.

Now for each terminal i choose the vertex set T_i such that $T_i \triangle \{i\}$ is the set of vertices incident to an edge in $\bar{Y}_i$. We claim that $\bar{\boldsymbol{Y}}$ is a $\boldsymbol{T}$-solution.

By construction, $\bar{Y}_i$ is a perfect matching on $T_i \triangle \{i\}$. To see that $\bar{Y}_{i,i+1}$ is a perfect matching on $T_i \triangle T_{i+1}$, note that since $\bar{Y}'_i$ and $\bar{Y}'_{i+1}$ are both contained in the path $P_{i,i+1}$, we may write

$$\bar{Y}'_{i,i+1} = E(P_{i,i+1}) \triangle \bar{Y}_i \triangle \bar{Y}_{i+1}.$$

Thus the odd degree nodes of $\bar{Y}'_{i,i+1}$ are precisely

$$\{i, i+1\} \triangle T_i \triangle \{i\} \triangle T_{i+1} \triangle \{i+1\} = T_i \triangle T_{i+1}.$$

As the odd degree nodes in $\bar{Y}'_{i,i+1}$ and $\bar{Y}_{i,i+1}$ are equal, the result follows. □

Definition 1.3.6. A *strong* $\boldsymbol{T}$*-solution* is a $\boldsymbol{T}$-solution with the additional properties:

(i) $\bar{X}_i \dot\cup \bar{X}_{i,i+1} \dot\cup \bar{X}_{i+1}$ consists of a single i-$(i+1)$-path, and

(ii) each edge in $\bar{X}_{i,i+1}$ is incident to one vertex in T_i and one in T_{i+1}.

Notice that in a strong $\boldsymbol{T}$-solution $\bar{\boldsymbol{X}}$, $|T_i| = |T_j|$ for all $i, j \in W$.

Lemma 1.3.7 (Strong Structure Lemma)**.** *Any* $\boldsymbol{T}$*-solution* $\bar{\boldsymbol{X}}$ *can be transformed into a strong* $\boldsymbol{R}$*-solution* $\bar{\boldsymbol{Y}}$ *of no higher cost, with* $R_i \subseteq T_i$ *for all* $i \in W$.

For the proof of this lemma we will need two auxilliary lemmas.

Lemma 1.3.8. *Let* $\bar{\boldsymbol{X}}$ *be a* $\boldsymbol{T}$*-solution such that for some* $i \in W$, $\bar{X}_i \dot\cup \bar{X}_{i,i+1} \dot\cup \bar{X}_{i+1}$ *does not satisfy property (i) of Definition 1.3.6. Then there exists an* $\boldsymbol{R}$*-solution* $\bar{\boldsymbol{Y}}$ *of no higher cost, with* $R_j \subseteq T_j$ *for all* $j \in W$*, and* $R_i \subsetneq T_i$.

Proof. Since i and $i+1$ are the only vertices that do not have degree 2 in $\bar{X}_i \dot\cup \bar{X}_{i,i+1} \dot\cup \bar{X}_{i+1}$, the connected component containing i and $i+1$ contains an i-$(i+1)$-path; call it P.

Define $R_i = T_i \cap V(P)$, $R_{i+1} = T_{i+1} \cap V(P)$, and $R_j = T_j$ for all other $j \in W$. Note that $R_i \subsetneq T_i$. We will now construct a $\boldsymbol{R}$-solution $\bar{\boldsymbol{Y}}$ as follows. Define $\bar{Y}_j = \bar{X}_j$ for all $j \notin \{i, i+1\}$, and $\bar{Y}_{j,j+1} = \bar{X}_{j,j+1}$ for all $j \notin \{i-1, i, i+1\}$. Now define $\bar{Y}_{i,i+1} = \bar{X}_{i,i+1} \cap P$, so it is a perfect matching on $R_i \triangle R_{i+1}$ with $c(\bar{Y}_{i,i+1}) \leq c(\bar{X}_{i,i+1})$. Also let $\bar{Y}_i = \bar{X}_i \cap P$, which is a perfect matching on $R_i \triangle \{i\}$. We have $c(\bar{Y}_i) = c(\bar{X}_i) - c(Q)$, where $Q = \bar{X}_i \setminus P$ is a perfect matching on $T_i \setminus R_i$. To define $\bar{Y}_{i-1,i}$, first let

$$\bar{Y}'_{i-1,i} = \bar{X}_{i-1,i} \triangle (\bar{X}_i \setminus P).$$

Notice that the odd degree nodes of $\bar{Y}'_{i-1,i}$ are precisely

$$(T_{i-1} \triangle T_i) \triangle (T_i \setminus R_i) = T_{i-1} \triangle R_i = R_{i-1} \triangle R_i.$$

Now, by discarding any cycles and shortcutting paths, we can choose $\bar{Y}_{i-1,i}$ to be a perfect matching on $R_{i-1} \triangle R_i$ that costs no more than $\bar{Y}'_{i-1,i}$. So we have $c(\bar{Y}_{i-1,i}) \leq c(\bar{X}_{i-1,i}) + c(Q)$.

We make precisely the symmetric construction to define $\bar{Y}_{i+1}$ and $\bar{Y}_{i+1,i+2}$. We have obtained the required $\boldsymbol{R}$-solution $\bar{\boldsymbol{Y}}$. □

Lemma 1.3.9. *Let* $\bar{\boldsymbol{X}}$ *be a* $\boldsymbol{T}$*-solution that satisfies Property (i) of Definition 1.3.6 but where Property (ii) fails for some terminal i. Then there exists an* $\boldsymbol{R}$*-solution* $\bar{\boldsymbol{Y}}$ *of no higher cost, with* $R_j \subseteq T_j$ *for all* $j \in W$*, and* $R_i \subsetneq T_i$.

Proof. Suppose w.l.o.g. $e = \{u, v\}$ is an edge in $\bar{X}_{i,i+1}$ with both $u, v \in T_i$. Because of Property (i) we know there exists a u-v-path in $\bar{X}_{i-1} \dot\cup \bar{X}_{i-1,i} \dot\cup \bar{X}_i$, say Q.

Let us define a new solution $\bar{\boldsymbol{Y}}$ equal to $\bar{\boldsymbol{X}}$ except for:

$$\begin{aligned} \bar{Y}_i &= (\bar{X}_i \setminus E(Q)) \cup \{e\} \\ \bar{Y}_{i,i+1} &= \bar{X}_{i,i+1} \cup (\bar{X}_i \cap E(Q)) \setminus \{e\}. \end{aligned}$$

The i-$(i+1)$-path in $\bar{\boldsymbol{X}}$ is still feasible in $\bar{\boldsymbol{Y}}$, and we can get an $(i-1)$-i-path $\bar{\boldsymbol{Y}}$ from the respective path in $\bar{\boldsymbol{X}}$, by replacing the subpath Q with the edge e. Thus, $\bar{\boldsymbol{Y}}$ is a feasible solution.

Let P be the maximal path in $\bar{Y}_i$ that contains e. Since every edge on P is used both on some $(i-1)$-i-path and i-$(i+1)$-path, we can replace P by an edge connecting the endpoints in $\bar{Y}_i$ and retain a feasible solution, with the property that

$$V(\bar{Y}_i) \setminus \{i\} \subsetneq V(\bar{X}_i) \setminus \{i\} = T_i,$$

and $V(\bar{Y}_j) = V(\bar{X}_j)$ for $j \neq i$. By Lemma 1.3.5 it now follows that we can find an $\boldsymbol{R}$-solution $\bar{\boldsymbol{Z}}$ with $R_i \subseteq V(Y_i) \setminus \{i\} \subsetneq T_i$ and $R_j \subseteq T_j$ for $j \in W$, as required. □

Proof of Lemma 1.3.7. We arrive at our Lemma from the fact that we can alternatingly apply Lemmas 1.3.8 and 1.3.9 to a $\boldsymbol{T}$-solution $\bar{\boldsymbol{X}}$ until we have a strong $\boldsymbol{R}$-solution $\bar{\boldsymbol{Y}}$. Since every time we apply Lemma 1.3.8, $\sum_{i \in W} |T_i|$ strictly decreases, this procedure must terminate in a finite number of steps. □

1.3.4 *From a $\boldsymbol{T}$-solution to an optimal embedding*

Observation 1.3.10. *Suppose $\bar{\boldsymbol{X}}$ is a strong $\boldsymbol{T}$-solution with $|T_i| = 1$ for all $i \in W$. Then $\bar{\boldsymbol{X}}$ is a hubbed solution.*

As we will see, as long as we have a strong $\boldsymbol{T}$-solution that is not a hubbed solution (implying that $|T_i| = \alpha > 1$ for some α and all i), we can find an $\boldsymbol{R}$-solution such that $\boldsymbol{R}$ is strictly smaller than $\boldsymbol{T}$.

Lemma 1.3.11. *Given a strong $\boldsymbol{T}$-solution $\bar{\boldsymbol{X}}$ with $|T_i| > 1$ for all i, there exists a strong $\boldsymbol{R}$-solution $\bar{\boldsymbol{Y}}$ of no higher cost with $R_i \subsetneq T_{i+1}$ for all i.*

Proof. We claim that we can find a new solution $\bar{\boldsymbol{Y}}$ with $V(\bar{Y}_i) = T_{i+1} \setminus \{u_{i+1}\} \cup \{i\}$ for some node $u_{i+1} \in T_{i+1}$. It then follows by Lemma 1.3.5 and Lemma 1.3.7 that we can find a strong $\boldsymbol{R}$-solution $\bar{\boldsymbol{Z}}$ with

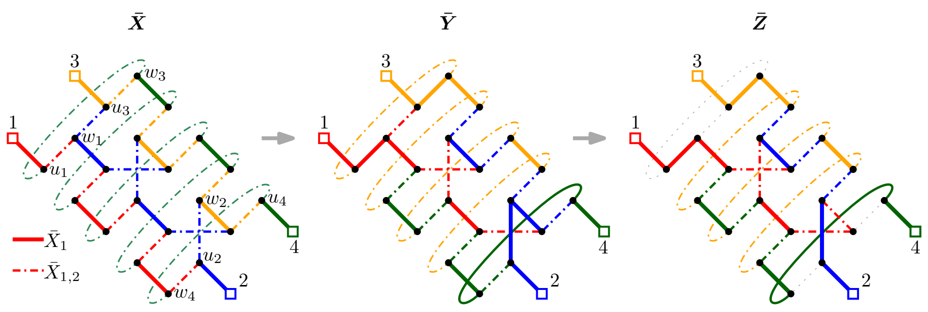
$$R_i \subseteq V(\bar{Y}_i) \setminus \{i\} = T_{i+1} \setminus \{u_{i+1}\},$$

which implies the required result.

For each terminal i we define $u_i \in T_i$ as the node matched to i in $\bar{X}_i$, and $w_i \in T_{i+1}$ as u_i if $u_i \in T_{i+1}$, or the vertex matched to u_i in $\bar{X}_{i,i+1}$ otherwise. Finally let L_i denote the i-u_i-w_i path in $\bar{X}_i \cup \bar{X}_{i,i+1}$.

Now take a solution $\bar{\boldsymbol{Y}}$ equal to $\bar{\boldsymbol{X}}$ except for

$$\begin{aligned} \bar{Y}_i &= \{\{i, w_i\}\} \cup \bar{X}_{i+1} \setminus L_{i+1} \\ \text{and} \qquad \bar{Y}_{i,i+1} &= \bar{X}_{i+1,i+2} \setminus L_{i+1} \setminus L_{i+2}. \end{aligned}$$



We will show that $|T_i| > 1$ implies that $\bar{Y}_i \cup \bar{Y}_{i,i+1} \cup \bar{Y}_{i+1}$ contains an i-$(i+1)$-path. Note that:

$$\bar{Y}_i \cup \bar{Y}_{i,i+1} \cup \bar{Y}_{i+1} = \{\{i, w_i\}, \{i+1, w_{i+1}\}\} \cup E(P_{i+1,i+2} \setminus L_{i+1} \setminus L_{i+2}).$$

As $w_j \in T_{j+1}$ for all j, clearly $P_{i+1,i+2}$ contains a w_i-w_{i+1}-subpath. Since $|T_j| > 1$ for all $j \in W$, we must have that L_{i+1} and L_{i+2} are vertex disjoint, and therefore $E(P_{i+1,i+2} \setminus L_{i+1} \setminus L_{i+2})$ induces a single non-empty connected component.

Now

$$V(P_{i+1,i+2} \setminus L_{i+1} \setminus L_{i+2}) = \begin{cases} V(P_{i+1,i+2}) \setminus \{i+1, u_{i+1}, i+2\} & \text{if } u_{i+1} \neq w_{i+1} \\ V(P_{i+1,i+2}) \setminus \{i+1, i+2\} & \text{otherwise} \end{cases}.$$

But as L_i and L_{i+1} are vertex disjoint, $w_i \neq u_{i+1}$. Therefore $P_{i+1,i+2} \setminus L_{i+1} \setminus L_{i+2}$ contains a w_i-w_{i+1}-path, implying that $\bar{Y}_i \cup \bar{Y}_{i,i+1} \cup \bar{Y}_{i+1}$ contains an i-$(i+1)$-path.

We conclude that $\bar{\boldsymbol{Y}}$ is a feasible solution. Finally note that:

$$\begin{aligned} V(\bar{Y}_i) &= V(\bar{X}_{i+1}) \setminus \{i+1, u_{i+1}\} \cup \{i, w_i\} \\ &= T_{i+1} \triangle \{i+1\} \setminus \{i+1, u_{i+1}\} \cup \{i, w_i\} \\ &= T_{i+1} \setminus \{u_{i+1}\} \cup \{i\}, \end{aligned}$$

where in the last equality we have used that $w_i \in T_{i+1}$. This proves our claim and hence the lemma. □

Recall that $|T_i| = |T_j|$ for all $i, j \in W$ in a strong $\boldsymbol{T}$-solution. By repeatedly applying Lemma 1.3.11, we obtain an optimal $\boldsymbol{T}$-solution with $|T_i| = 1$ for all $i \in W$, which by Observation 1.3.10 is a hubbed solution. This completes the proof of Theorem 1.3.1.

1.4 *The tree case*

We consider the case where H is a bounded-degree tree. Since H is bipartite, we may restrict ourselves to integral solutions to (1.2). We first show that there is an optimal solution of a particular form, which we refer to as a *hubbed solution*.

Lemma 1.4.1. *There exists an optimal solution $\boldsymbol{Y}$ to the tree* MVPN *problem, such that, for some choice $h_{ij} \in V$ for each $\{i,j\} \in E(H)$ (which we call* hub vertices*), Y_i is the edge set of a Steiner tree with terminals $\{i\} \cup \{h_{ij} : \{i,j\} \in E(H)\}$.*

Proof. We prove that we can transform an arbitrary solution $\boldsymbol{Y}$ into a feasible solution $\boldsymbol{Z}$ of the required form.

Choose an arbitrary terminal and consider H to be rooted at this node. Let $C(i)$ denote the set of children of terminal i in H. We construct $\boldsymbol{Z}$ as follows.

We initialize $Z_i = \emptyset$ for all leaf terminals i. Now suppose we have defined Z_j for all the children of a node i. Then define

$$Z_i' = \bigcup_{j \in C(i)} \{e \in Y_i \cup Y_j : e \in P_{ij}\} \setminus Z_j.$$

Now let Z_i be the connected component of (V, Z_i') that contains i. By working up from the leaves of H, this clearly defines $\boldsymbol{Z}$.

Since $Z_i \subseteq \bigcup_{j \in \{i\} \cup C(i)} Y_j$ for all i, and $Z_i \cap Z_j = \emptyset$ for all $j \in C(i)$, $\boldsymbol{Z}$ costs no more in $\boldsymbol{Y}$.

To see that $\boldsymbol{Z}$ is indeed feasible and of the required form, note that for any terminal i and child $j \in C(i)$, by definition $Z_i' \cup Z_j$ must contain an i-j-path. If Z_j is empty, clearly Z_i must contain an i-j path. We set $h_{ij} = j$ and we are done. If not, then there exists a vertex h_{ij} in the single nonempty connected component of Z_j such that Z_i' contains a path from i to h_{ij}. But that path must be contained in the connected component of Z_i' that contains i, which is exactly Z_i, as required. □

With this structural lemma in place, Theorem 1.1.2 follows easily.

Proof of Theorem 1.1.2. We can solve the Steiner tree problem for a fixed number of terminals in polynomial time [28]. Therefore, for H a tree of bounded degree, finding an optimal solution reduces to finding the location of the hub vertices. We will show that we can do this efficiently with dynamic programming.

Suppose we root H at some terminal r. For each terminal i we let $\zeta(i, h)$ denote the minimum cost of the edge sets bought by all terminals in the subtree rooted at i, over all hubbed solutions such that the hub between i and its parent is located at h.

Now, define $\mathrm{MSt}(X)$ for $X \subseteq V$ as the minimum cost of a Steiner tree on terminal set X. We can calculate $\zeta(\cdot, \cdot)$ recursively as follows:

$$\zeta(i, h) = \min_{(h_j)_{j \in C(i)} \in V^{C(i)}} \mathrm{MSt}(\{i, h\} \cup \{h_j : j \in C(i)\}) + \sum_{j \in C(i)} \zeta(j, h_j).$$

In other words, we simply try all possible combinations of choices of $h_{j_1}, h_{j_2}, \ldots$ for $j_1, j_2, \ldots \in C(i)$. As the degree of H is bounded, the number of combinations is polynomially bounded. Since we can solve the Steiner tree instance in polynomial time as well, the result follows.

□

As remarked in the introduction, a constant factor approximation can easily be obtained when H is an arbitrary tree, again from the structural lemma.

Theorem 1.4.2. *The* MVPN *problem where H is a tree has a 2α-approximation, where α is the approximation ratio for Steiner tree.*

Proof. Root H at an arbitrary terminal. Again, let $C(i)$ denote the set of children of terminal i. Take any optimal solution $\boldsymbol{Y}$. Then define a new solution

$Z_i := \bigcup_{j \in \{i\} \cup C(i)} Y_j$, which costs at most $2OPT$. Now for each terminal i, Z_i contains a Steiner tree on i and its children $C(i)$.

Let X_i be an α-approximate Steiner tree on i and its children $C(i)$. Then $\boldsymbol{X}$ is a feasible solution, and $c(\boldsymbol{X}) \leq \alpha \cdot c(\boldsymbol{Z}) \leq 2\alpha \cdot OPT$. □

At the time of writing the best known approximation for Steiner tree is $\ln 4 + \epsilon < 1.39$ [18], so the MVPN problem where H is a tree is approximable within 2.78.

1.5 Conclusion

Our results for H a tree can be extended easily to the capped hose model where the support (edges with $d_{ij} > 0$) forms a tree. If the support of $\boldsymbol{d}$ is a cycle, but $\boldsymbol{b}$ and $\boldsymbol{d}$ are otherwise arbitrary, the situation is unclear. There is a natural analog of the embedding algorithm. First, ensure that the components of $\boldsymbol{b}$ and $\boldsymbol{d}$ are all minimal, i.e., no component can be decreased without changing the uncertainty set. Then compute the cheapest embedding of the weighted version of the graph used in Section 1.3; edges $\{i, h_i\}$ get weight b_i, and edges $\{i, i+1\}$ get weight $d_{i,i+1}$. We leave it as an open question whether this algorithm is always optimal. More speculatively, we feel that our results suggest that embedding algorithms may play a deeper role in the subject than is currently apparent.

2
Improved approximation algorithms for subadditive inventory problems

Abstract We provide the first sub-logarithmic approximation algorithms for the submodular joint replenishment problem and the inventory routing problem. Our work builds upon the work of Nagarajan and Shi [65], who provided the first sub-logarithmic approximation algorithm for the special case of polynomial holding costs.

We study both problems in the context of an common generalization we call the subadditive inventory problem. Here we are given a set of N items and discrete time horizon of T days in which demand for the items must be satisfied. There is a subadditive cost function for ordering a set of items on a given day. Demand for an item at time t can be satisfied by an order on any day prior to t, but a holding cost is charged for storing the items during the intermediate period. The goal is to minimize the sum of the ordering and holding cost.

The submodular joint replenishment problem is the special case in which the ordering cost function is submodular and the inventory routing problem is the case in which the ordering cost function is the routing cost in a metric. We provide (randomized) iterative rounding based $O(\log\log\min(N,T))$-approximation algorithms for both these problems, improving the previous best known approximation ratios of $O(\log T)$ and $O(\log N)$ by an exponential factor.

We also describe techniques for bounding the length of the time horizon in terms of the size of the item set, which enables the unified approximation ratio in terms of N and T. The results are formulated in general terms and could be applied to other ordering cost functions in future research.

This chapter contains unpublished results.

Part of this work (primarily in Section 2.4) was done while the author was visiting the Tepper School of Business at Carnegie Mellon University. The author would like to thank Prof. R. Ravi for hosting him.

The author also thanks Neil Olver, who proposed the (crucial) idea of looking at laminar families first. Apart from that, the many discussions at every stage in the development of this chapter were extremely helpful. Whenever the author had become hopelessly trapped in his own web of definitions, these talks helped craft the simplifications necessary to become unstuck again, and ultimately led to many of the generalizations that make up the rest of this chapter.

2.1 *Introduction*

The inventory problem studied in this chapter captures a number of related models studied in the supply chain literature. One of the simplest is the dynamic economic lot size model [77]. Here we have varying demand for a single item over T time units. Demand at time t or later can be satisfied by an order at time t (but not vice versa). For each day, there is a per unit cost for holding the items in storage. There is also a fixed setup cost for ordering any positive quantity of the item, which is the same for each day. We want to decide on how many items to order on each day of the time horizon so as to minimize the total ordering and holding cost.

The joint replenishment problem (JRP) generalizes this problem to multiple items. We now have a unique holding cost for each day and each item, and a per item setup cost for ordering any quantity of that item. Furthermore, there is a general setup cost for ordering any items at all. This setup cost structure is called *additive*. While having limited expressive power, when compared to some of the more complex generalizations that have been studied, the additive joint replenishment problem is long known to be NP-hard [6]. This problem has attracted considerable attention from the theory community in the past, resulting in a line of progressively stronger approximation algorithms [62, 63], the best of which gives an approximation ratio of 1.791 [12].

A more general version of this problem uses an ordering cost structure in which the setup cost is a submodular function of the items ordered. This model, introduced by Cheung et al. [24], is called the *submodular JRP* and captures both the additive cost structure as well as other sensible models. In the same chapter, the authors give constant factor approximation algorithms for special cases of submodular cost functions, such as tree cost, laminar cost (i.e. coverage functions, where the sets form a laminar family) and cardinality cost (where the cost is a concave function of the number of distinct items). For the general case, they provide an $O(\log NT)$ approximation algorithm, leaving the quest for a constant factor approximation to future researchers.

In the inventory routing problem (IRP) setup costs are routing costs in a metric space. There is a root node and every item corresponds to a point in the metric. The setup cost for a given item set is then given by the length of the shortest tour visiting the depot and every item in the set.[1]

[1] Another sensible way of expressing the routing cost is as the cost of a minimum spanning tree instead of a tour. This reduces the cost by at most a constant factor, and since no constant factor approximation for the IRP is known at this point, the distinction is largely irrelevant.

The IRP has been extensively studied in the past three decades (see [25] for a recent survey), mainly from an experimental computational perspective. The usual interpretation of the model is that the root node represents a central depot and every other point in the metric represents a warehouse to be supplied from the depot. To streamline terminology with the joint replenishment problem, we will keep using the term *items* though. We remark that there is a perfectly natural interpretation to go with this. Each item is sourced from a different location (a factory, say), and needs to be picked up and transported to the depot.

Contrasting with the extensive body of computational results, both for the IRP described here as for more exotic variants of it, very little is known about its approximability. Recently, Fukunaga et al. [42] presented a constant factor approximation under the restriction that orders for a given item must be scheduled *periodically*, i.e. continuously recurring after a fixed number of time units. This restriction appears to significantly simplify the construction of an approximation algorithm, as prior to this chapter the best known polynomial time approximation algorithms gave (incomparable) $O(\log N)$ and $O(\log T)$ performance guarantees.

The replenishment problem with fixed turnover times (RFTT), studied by Bosman et al.([16], Chapter 3), is a related problem in which locations need to be revisited within a fixed time period (turnover time) after the last replenishment. The authors present logarithmic approximation algorithms as well as constant factor approximations on tree metrics for the objective of minimizing average and maximum tour length. While the demand structure is different in this problem, there is a constant factor reduction from the average tour length objective for the RFTT, to the IRP. The RFTT problem has compact input description though, which means that polynomial time algorithms for the IRP do not automatically carry over. Using the techniques similar to the ones in Section 2.2.3, however, these issues can be circumvented.

In the recent work of Nagarajan and Shi [65], upon which this work builds extensively, the logarithmic barrier was broken for the special case of both the IRP and submodular JRP, in which the cost functions and schedules are unrestricted, but the holding costs are assumed to grow as a fixed degree polynomial. This is a very natural restriction; in particular it captures the case where holding an item incurs a fixed rate per unit per day, depending only on the item. For this case the authors provide an $O(\log T/\log\log T)$ approximation algorithm, and express the belief that the techniques developed might prove useful in obtaining a constant factor approximation for the general case of both problems. We provide evidence to this effect, by improving their approximation ratio by an exponential factor to $O(\log\log T)$ and extending the result to arbitrary holding cost. We also provide some general techniques to turn (sub)logarithmic approximation algorithms in terms of T into equivalent algorithms in terms of N.

We should also mention the work of Chekuri, Ene, Vondrak and Wu [22, 23, 34] on submodular partitioning problems. In these problems, a ground set V must be partitioned across k different sets to minimize a submodular cost function. They use rounding of a relaxation based on the Lovász extension to unify and improve several prior results. Their approach inspired the use of the Lovász extension in the rounding algorithm for the submodular joint replenishment problem (SJRP) in this chapter.

Though not we have not reached the ultimate goal of an approximation algorithm with constant factor performance guarantee yet, we hope the techniques presented in this chapter may prove a useful stepping stone for future research. Our main contributions are summarized in the following theorems.

Theorem 2.1.1. *There is a polynomial time $O(\log\log\min(N,T))$-approximation algorithm for the inventory routing problem.*

Theorem 2.1.2. *There is a polynomial time $O(\log\log\min(N,T))$-approximation algorithm for the submodular joint replenishment problem.*

The proofs of Theorems 2.1.1 and 2.1.2 are deferred to the end of Sections 2.3 and 2.4 respectively.

2.1.1 Preliminaries and outline of the chapter

In this chapter, we consider inventory problems with subadditive ordering cost functions. In particular, we will always assume orderings costs are *monotone, nonnegative, normalized and subadditive*, see Definition 2.1.3.

Definition 2.1.3. Let V be a set and $f : V \to \mathbb{R}$ be a set function. The function f is:

- **subadditive** if $f(S \cup T) \leq f(S) + f(T)$ for all $S, T \subseteq V$
- **submodular** if $f(S \cup T) + f(S \cap T) \leq f(S) + f(T)$ for all $S, T \subseteq V$

- **monotone** if $f(S) \leq f(T)$ for all $S \subseteq T \subseteq V$
- **nonnegative** if $f(S) \geq 0$ for all $S \subseteq V$
- **normalized** if $f(\emptyset) = 0$

We use log for the base 2 logarithm. The notation $[k]$ is used for the first k integers $\{1, \ldots, k\}$. The support $\text{supp}(w)$ of a function $w : X \to \mathbb{R}$ is the subset of X for which w maps to nonzero values.

Outline In Section 2.2 we present a general framework for inventory problems with subadditive ordering costs, which covers both the submodular JRP and the IRP. We then provide some general theorems that can be used to simplify the problem, provided we can approximately solve a corresponding (exponential sized) LP relaxation. In Section 2.2.1, following the approach in [65], we reduce the problem with holding costs to a covering problem without holding costs but with strict time windows in which each demand must be served (Theorem 2.2.4). In Section 2.2.2 we then extend the technique by forcing all time windows to a lie in a special family of intervals we call a *left aligned* family (Theorem 2.2.11). Intuitively, this family is more general than a laminar family but retains some of its useful properties, which we exploit later on. Finally, in Section 2.2.3 we show that we can bound the length of the time horizon T polynomially in terms of the size of the item set N (Theorem 2.2.14). This implies that a sublogarithmic approximation in terms of T yields a sublogarithmic approximation in terms of N for free. In Section 2.2.4 we summarize the results in Theorem 2.2.16.

We remark that the general inventory problem we describe in Section 2.2 is based on subadditive functions, departing from the unified framework proposed by Nagarajan and Shi [65]. They consider *β-approximately fractionally subadditive functions*, which have the following property. Let S be a set and $\{S_i, \lambda_i\}$ and $\lambda \geq 0$ a weighted collection of sets that fractionally covers S, i.e. $\sum_{i: v \in S_i} \lambda_i \geq 1$ for all $v \in S$. Then:

$$f(S) \leq \beta \sum_i \lambda_i f(S_i).$$

Though β-approximate fractional subadditivity is incomparable to subadditivity, we argue that subadditive functions with a strong LP relaxations must be approximately fractionally subadditive. Therefore it appears likely that approximate fractional subadditivity is a prerequisite for strong approximation algorithms. However, it is not clear that this property alone is enough, nor do we make use of the property directly in this chapter. For this reason, we stick to the more general formulation.

In Section 2.3 we provide an $O(\log \log T)$-approximation algorithm for the simplified covering problem of the reduction in Section 2.2 when the ordering cost function is given by routing costs. This then implies an $O(\log \log \min(N, T))$-approximation algorithm for the IRP. The algorithm uses randomized iterative rounding of a path based relaxation. We can show that after sampling every path in the support of the relaxation $O(\log \log T)$ times, we can remove a constant fraction of the edges and reorder the remaining paths such that we retain a feasible solution.

In Section 2.4 we provide a matching $O(\log \log T)$-approximation algorithm for the covering problem under submodular ordering costs. We again use iterative rounding, but the approach is fairly different, and naturally deterministic in nature. Instead of randomly rounding item sets in the support of a relaxation, we carefully try to pick a set for each day such that we win a constant fraction of its cost back in the subsequent reduction of the cost of the relaxation. If such a set cannot be found, we can show that after some cleaning up, we can collapse the time horizon by merging adjacent time units and repeat.

2.2 *Reducing general inventory problem to covering problem*

The general framework of the inventory problems we investigate is defined by a set of items[2] $[N] = \{1, \ldots, N\}$, a monotone, nonnegative, normalized and subadditive ordering cost function $f : 2^{[N]} \to \mathbb{R}$ and a time horizon $[T] = \{1, \ldots, T\}$. We will typically refer to the atomic time units as days or simply time units.

[2] Also referred to as *clients* in the context of the IRP, or more generally, nodes.

For each item i and time unit t, there is a nonnegative holding cost $h^i t \geq 0$ and demand $d_{it} \geq 0$. The collection of item-day pairs for which there is positive demand is denoted $D = \{(i,t) : d_{it} > 0\}$. Demand for day t can be satisfied on or before day t. If we satisfy demand for item i on day t, using an order on some day $s < t$, we need to store the items in the intervening days, and we pay a holding cost of $h^i_{t'}$ per unit we store for every day $t' \in [s, t-1]$. The magnitude of the demand only plays a role in the holding cost; the ordering cost is determined by the unique items ordered and is independent of how many units are ordered.

Given these inputs, we need to place an order for items to be delivered on each day so as to minimize the total ordering cost plus the holding cost. Since the cost of delivering an item does not depend on the size of the order and we want to store items as briefly as possible, it is always optimal to deliver just enough units of an item to satisfy demand until the next order for that item is scheduled. Hence, once we decide which items to order on which days, the optimal schedule is completely determined.

The inventory routing problem is the special case where we have a metric on $[N]$, some distinguished root node $r \in [N]$, and the ordering cost $f(S)$ of a set $S \subseteq [N]$ is the length of a shortest tour visiting all of nodes in S and r.[3] The submodular joint replenishment problem is the special where f is a monotone, nonnegative, normalized, submodular set function.

[3] Of course this makes $f(S)$ NP-hard to compute, and in practice we want a solution to provide an explicit tour on S as well. Alternatively, we could use the cost of a minimum spanning tree, which eliminates the issue entirely.

An integer programming formulation for this problem is given in (2.1). Here the variable y^S_t indicates whether item set S is ordered on day t, and x^i_{st} indicates whether the demand for item i on day t is satisfied by an order on day s. Finally we use H^i_{st} here for convenience to denote the sum $H^i_{st} = d_{it} \sum_{s'=s}^{t-1} h^i_{s'}$, i.e., the total holding cost we pay if we satisfy demand for i on day t by an order on day s.

$$
\begin{aligned}
\text{minimize} \quad & \sum_{t\in[T]} \sum_{S\subseteq[N]} f(S) y^S_t + \sum_{t\in[T]} \sum_{i\in[N]} \sum_{s\leq t} H^i_{st} x^i_{st} \\
\text{subject to} \quad & x^i_{st} \leq \sum_{S: i\in S} y^S_s \qquad \forall (i,t) \in D, s \leq t \\
& \sum_{s\leq t} x^i_{st} = 1 \qquad \forall (i,t) \in D \\
& y^S_t, x^i_{st} \in \{0,1\} \qquad \forall i, s < t, S
\end{aligned}
\tag{2.1}
$$

After relaxing integrality constraints of ILP (2.1) to nonnegativity constraints, we are left with an LP with an exponential number of variables. Its dual is

$$
\begin{aligned}
\text{maximize} \quad & \sum_{i,t\in D} b^i_t \\
\text{subject to} \quad & b^i_t - a^i_{st} \leq H^i_{st} \qquad \forall i, s < t \\
& \sum_{i\in S} \sum_{t\geq s} a^i_{st} \leq f(S) \qquad \forall S \subseteq [N], s \in [T] \\
& b^i_t, a^i_{st} \geq 0 \qquad \forall i, t \in D, s \leq t.
\end{aligned}
\tag{2.2}
$$

This dual has an exponential number of constraints, and to solve it in polynomial time we need to be able to separate constraints of the form $\sum_{i\in S} a_i \leq f(S)$ in polynomial time.

For f a submodular function, the separation problem is a submodular function minimizaitn which can be solved in polynomial time.[4] When f is the minimum routing cost in a graph, i.e. $f(S)$ is the cost of a tour visiting S and a specified root node, we can approximately solve it to within a constant factor, as was noted by Nagarajan and Shi [65].

[4] A more detailed discussion can be found in [65], or in the appendix, where we show how to solve a slightly different LP from Section 2.2.2 which poses essentially the same separation problem.

Guide to the remainder of this section We will assume that we have a way to approximately (within a constant) solve the LP relaxation of (2.1) for the class of ordering cost functions we are interested in. As we argued, this is definitely true for submodular and routing cost functions, but we will state our results in general terms to make them applicable to other cost functions as well. Given this, we will show how to reduce the problem to a sequence of progressively simpler problems.

In Subsection 2.2.1, we eliminate the holding costs and generate a problem that has time windows instead. This part is taken directly from the earlier work by Nagarajan and Shi [65]. Then in Subsection 2.2.2 we will show how to reduce this problem to one in which the time windows lie in a particular family of intervals with nice properties. Next, in Subsection 2.2.3 we show how to bound the length of the time horizon T in terms of the item set N. This allows us to turn any approximation ratio in terms of T into one in terms of N. Finally in Subsection 2.2.4 we make some further simplifications and summarize our result in Theorem 2.2.16.

2.2.1 *A bridging problem*

In this subsection we will show how to reduce the problem of rounding an approximate solution to the LP relaxation of ILP (2.1), to rounding a solution to a relaxation of a different problem. This problem is simpler covering problem, in which the holding costs are eliminated entirely. The simplification loses at most a constant factor in the cost of the optimal solution.

The reduction to a covering problem is based on the concept of shadow intervals, a term coined by Nagarajan and Shi in [65]. We replicate their core arguments here, staying with their notation as much as possible. In subsection 2.2.2, we extend the results to derive an even more restricted covering problem that we will exploit in later sections.

Shadow intervals Suppose we have a solution y, x to the LP relaxation of ILP (2.1). For each demand point $(i,t) \in D$ we define the *shadow interval* $[s'(i,t), t]$, where $s'(i,t)$ is the unique time period such that:

$$\sum_{s=s'(i,t)}^{t} x^i_{st} \geq 1/2 \quad \text{and} \quad \sum_{s=s'(i,t)+1}^{t} x^i_{st} < 1/2.$$

The shadow intervals have the property that if we serve each demand point within its shadow interval, the holding costs are guaranteed not to be far from the LP holding costs, as formalized in Lemma 2.2.1.

Lemma 2.2.1 (Lemma 3.1 in Nagarajan and Shi [65]). *Given a solution y, x to the LP relaxation of (2.1), let $[s'(i,t), t]$ be the shadow interval of demand point $(i,t) \in D$. Then:*

$$H^i_{s'(i,t),t} \leq 2 \sum_{s \leq t} H^i_{st} x^i_{st}.$$

Proof. First note that H^i_{st} is monotonically decreasing in s for fixed i, t. Furthermore, by definition of $s'(i,t)$ and the second constraint of LP (2.1), we have that

$$\sum_{s=1}^{s'(i,t)} x^i_{st} = 1 - \sum_{s=s'(i,t)+1}^{t} x^i_{st} > 1 - 1/2 \geq 1/2. \tag{2.3}$$

Hence the required inequality follows:

$$2 \sum_{s \leq t} H^i_{st} x^i_{st} \geq 2 \sum_{s \leq s'(i,t)} H^i_{st} x^i_{st} \geq H^i_{st} \geq H^i_{s'(i,t)t} \quad ,$$

where we first use that $s'(i,t) \leq t$, then use Equation 2.3 and finally monotonicity of H^i_{st}. □

This motivates the following bridging problem. LP (2.4) requires each demand point to be (fractionally) served within its shadow interval, but does not take holding cost into account.

$$\begin{aligned} \text{minimize} \quad & \sum_{t \in [T]} \sum_{S \subseteq [N]} f(S) y^S_t \\ \text{subject to} \quad & \sum_{s=s'(i,t)}^{t} \sum_{S: i \in S} y^S_s \geq 1 \qquad \forall i, t \in D \\ & y^S_t \geq 0 \qquad \forall t, S \end{aligned} \tag{2.4}$$

Reduction We will now show how a good solution to the bridging problem relates to a good solution to the original problem and vice versa.

Lemma 2.2.2 (Lemma 3.3 in Nagarajan and Shi [65]). *The optimal objective value of LP (2.4) is at most*

$$2 \sum_{t \in [T]} \sum_{S \subseteq [N]} f(S) y^S_t,$$

where y is taken from the same LP solution used to generate the shadow intervals.

Proof. We will show that $2y$ is a feasible solution to LP (2.4), which proves the bound.

Note that by definition $\sum_{s=s'(i,t)}^{t} x^i_{st} \geq \frac{1}{2}$ for all $i, t \in D$ and the first constraint of LP (2.1) ensures that $\sum_{s=s'(i,t)}^{t} \sum_{S: i \in S} y^S_s \geq \frac{1}{2}$ as well. Multiplying both sides by 2 gives

$$\sum_{s=s'(i,t)}^{t} \sum_{S: i \in S} 2 y^S_s \geq 1,$$

implying that $2y$ is feasible. □

And in the other direction.

Lemma 2.2.3. *Any integral point to LP (2.4) can be transformed into an integral point for ILP (2.1) at an additional cost of at most:*

$$2 \sum_{t \in [T]} \sum_{i \in [N]} \sum_{s \leq t} H^i_{st} x^i_{st},$$

where x is taken from the same LP solution used to generate the shadow intervals.

Proof. Since any demand point $(i,t) \in D$ can be served anywhere prior to t in the original instance of ILP (2.1), it is clear that a feasible point to the derived LP (2.4) instance is feasible to the original instance. We only need to bound the cost. The ordering cost of the two problems are equal, so the only difference is in the holding cost. By Lemma 2.2.1, the holding cost associated with serving a demand point (i,t) in its shadow interval is at most $2\sum_{s\leq t} H^i_{st} x^i_{st}$. Summing over all i and t gives the required bound, completing the proof. □

The main theorem of this subsection follows almost immediately from the preceding lemmas.

Theorem 2.2.4. *At the loss of a constant factor, we can reduce the subadditive inventory problem to the problem of finding an integral solution to LP* (2.4)*, provided we can find an approximate solution to the LP relaxation of ILP* (2.1)*.*

Proof. Take the approximate solution to the LP relaxation of ILP (2.1) to construct the shadow intervals. The resulting bridging problem has optimal cost at most 2 times the cost of the LP solution by Lemma 2.2.2. Any approximately optimal solution to the bridging problem can be converted back to a solution to the original instance at the loss of at most 2 times the cost of the LP solution in additional holding cost by Lemma 2.2.3. We conclude we lose at most a constant factor in the reduction, finishing the proof. □

In the remainder of this section, we assume we have generated the bridging problem and have an approximate solution to the LP relaxation (2.4) of this problem and want to find an integral solution.

2.2.2 *Subadditive cover over time*

We abstract the objective of finding a cheap integral point to the bridging problem into the following generic problem.

Definition 2.2.5 (Subadditive cover over time)**.** Given are a ground set $[N] = \{1,\dots,N\}$ and monotone, nonnegative, normalized, subadditive set function $f : 2^{[N]} \to \mathbb{R}_{\geq 0}$, as well as a time horizon $[T] = \{1,\dots,T\}$, and a set of demand windows $W_i \subseteq \{[s,t] : s,t \in [T]\}$ for each $i \in [N]$.

We need to assign a set of elements S_t to each day $t \in [T]$, such that every element is covered in each of its demand windows, that is:

$$i \in \bigcup_{r\in[s,t]} S_r \qquad \forall [s,t] \in W_i, \quad \forall i \in [N].$$

The goal is to find an assignment of minimal total cost $\sum_{t\in[T]} f(S_t)$.

We also associate the canonical LP (2.5) with the subadditive cover over time problem. Note that this LP is equivalent to LP (2.4) but rephrased to be more in line with the problem formulation of Definition (2.2.5).

$$\begin{aligned} \text{minimize} \quad & \sum_{t\in[T]} \sum_{S\subseteq[N]} f(S) y^S_t \\ \text{subject to} \quad & \sum_{r\in[s,t]} \sum_{S:i\in S} y^S_r \geq 1 \qquad \forall [s,t] \in W_i, \quad \forall i \in [N] \\ & y^S_t \geq 0 \qquad \forall t, S \end{aligned} \tag{2.5}$$

As a further simplification, we will show we can focus on a restricted subclass of this problem in which the allowed time windows lie in a special family of intervals. Intuitively, these families are close to laminar families, and much of this chapter can be read with laminar families in mind. We will first introduce the families and then show how to reduce the problem to the more restricted class.

Aligned families

Definition 2.2.6. A collection of intervals $\mathcal{F} \subseteq \{[s,t] : s,t \in \mathbb{Z}_{\geq}$ is

- *left aligned*, if for all $[s,t] \in \mathcal{F}$ there exists $i,k \in \mathbb{Z}_{\geq}$ such that $s = k2^i + 1$ and $t < (k+1)2^i$,
- *right aligned*, if for all $[s,t] \in \mathcal{F}$ there exists $i,k \in \mathbb{Z}_{\geq 0}$ such that $s > k2^i$ and $t = (k+1)2^i$.

A family that is a left or right aligned is simply called *aligned*.

Given an aligned family $\mathcal{F}$, the minimal i for which an interval $[s,t] \in \mathcal{F}$ satisfies the properties is called the *level* of $[s,t]$ and denoted $\ell([s,t])$.

The names left and right aligned come from the visual similarity to the text formatting styles, when plotting the intervals in the family as lines stacked above each other (see Figure 2.1).

Note that if $\mathcal{F}$ is both left and right aligned, it is special case of a laminar family. In particular, a subfamily of the dyadic intervals $\{(k2^i, (k+1)2^i] : i,k \in \mathbb{Z}_{\geq}\}$. Another way to define aligned families is with respect to this family of intervals. This route provides more visual intuition and is given in Definition 2.2.7

Definition 2.2.7. Let $\mathcal{D} = \{(k2^i+1, (k+1)2^i] : i,k \in \mathbb{Z}_{\geq}\}$ denote the family of dyadic intervals over the nonnegative integers.

A family of intervals $\mathcal{F} \subseteq \{[s,t] : s,t \in \mathbb{Z}_{\geq 0}\}$ is called

- *left aligned* if for all $[s,t] \in \mathcal{F}$ there exists $[s,t'] \in \mathcal{D}$ with $t' \geq t$,
- *right aligned* if for all $[s,t] \in \mathcal{F}$ there exists $[s',t] \in \mathcal{D}$ with $s' \leq s$.

A family that is a left or right aligned is simply called *aligned*.

Given an aligned family $\mathcal{F}$, the minimal enclosing interval $[s',t'] \in \mathcal{D}$ of an interval $[s,t] \in \mathcal{F}$ is called the *parent* of $[s,t]$ and denoted $p([s,t])$.

So in an aligned family, every member has an enclosing interval in $\mathcal{D}$ with which it shares a left/right (depending on the type) endpoint. It is easy to verify that Definitions 2.2.6 and 2.2.7 give equivalent characterizations of left and right aligned. Definition 2.2.7 also illustrates how the concept of level is related to the concept of level in a laminar family, as the level of an interval in an aligned family corresponds to the level of its parent in the laminar family $\mathcal{D}$.

An example of a left aligned family compared to the dyadic family is shown in Figure 2.1.

Using Definition 2.2.7 the following observation about the symmetry between left and right aligned families is immediately clear.

Observation 2.2.8. *Let $\mathcal{F}$ be a left (right) aligned family of intervals, defined on some ground set $[1,2,3,\ldots,2^k]$. Then, $\mathcal{F}$ becomes right (left) aligned if we reverse the order of the ground set, i.e. under the mapping $t \to 2^k + 1 - t$.*

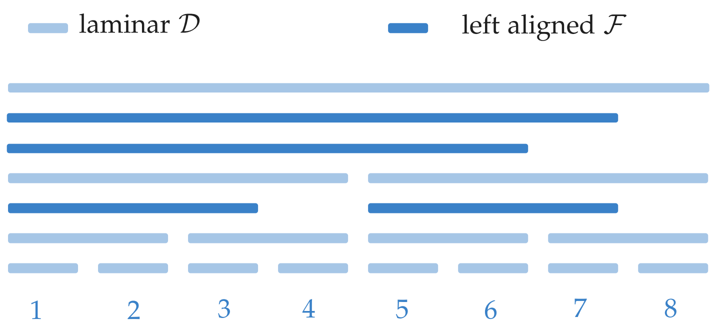


Figure 2.1: Example of laminar and left aligned families of intervals on $\{1,\ldots,8\}$. Note that the union of $\mathcal{F}$ and $\mathcal{D}$ is also left aligned.

Reduction to aligned instances

Definition 2.2.9. Given an interval $[s,t]$, the right aligned part $R([s,t])$ and the left aligned part $L([s,t])$ form a partition of $[s,t]$ and are defined as:

$$R([s,t]) = [s, k2^i] \quad \text{and} \quad L([s,t]) = [k2^i + 1, t],$$

where i, k are integers, such that $k2^i \in [s,t]$ and i is maximal. If $k2^i = t$, then $L([s,t]) = \emptyset$, and if $k2^i + 1 = s$ then $R([s,t]) = \emptyset$ by convention.

It is clear from the definition that the left aligned part of an interval is part of a left aligned family and similarly for the right aligned part. Furthermore, any LP solution must cover every item by at least half in either the right or left aligned part of its demand window, as Observation 2.2.10 formalizes.

Observation 2.2.10. *Let y be a solution to LP (2.5). Every demand window $[s,t] \in W_i$ for $i \in [N]$ is covered at least half in its left aligned part or at least half in its right aligned part:*

$$\sum_{t \in L([s,t])} \sum_{S: i \in S} y_r^S \geq \frac{1}{2} \quad \text{or} \quad \sum_{t \in R([s,t])} \sum_{S: i \in S} y_r^S \geq \frac{1}{2}.$$

We will call an instance of subadditive cover over time *left (right) aligned* if there exists a left (right) aligned family $\mathcal{F}$, such that all time windows in the instance $\bigcup_{i \in [N]} W_i \subseteq \mathcal{F}$ are contained in it.

Theorem 2.2.11. *At the loss of a constant factor, we can reduce the subadditive cover over time problem to a pair of left aligned subadditive cover over time problems.*[5]

[5] See Figure 2.2 for an illustration.

Proof. Let y be a solution LP (2.4). We will first generate two new instances of the subadditive cover over time problem, one being left aligned and the other right aligned. For each $i \in [N]$ and demand window $[s,t] \in W_i$, if $L([s,t])$ receives half a unit of coverage under y, put $L([s,t])$ in the left aligned instance and put $R([s,t])$ in the right aligned instance otherwise.

The resulting instances are left and right aligned respectively by construction. Moreover, $2y$ is a feasible solution to each of the instances, which follows from Observation 2.2.10 and the way we constructed the instances. Hence the combined cost of the optimal solutions to the LP relaxations of the generated instances is at most 4 times that of the original instance. Furthermore, we can translate integral solutions to the left and right aligned instances back to one for the original instance by adding them together, which does not increase the cost by subadditivity of f.

The theorem still requires a pair of left aligned instances, which can be resolved by reversing the time axis of the right aligned instance. This does not change the cost of the solution, and by Observation 2.2.8 makes the problem left aligned. This concludes the proof.

□

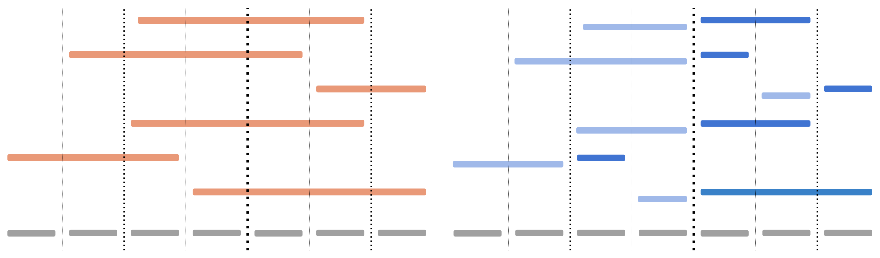

original time windows W_i
split into left/right aligned families

Figure 2.2: On the left an arbitrary collections of intervals is shown. On the right the intervals are split, resulting in a left aligned (light blue) and right aligned (dark blue) family of intervals.

2.2.3 *Bounding the time horizon*

Finally, we will show how to reduce the problem to the case where the time horizon is bounded by the size of the item set N. This allows us to focus on proving approximation ratios in terms of T, as they carry over to approximation ratios in terms of N without further adjustments.

We introduce the following notation for the cost of a singleton item set:

$$f(i) = f(\{i\}).$$

First, we show that the cost of singleton items are not too far apart.

Lemma 2.2.12. *At the loss of a constant factor, we can reduce the left aligned subadditive cover over time problem to a collection of left aligned subadditive cover over time problems with* $\min_{i \in [N]} f(i) \geq \frac{1}{N} \max_{i \in [N]} f(i)$.

Proof. We will iteratively split the item set $[N]$ into a sequence of sets $V_1, V_2, \ldots$ with items of descending singleton cost $f(i)$. Initially, let V_1 contain $[N]$. Then, for $j = 1, \ldots, N$ and while $|V_j| > 0$, let $\mu_j = \max_{v \in V_j} f(i)$. Remove all items v with $f(v) < \mu_j / |V_j|$ from V_j and add them to V_{j+1}.

Now each set V_j induces an instance of the required form. We claim that if we treat all these instances separately and take the union of their schedules the resulting schedules cost at most three times the original optimum. The following construction gives a set of schedules that satisfies this bound.

Take an optimal solution $S_1, \ldots, S_T$ to the original instance and generate a schedule for each instance V_j by selecting item set $S_t^j := S_t \cap V_j$ on day $t \in [T]$.

On each day t, the total cost of the resulting schedule is dominated by the cost of two item sets S_t^j, S_t^{j+1} where j is the smallest index for which S_t^j is nonempty. To be precise, it holds that:

$$\sum_{k \geq j+2} f(S_t^k) \leq \sum_{v \in \bigcup_{k \geq j+2} S_t^k} f(v) \leq \sum_{v \in \bigcup_{k \geq j+2} S_t^k} \mu_{j+1} / |\bigcup_{k \geq j+1} V_k| \leq \mu_{j+1}.$$

Here, in the first inequality we use subadditivity of f. For the second, we use the definition of the cutoff value for items that are removed from V_{j+1}

and put in V_{j+2}. In the last inequality, we use that $\bigcup_{k\geq j+2} S_t^k \subseteq \bigcup_{k\geq j+1} V_k$ and therefore $|\bigcup_{k\geq j+2} S_t^k| \leq |\bigcup_{k\geq j+1} V_k|$.

Then, using $\mu_j = \max_{v\in V_j} f(v)$, $\max_{v\in V_{j+1}} f(v) \leq \min_{v\in V_j} f(v)$, and $S_j^t \subseteq V_j$, we get that:

$$\sum_{k\geq j+2} f(S_t^k) \leq \mu_{j+1} \leq \min_{v\in V_j} f_v \leq f(S_j^t).$$

In particular, the cost of the union of the schedules is at most

$$f(\bigcup_{k\geq j} S_t^k) \leq \sum_{k\geq j} f(S_t^k) \leq f(S_t^j) + f(S_t^{j+1}) + f(S_t^j) \leq 3f(S_t),$$

as required. Finally, we note we did not touch the demand windows and therefore the instances generated are also left aligned. This finishes the proof. □

We also need the following sparsity result on near-optimal solutions to LP relaxation (2.5) of the subadditive cover over time problem. It allows us to assume that every day, the fractional coverage of the sets in the LP solution adds up to at least a full unit or it covers no items at all. The intuition behind the proof is that if it is not the case, we can group time intervals with less than a unit coverage, and copy all coverage inside those intervals to the endpoints.

Lemma 2.2.13. *Take a solution y to LP (2.5). Then there exists a solution $\bar{y}$ whose cost is within twice the cost of y, that satisfies for each day t:*

$$\sum_{S\subseteq[N]} \bar{y}_t^S \geq 1 \quad \textit{or} \quad \sum_{S\subseteq[N]} \bar{y}_t^S = 0.$$

Proof. Copy y to a new solution $\bar{y}$. Call a day t *bad* if $\sum_{S\subseteq[N]} \bar{y}_t^S \in (0,1)$ and *good* otherwise. We iteratively fix $\bar{y}$, maintaining that at each point there exists an index k such that the first k days are good and $\sum_{t=1}^k \sum_S f(S)\bar{y}_t^S \leq \sum_{t=1}^k \sum_S f(S) 2y_t^S$.

Starting with $k = 0$, suppose the first k days are fixed. Find the first day $\ell > k$ that is bad, and let $m > \ell$ be the first index such that the fractional coverage of sets on the interval $[\ell, m]$ sums to at least one, i.e. $\sum_{t=\ell}^m \sum_{S\subseteq[N]} y_t^S \geq 1$, if such an index exists. We will deal with the other case, i.e. $\sum_{t=\ell}^T \sum_{S\subseteq[N]} y_t^S < 1$ later.

Since m is the minimal, it must be that y fractionally covers strictly less than a full set on the interval $[\ell, m-1]$. It follows that every node v whose time window overlaps with $[\ell, m]$ must have $\ell \in F_v$ or $m \in F_v$. So, we can copy all coverage inside $[\ell, m]$ in y to each of the endpoints ℓ and m and retain a feasible solution, i.e.

$$\bar{y}_\ell \leftarrow \sum_{t=\ell}^m y_t \quad \text{and} \quad \bar{y}_m \leftarrow \sum_{t=\ell}^m y_t \quad \text{and} \quad \bar{y}_t \leftarrow 0 \quad \text{for} \quad \ell < t < m$$

After doing this all days up to m are good, and the additional cost can be charged against $y_\ell, \ldots, y_m$, so we maintain the cost bound. We only need to deal with the edge case where $\sum_{t=\ell}^T \sum_{S\subseteq[N]} y_t^S < 1$. In this case, any node v whose time window overlaps with $[\ell, T]$, must have $\ell - 1 \in F_v$, otherwise it could not receive at least a unit coverage under y. Hence, we can copy all coverage to $\ell - 1$ without losing feasibility

$$\bar{y}_{\ell-1} \leftarrow \sum_{t=\ell}^{T} y_t \quad \text{and} \quad \bar{y}_t \leftarrow 0 \quad \text{for} \quad \ell < t \leq T.$$

The cost of this action can be charged against $y_\ell, \ldots, y_T$. Since, we already know $\ell - 1$ is a good day and we increase its coverage, we only need to worry about days ℓ to T, which get set to 0 and thus become good. So, we conclude all days are good and $\bar{y}$ costs at most twice the cost of y, finishing the proof. □

Theorem 2.2.14. *At the loss of a constant factor we can reduce the left aligned subadditive cover over time problem to a collection of left aligned subadditive cover over time problems with $T = N^2$.*

Proof. By Lemma 2.2.12 we can reduce a left aligned instance to a collection of instances with $\min_i f_i \geq \frac{1}{N} \max_i f_i$. By Lemma 2.2.13, we may assume we have for each of those instances an LP solution with $\sum_S y_t^S \geq 1$ or $\sum_S y_t^S = 0$ for all t. At this point, we might as well delete the days t with $\sum_S y_t^S = 0$ from the instance. So, we can assume that $\sum_S y_t^S \geq 1$ for all t.

Suppose that we simply schedule the entire set $[N]$ at time $t = N^2$. By Lemma 2.2.12, this costs at most

$$f([N]) \leq N \max_i f_v \leq N^2 \min_i f_i.$$

By Lemma 2.2.13, on the other hand, the cost of the LP solution for any day t is at least

$$\sum_S f(S) y_t^S \geq \sum_S (\min_i f_i) y_t^S \geq \min_i f_i.$$

So, the LP solution for the first N^2 days costs at least $N^2 \min_i f_i$. This means that we can charge the cost $f([N])$ of covering all elements at day N^2 against the LP, losing only a constant factor in the cost of the solution.

If we cover all elements on day N^2, we effectively reset the instance and we only need a schedule for the first N^2 days. Hence, we can delete any remaining time periods for the instance and we complete the proof. □

2.2.4 *Approximating the subadditive inventory problem*

Finally, we make some assumptions that are mainly useful to reduce notation in the coming sections. We assume that every item $i \in [N]$ has only one demand point i, t with $d_{it} > 0$. If not we make multiple copies of each item, one for every day t with positive demand.[6] We also assume that T is of the form 2^{2^k} for $k \in \mathbb{N}$. This ensures that $\log \log T$ is a positive integer. If it is not the case we can simply round up T.

This blows up the number of items and length of the time horizon by a polynomial factor[7], but this causes only a constant factor increase in the approximation ratio of any (sub)-logarithmic approximation algorithm, like the ones presented in the rest of this chapter.

Our assumptions are summarized in Definition 2.2.15.

Definition 2.2.15. An instance of subadditive cover over time is *nice* if

- the time windows form a left aligned family

[6] We should take care to apply the techniques in the previous section to reduce the time horizon first though. This ensures that the time horizon is bounded in the size of the underlying item set, and not the size of the expanded item set a copy for each demand point.

[7] In expanding the item set, N gets multiplied by a factor $T = O(N^2)$, growing to $O(N^3)$. The time horizon gets multiplied by a factor T as well and grows to $O(N^4)$.

- each item has positive demand on exactly one day
- $T = 2^{2^k}$ for some $k \in \mathbb{N}$

This brings us to the main result of this section, Theorem 2.2.16, the proof of which is found by combining Theorems 2.2.4, 2.2.11 and 2.2.14.

Theorem 2.2.16. *Consider a class of inventory problems for which we can find an $O(1)$-approximate solution to ILP (2.1). Furthermore, suppose we have a polynomial time $O(\log\log T)$-approximation algorithm for nice instances of the corresponding subadditive cover over time problem.*

Then there is an $O(\log\log\min(N, T))$-approximation algorithm for that class of subadditive inventory problems.

2.3 *Steiner tree over time*

In this section we give an algorithm for a special case of the subadditive covering over time problem. Namely, we consider *Steiner tree over time* (STOT). As before, we have a discrete time horizon $[T] = \{1, \ldots, T\}$. This time we get a set of nodes V in a semimetric space with distance function $c : V \times V \to \mathbb{R}_{\geq 0}$. Furthermore, there is distinguished root [8] node $r \in V$ and every other node $v \in V$ has a time window $F_v = [s, t]$ for $s, t \in [T]$. We want to connect every node in $V - r$ to r inside its time window. Formally, we need to find a tree $\mathcal{T}_t$ rooted at r for each day $t \in [T]$, such that every node $v \neq r$ is connected to r by some tree $\mathcal{T}_t$ for $t \in F_v$. The cost of a tree $\mathcal{T}$ (i.e. the sum of the length of its edges), is denoted $c(\mathcal{T})$. The objective is then to minimize the total cost of the trees $\sum_t c(\mathcal{T})$.

[8] The words depot and root are used interchangeably in this section.

While the algorithm in this section is developed with the express purpose of using it to approximate the IRP, the routing costs formulation of the Steiner tree over time problem diverges from that of the IRP. The reason for this is that our algorithm fundamentally constructs a tree, and not a tour, so the Steiner tree over time formulation is more natural. After constructing a solution with tree cost, we can turn it into a TSP cost based solution by doubling the edges of the tree and shortcutting (at the loss of a constant factor), so we lose no generality by using this formulation.

The Steiner tree over time problem should not be confused with the problem known in the literature as the *minimum spanning tree with time windows* problem (Solomon (1986) [76]).

Here, we again have a depot and a set of nodes with time windows. The goal is to construct a single tree, however, and provide an order on the nodes such that a vehicle travelling at unit speed on the tree (optionally waiting at nodes), can visit the nodes within their time windows.

Interestingly, this problem has an analogous motivation to the Steiner tree over time problem: it captures the 'difficult core' of the vehicle routing problem with time windows, by eliminating the problem of finding a minimum cost tour. The STOT problem has a similar relation to the IRP problem.

Assumptions In the rest of this section we will always assume that an instance is nice, which only loses a constant factor by the arguments in Subsections 2.2.2–2.2.3. Furthermore, we associate with each node v a *level* in the natural way, namely $\ell(v) = \ell(F_v)$ equal to the level of its time window in the left aligned family.

2.3.1 *Fractional path relaxation*

The main part of our result works by iteratively massaging a specific type of fractional solution until it becomes integral. We now describe this type of solution.

We let $\mathcal{P}$ denote the collection of directed paths in V. For each such directed path $P \in \mathcal{P}$, let $P \odot_t v$ signify that P connects v to $\mathcal{T}_t$, i.e. $P \odot_t v$ if there is a directed subpath on P from v to a node in the tree $\mathcal{T}_t$ containing the root on day t. If $v \in \mathcal{T}_t$ we let $P \odot_t v$ hold for all P by convention.

Definition 2.3.1 (Fractional path solution)**.** A fractional path solution (FPS) is a pair $w, \mathcal{T}$, giving for each day t a tree $\mathcal{T}_t$ rooted at r and weights $w_t : \mathcal{P} \to [0,1]$, with the property that:

$$\sum_{t \in F_v} \sum_{P \odot_t v} w_t(P) \geq 1 \qquad \forall v \in V - r.$$

The cost of the fractional path solution is given by the sum of the cost of the trees and the cost of the paths weighted by w:

$$\sum_t \sum_P w_t(P) c(P) + \sum_t c(\mathcal{T}_t).$$

Note that a fractional path solution with $w_t(P) \in \{0,1\}$ for all P and t yields a feasible solution to the Steiner tree over time problem.

Moreover, we can start with a solution y to LP 2.5 and convert it to a fractional path solution at the loss of a constant[9]. Hence, we focus on turning a fractional path solution into an integral one without losing too much in the cost of the solution. Loosely speaking, the way we achieve this is as follows.

[9] We inherit such a solution y from the reduction if we use the algorithm in this section to solve the IRP. We can however, also solve the LP for tree cost more directly, see Section 2.5.1 of the Appendix for pointers.
To convert y to an FPS: initialize the trees to contain only the depot. For each day t and set S in $\text{supp}(y_t)$, construct an minimum spanning tree on $S + r$, and add a DFS path P in that tree to the solution with weight $w_t(P) = y_t^S$. The cost of such a path satisfies $c(P) = O(f(S))$ regardless of the type of routing cost chosen.

First, we sample each path P on day t with probability roughly equal to $\log \log T$ times $w_t(P)$. If the path is sampled, we add all nodes on it to the tree for day t. Next, we try to reduce the cost of the fractional paths by deleting edges, wherever the addition of nodes to the root tree has made this possible. We iterate until the trees form a feasible solution to the Steiner tree over time problem. A more detailed description of the procedure follows in the rest of the section, after we developed some necessary definitions and technical lemmas. For now, we state our main result.

Theorem 2.3.2. *Given a fractional path solution of cost C_{FPS} we can find a randomized solution to the Steiner tree over time problem, of expected cost $O((\log \log T) C_{FPS})$.*

2.3.2 *Algorithm details*

We need some preliminary notation and definitions before we describe the algorithm in more detail.

Given a directed path P, we use $v \in P$ or $e \in P$ to denote v or e is a node or edge on P. The head of P, denoted $\text{head}(P)$, is the unique node $v \in P$ without an outgoing edge. The $\text{tail}(P)$ is the unique node $v \in P$ without an incoming edge. Given a collection of edges E, we denote by $P \divideontimes E$ the collection of directed paths obtained by deleting all edges in E from P, disconnecting P into multiple subpaths. Given an edge $e \in P$, we use $P[e]$ to denote the path in $P \divideontimes \{e\}$ containing $\text{tail}(P)$. Given a tree $\mathcal{T}$ and path P, we use $\mathcal{T} + P$ to denote the tree spanning the vertices of $\mathcal{T}$ and P, constructed from the edges of $\mathcal{T}$ and P.[10]

[10] For example, take the union of the edges in $\mathcal{T}$ and P and delete edges from cycles until the edge set becomes acyclic again.

Definition 2.3.3. Given a fractional path solution $w, \mathcal{T}$, for each $v \in V - r$ we divide its time window $[s,t] = F_v$ into two intervals (which overlap at the boundary)

$$S_v = [s, s(v)] \quad \text{and} \quad R_v = [s(v), t],$$

where $s(v)$ is the unique time unit such that:

$$\sum_{t \in F_v : t \geq s(v)} \sum_{P \odot_t v} \geq \frac{1}{2} \quad \text{and} \quad \sum_{t \in F_v : t > s(v)} \sum_{P \odot_t v} < 1/2.$$

We call S_v the *sow* phase and R_v the *reap* phase of v.

The following definition makes sense in the context of Procedure 2.3.5, where at each point we have an FPS $w, \mathcal{T}$ from the previous iteration, and a new FPS $w', \mathcal{T}'$ under modification in the current iteration.

Definition 2.3.4. Let $w, \mathcal{T}$ and $w', \mathcal{T}'$ be two fractional path solutions, the *old* and *new* fractional path solution respectively. Let S_v and R_v denote the sow and reap phase wrt the old weights w.

We say a node v has *germinated* if it has been connected to the new root tree $\mathcal{T}_t'$ for some $t \in S_v$ in v's old sow phase. For $i = 1, 2, ..$ and a path P, we say an edge $e \in P$ is *i-redundant* if the last node v on $P[e]$ with level $\ell(v) \leq i$ has germinated. If no such node v exists the edge is i-redundant by convention. An edge e is *fully redundant* if it is redundant for all $i = 0, 1, \ldots, \log T$.

On a high level, the algorithm does the following. We start with a fractional path solution $w, \mathcal{T}$, and iteratively round it until $\mathcal{T}$ induces a feasible solution to the Steiner tree over time problem. Each iteration consists of the following main steps.

First we sample paths for each day t with probability proportional to $\log \log T$ times the weights w_t, and expand the root trees of the corresponding days with the sampled paths. Then, we remove all nodes from paths not in their reap phase, doubling the weights to compensate for this so that they retain a unit connectivity. Finally, we delete all fully redundant edges, splitting the paths into multiple components. We reassign each of the created paths to a day for which the head has been connected to the root tree to retain feasibility.

The intuition is this. Suppose e is fully redundant. Then all nodes on $P[e]$ are either connected to the root already, or can be reconnected to the root by the connection of some node on $P[e]$. Hence, we can safely delete e. The cost bound of the algorithm then comes from showing that we delete enough fully redundant edges to reduce the cost of the solution by a constant fraction, after taking the effect of doubling the weights into account.

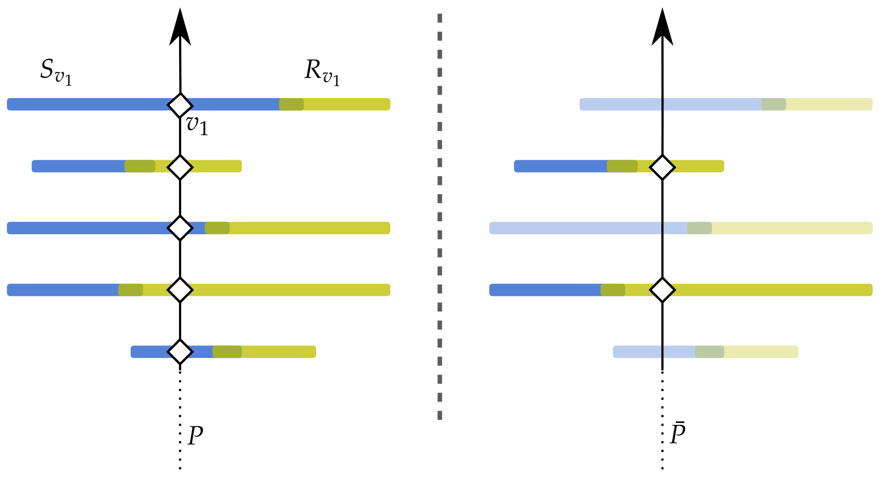


Figure 2.3: Deleting nodes from path $\bar{P}$ for which t is not in the reap phase. The bars represent the relative position of t wrt each node's sow (blue) and reap (green) phase. E.g. v_1's sow phase S_{v_1} contains t, so v_1 is deleted.

In the description of Procedure 2.3.5 K_{smp} is some appropriately large constant to be fixed later.

Procedure 2.3.5. Input: a fractional path solution $w, \mathcal{T}$.

1. Set
$$\mathcal{T}' \leftarrow \mathcal{T}.$$
For all $t \in [T]$ and $P \in \text{supp}(w_t)$:

   With probability $\min[K_{smp} \log\log(T) w_t(P), 1]$ add P to the tree:
$$\mathcal{T}'_t \leftarrow \mathcal{T}'_t + P.$$
   Let $\bar{P} \leftarrow P$ and delete[11] from $\bar{P}$ all v s.t. $t \notin R_v$. Set
$$w'_t(\bar{P}) \leftarrow 2w'_t(P).$$

Copy the trees to a new solution.

First sample the paths with probability proportional to $\log\log(T)w_t(P)$ and expand the trees with the sampled paths. Then strip each node v from every path not in its reap phase R_v and copy double the weights to the new solution.

[11] Shortcutting incident edges. See Figure 2.3 for an illustration.

2. For all $t \in [T]$ and $P \in \text{supp}(w'_t)$:

   For each $P' \in P \circledast \{e \in P : e \text{ is fully redundant}\}$:

      Let $t' = \max\{t' \leq t : \text{head}(P') \in \mathcal{T}'_{t'}\}$. Set
$$w'_{t'}(P') \leftarrow w'_{t'}(P') + w'_t(P).$$
   Finally, set $w'_t(P) \leftarrow 0$.

We delete fully redundant edges and reconnect each component to the latest root tree prior to t that contains its head. Such a tree is guaranteed to exist since the definition of fully redundant ensures the head of each newly generated component has germinated.

See Figure 2.4 for an illustration of this step.

3. Set
$$w \leftarrow w' \quad \text{and} \quad \mathcal{T} \leftarrow \mathcal{T}'.$$
If $\mathcal{T}'$ induces a feasible solution to the Steiner tree over time problem, return $\mathcal{T}$. Otherwise, repeat from Step 1.

Output: a solution to Steiner tree over time problem $\mathcal{T}$.

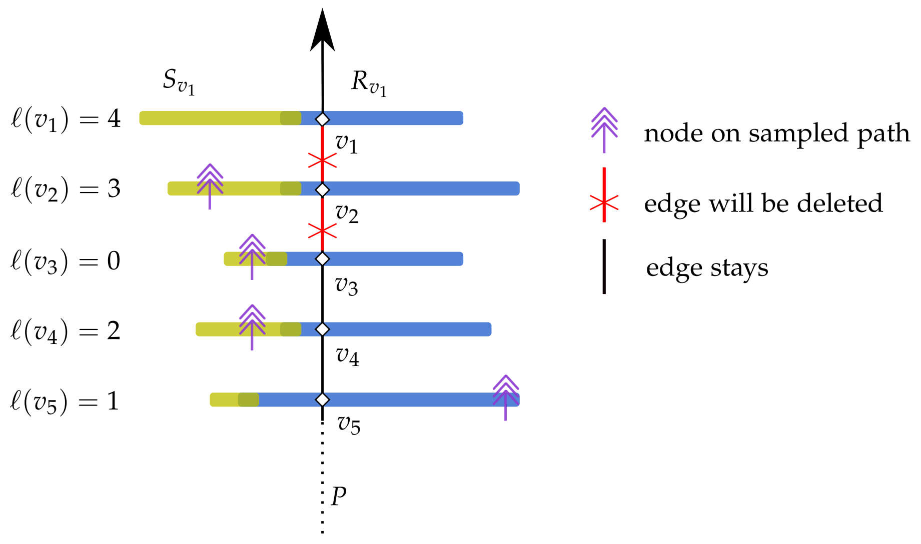


Figure 2.4: Illustration of path P in Step 2. Node v_5 is on a path that has been sampled, but outside its sow phase, so v_5 has not germinated. Hence we cannot remove its outgoing edge. The same holds for v_4 even though it has germinated. We can remove the edge out of v_3. Since $\ell(v_3) = 0$ and v_3 has germinated, this edge is i-redundant for all $i \geq 0$.

The connected components will be reassigned as follows. The component containing v_5, v_4, v_3 will be added to the day indicated by the purple arrow for node v_3. The component containing v_2 gets moved to the day indicated by the purple arrow for node v_2 and the component containing v_1 remains in place.

Feasibility We first show feasibility of Procedure 2.3.5.

In the first two lemmas, which are about Step 2, P, t, P' and t' correspond directly to the variable in the scope of that step.

Lemma 2.3.6. *Consider any execution of the outer loop of Step 2. Suppose edge $e \in P$ is fully redundant and let v be the last node on $P[e]$ with $\ell(v) \leq i$.*

Then the edge e' going out of v is also fully redundant.

Proof. Suppose not. Then there is some $j < i$, such that the last node $u \in P[e']$ with $\ell(u) \leq j$ has not germinated. But since v is the last node on $P[e]$ with $\ell(v) \leq i$, clearly u is the last node on $P[e]$ with $\ell(u) \leq j$ as well. This implies that e is not fully redundant, a contradiction. □

Lemma 2.3.7. *Consider any execution of the inner loop of Step 2.*

It holds that $t' \in F_u$ for all $u \in P'$.

Proof. Let $v = \text{head}(P')$ and recall that $t' = \max\{t' \leq t : \text{head}(P') \in \mathcal{T}'_{t'}\}$. First note that $t' \in F_v$: either $v = \text{head}(P)$ as well and thus $v \in \mathcal{T}'_t$, or P' was split off by deleting the edge e going out of v. This means e was fully redundant, which means v must have germinated. So v is connected to the root at some time $t'' \in S_v \subseteq F_v$, and $t'' \leq t'$ by maximality.

Now, if no node u with $\ell(u) < \ell(v)$ exists, the lemma must hold, since in that case $\min F_u \leq \min F_v$ for all $u \in P$. We claim that indeed all $\ell(u) \geq \ell(v)$ for $u \in P'$, which in turn implies the lemma.

Suppose not. Take the last node $u \in P'$ with $\ell(u) = j < \ell(v)$. Since e is fully redundant, the edge e' going out of u must be fully redundant as well, by Lemma 2.3.6. But this implies that e' should have been deleted and therefore u is not on the same component as v, a contradiction. This completes the proof. □

Lemma 2.3.8. *When Procedure 2.3.5 returns, it outputs a feasible solution to the Steiner tree over time problem.*

Proof. We will show that the fractional path solution remains feasible throughout the execution of the algorithm. Since an FPS with zero weights is a solution to the Steiner tree over time problem, the result then follows.

In Step 1, we expand the trees which cannot reduce connectivity. We delete every node v from paths on times not in their reap phase R_v, but by definition this leaves at least $\frac{1}{2}$ a unit connectivity, so after doubling the weights all nodes are again fully connected.

Now consider any execution of the inner loop of Step 2. By Lemma 2.3.7, every node u on component $P' \in P \divideontimes \{e \in P : e \text{ is fully redundant }\}$, has $t' \in F_u$. Hence $P' \odot_{t'} u$ and no connectivity of u is lost in this step.

Step 3 just loops without altering the solution, so feasibility is clearly not affected. This completes the proof. □

Cost analysis We now develop the lemmas needed to bound the cost of Procedure 2.3.5.

We first show that in each main iteration of Procedure 2.3.5, the expected cost of the trees grows by $O(\log\log(T))$ times the cost of the paths of the FPS.

Lemma 2.3.9. *After Step 1 the expected cost of the new collection of trees is*

$$E\left[\sum_t c(\mathcal{T}_t')\right] = \sum_t c(\mathcal{T}_t) + O(\log\log(T)) \sum_t \sum_P w_t(P)c(P).$$

Proof. This follows directly from the fact that we sample the paths with weights $O(\log\log T)$ times w, and every time we sample a path we use a subset of its edges to connect nodes to the trees. □

In Step 1 we also double the cost of the path weights, in the worst case.

Lemma 2.3.10. *After Step 1, the cost of the new paths is*

$$\sum_t \sum_P w_t'(P)c(P) \le 2\sum_t \sum_P w_t(P)c(P).$$

Proof. Directly from the description and the fact that deleting nodes from a path does not increase its cost. □

However, we can delete a large enough fraction of the edges to counterbalance this. We first need on intermediate step. Recall that a node has germinated if it has been connected to the root in its sow phase.

Lemma 2.3.11. *The probability that a node v has not germinated after Step 1 is*

$$\epsilon/\log T,$$

where ϵ can be made arbitrarily small by making K_{smp} large enough.

Proof. Let us write $K' := K_{smp}\log\log T$ to reduce notation. Pick any node v. Recall that v has germinated if it is connected to the root tree for some $t \in S_v$. By definition of the sow phase S_v we have

$$\sum_{t\in S_v} \sum_{P\odot_t v} w_t(P)K' \ge \frac{1}{2}K'. \tag{2.6}$$

If $w_t(P)K' \ge 1$ then v has germinated with probability 1, so assume not. Then the probability that a given path P does not get sampled at time t is $(1 - w_t(P)K')$. Since each path P gets sampled with independent probability, the probability that no path $P \odot_t v$ for $t \in S_v$ gets sampled is:

$$\begin{aligned}\prod_{t\in S_v}\prod_{P\odot_t v}(1 - w_t(P)K') \le \prod_{t\in S_v}\prod_{P\odot_t v} e^{-w_t(P)K'} &= e^{-\sum_{t\in S_v}\sum_{P\odot_t v} w_t(P)K'} \\ &\le e^{-\frac{1}{2}K'} \le \frac{1}{e^{\frac{1}{2}K_{smp}}\log T},\end{aligned} \tag{2.7}$$

where we used $1 + x \le e^x$ in the first inequality, and the penultimate inequality is found by inserting inequality (2.6). This proves the lemma. □

We use this to show we can delete a large fraction of the edges in Step 2.

Lemma 2.3.12. *For any day t and path P, each edge $e \in P$ becomes fully redundant after Step 1 with probability at least $\frac{3}{4}$, for K_{smp} a sufficiently large constant.*

Proof. An edge e is fully redundant unless it is not i-redundant for some $i = 0, 1, \ldots, \log T$. Now, e is not i-redundant if the last node v on $P[e]$ with $\ell(v) \leq i$ has not germinated. By Lemma 2.3.11 this happens with probability $\epsilon / \log T$. By applying a union bound we get that the probability that e is not i-redundant for at least one i, is at most

$$(\log T + 1)\epsilon / \log T \leq 2\epsilon.$$

By Lemma 2.3.11, we can make this expression arbitrarily small by making K_{smp} sufficiently large, which proves the lemma. □

Now, we are finally ready to bound the cost reduction, by showing that the cost of the path collection halves every main iteration of Procedure 2.3.5.

Lemma 2.3.13. *After Step 2, the expected cost of the new paths is*

$$E\left[\sum_t \sum_P w'_t(P)c(P)\right] \leq \frac{1}{2} \sum_t \sum_P w_t(P)c(P).$$

Proof. By Lemma 2.3.10, the cost of the paths doubles in Step 1. By Lemma 2.3.12 however, every edge on these paths is deleted with probability $\frac{3}{4}$ in Step 2. Hence, the expected cost of the paths after Step 2 is at most $\frac{1}{2}$ times the cost of the paths at the beginning of the iteration. □

Lemma 2.3.14. *When Procedure 2.3.5 returns, it outputs a collection of trees of expected cost*

$$\sum_t c(\mathcal{T}_t) + O(\log\log(T)) \sum_P w_t(P)c(P),$$

where $w, \mathcal{T}$ is the input solution.

Proof. Let $w^{(i)}, \mathcal{T}^{(i)}$ denote the FPS created in the ith iteration of Procedure 2.3.5, as it ends up after Step 2. Take $w^{(0)}, \mathcal{T}^{(0)}$ equal to $w, \mathcal{T}$.

Repeated application of Lemma 2.3.9 gives

$$E\left[\sum_t c(\mathcal{T}_t^{(i)})\right] = E\left[\sum_t c(\mathcal{T}_t) + \sum_{j \leq i} O(\log\log(T)) \sum_t \sum_P c(P) w_t^{(i)}(P)\right]. \quad (2.8)$$

While repeated application of Lemma 2.3.13 yields

$$E\left[\sum_t \sum_P c(P) w_t^{(i)}(P)\right] \leq \frac{1}{2^i} \sum_t \sum_P c(P) w_t(P). \quad (2.9)$$

Inserting (2.9) into (2.8), and working out the geometric series we find

$$E\left[\sum_t c(\mathcal{T}^{(i)})\right] = \sum_t c(\mathcal{T}_t) + O(\log\log(T)) \sum_P w_t(P)c(P), \quad (2.10)$$

for all i, which proves the lemma. □

2.3.3 *Main result*

We are now ready to prove our main results.[12]

[12] Recall Theorem 2.3.2: Given a fractional path solution of cost C_{FPS} we can find a randomized solution to the Steiner tree over time problem, of expected cost $O((\log\log T)C_{FPS})$.

Proof of Theorem 2.3.2. We show that the Procedure 2.3.5 describes an algorithm that satisfies the theorem.

Lemma 2.3.8 proves that Procedure 2.3.5 returns a feasible solution. While Lemma 2.3.14 proves that it costs no more than $O(\log\log(T)C_{FPS})$.

It remains to argue that we can implement the procedure in polynomial time. Let us first show that the expected number of iterations is polynomial. As a consequence of Lemma 2.3.11, each iteration any unconnected node gets connected to the root with high probability. Ergo, the number of unconnected nodes decrease by a constant fraction every iteration, and after $O(\log N)$ iterations, the expected number of uncovered nodes is $O(1)$.

During an actual run of the algorithm, at that point we can simply buy a unique connection to the root for every unconnected node, and terminate the algorithm. In expectation this will cost no more than a constant times the cost of connecting the most distant node to the root, which is a lower bound on OPT.

Furthermore, the inner steps of each iteration can be implemented in time polynomial in the number of nodes $|V|$ and the number of paths in the support of w_t summed over the entire time horizon. We just need to make sure that the path splitting in Step 2 does not cause the number of paths in the support of the fractional path solution to explode. This is not a problem though: since every path must contain at least one node, every path in the input fractional path solution can 'generate' at most $|V|$ new paths down the line, a polynomial increase. This completes the proof. $\square$

Theorem 2.3.15. *There is a polynomial time $O(\log\log T)$-approximation for the Steiner tree over time problem.*

Proof. As we argued at the beginning of this section, we can find an $O(1)$-approximately optimal fractional path solution efficiently, and therefore this theorem is immediately implied by Theorem 2.3.2. $\square$

Recall Theorem 2.1.1: There is a polynomial time $O(\log\log\min(N,T))$-approximation algorithm for the inventory routing problem.

Proof of Theorem 2.1.1. Follows from combining Theorem 2.2.16 with Theorem 2.3.15. $\square$

2.4 *Submodular cover over time*

Consider the following problem, *submodular cover over time*. We are given a discrete time horizon $[T] = \{1, \dots, T\}$, a set of nodes V and a monotone, non-negative, normalized, submodular set function $f : 2^V \to \mathbb{R}$. Furthermore, every node $v \in V$ has a time window $F_v = [s,t]$ for $s,t \in [T]$. We want to choose a set S_t for each day $t \in [T]$ such that every node $v \in V$ is contained in a set S_t with $t \in F_v$ in its time window. The objective is to minimize the sum of the cost of the chosen sets: $\sum_{t\in[T]} f(S)$.

In this section we will assume that the time windows in an instance lie in a left aligned family $\mathcal{F}$, which by Theorem 2.2.11 loses at most a constant factor.

2.4.1 *Lovasz-extension relaxation*

Our algorithm uses iterative rounding of a relaxation based on the Lovasz-extension of f.

Definition 2.4.1. The Lovasz-extension of a submodular set function $f : 2^V \to \mathbb{R}$ is the function $\hat{f} : [0,1]^V \to \mathbb{R}$ defined as:

$$\int_0^1 f(\{v : x_v \geq \theta\})\, d\theta.$$

Note that $\hat{f}$ corresponds exactly to f for integral inputs. It is well-known that the Lovász extension of submodular set function is convex [41, Sec. 6.3]. Hence the program given in (2.11) is a relaxation of the submodular cover over time problem and can be solved in polynomial time to any required precision.[13]

$$\begin{aligned} \text{minimize} \sum_{t \in [T]} \hat{f}(x^t) & \\ \text{subject to} \sum_{t \in F_v} x_v^t = 1 & \quad , \forall v \in V \\ x_v^t \geq 0 & \end{aligned} \tag{2.11}$$

Our main result in this section is the following.

Theorem 2.4.2. *There is a polynomial time algorithm that takes a solution to relaxation* (2.11) *of cost C_{LOV}, and produces a feasible solution to the submodular cover over time problem of cost* $\log\log(T) C_{LOV}$.

The proof of Theorem 2.4.2 is based on Procedure 2.4.4. Before we describe it in detail we will introduce some concepts and notation.

Concepts and notation Let $x \in [0,1]^V$ denote a vector indexed by V. For $\theta \in [0,1]$ we define the *level set*:

$$L_\theta(x) = \{v \in V : x_v \geq \theta\}.$$

With this notation we can conveniently express the Lovasz-extension as an expectation:

$$E_\theta[f(L_\theta(x))], \qquad \theta \sim \text{Uniform}(0,1).$$

This perspective has a nice graphical interpretation which will prove useful in understanding the proof of an important technical lemma later on, and the algorithm in general. If we plot $f(L_\theta(x))$, for θ in the interval $[0,1]$, then $\hat{f}(x)$ is the area between the graph and the horizontal and vertical axes. See Figure 2.5 for an example.

[13] However, the simplest way to find a solution to (2.11), is to convert a solution to LP (2.5), which can solved using the methods described in Section 2.5.1 of the appendix.
For f submodular, we can always uncross a solution such that the support of every day is a chain, which does not increase the cost due to the submodular inequality

$$f(S) + f(T) \geq f(S \cup T) + f(S \cap T).$$

But if the support of y is indeed a chain, then it is easy to verify that $\sum_S f(S) y^S$ is exactly equal to $\hat{f}(x)$ where $x_v = \sum_{S : v \in S} y^S$.

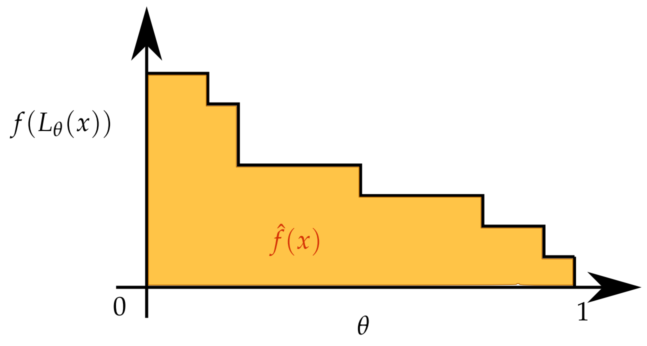


Figure 2.5: Graphical interpretation of the Lovász extension for a monotone, nonnegative, normalized, submodular set function f. We have θ on the horizontal axis, $f(L_\theta(x))$ on the vertical axis and $\hat{f}(x)$ is the area between the graph and axes.

We will often use an operation on x where we truncate all entries for elements in a given level set $L_\theta(x)$ at θ, so we introduce the dedicated notation

$x^{|\theta}$ for that. The entries of $x^{|\theta}$ are defined by:

$$x_v^{|\theta} = \begin{cases} x_v, & x_v < \theta \\ \theta, & x_v \geq \theta. \end{cases}$$

At this point we remark that the value of the Lovász extension on a truncated vector $x^{|\theta}$, can be found by evaluating the integral for x over a restricted range:

$$\hat{f}(x^{|\theta}) = \int_0^\theta f(L_\eta(x))\, d\eta.$$

Finally, we introduce the concept of an α-supported set in Definition 2.4.3. For now, think of these as level sets for which setting the corresponding entries to zero would yield a large reduction in the cost of the Lovász extension.

Definition 2.4.3. Given $\theta \in [0,1]$ and $x \in [0,1]^V$, the set $L_\theta(x)$ is α-supported if:

$$\hat{f}(x) - \hat{f}(x^{|\theta}) \geq \alpha f(L_\theta(x)). \tag{2.12}$$

We will generate more intuition for this definition later in this section, but first we will describe the algorithm.

2.4.2 *Algorithm details*

In the description of Procedure 2.4.4 α_{smp} denotes an appropriately small constant to be determined later.

Procedure 2.4.4 (Iterative rounding of Relaxation 2.11)**.** Let x be a solution to Relaxation (2.11).

1. Initialize a solution to the submodular cover over time problem:

$$S_t \leftarrow \emptyset \qquad \forall t \in [T].$$

2. For $i = 1, \dots, \log T$ repeat Steps 2a and 2b:

   (a) For each $t \in [T]$:

   - If there exists $\theta \in [0,1]$ such that $L_\theta(x^t)$ is $\frac{\alpha_{smp}}{\log\log T}$-supported, pick one such θ and set:

$$S_t \leftarrow S_t \cup L_\theta(x^t) \quad \text{and} \quad x^t \leftarrow x^{t|\theta}. \tag{2.13}$$

   - Otherwise:

$$S_t \leftarrow S_t \cup L_1(x^t). \tag{2.14}$$

   (b) Merge time periods by setting for all $t \in \{k2^i : k = 0, 1, 2, \dots\}$[14]:

$$x^{t+1} \leftarrow x^{t+1} + x^{t+2^{i-1}+1} \quad \text{and} \quad x^{t+2^{i-1}+1} \leftarrow 0. \tag{2.15}$$

3. Finally, return the solution $S_t, t \in [T]$.

[14] A visual illustration to help interpreting this formula is given in Figure 2.6.

We will first prove that Procedure 2.4.4 returns a feasible solution to the submodular cover over time problem, and then we will bound the cost.

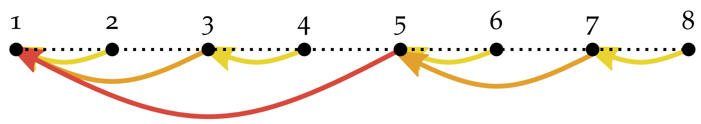


Figure 2.6: Illustration of time unit merging in Step 2b on time horizon $[T] = \{1, \dots, 8\}$ over 3 iterations. In the first iteration, even time units are merged into the preceding odd time units, in the second iteration, 3 and 7 are merged into 1 and 5 respectively, and finally 5 is merged into 1.

Feasibility At the start of the algorithm, the fractional solution covers each node v fully within its time window F_v. The only operation in the algorithm that can decrease v's coverage is given by (2.13) in Step 2a. But this always means v gets added to a set in the output solution, so v is alway fractionally covered in the relaxation or covered in the output solution.

The fractional coverage of v in the relaxation might get moved to a different time unit though, as a consequence of the merging in Step 2b. We will show that this does not cause problems. Firstly, all fractional coverage 'remains inside the time window' of v for the first $\ell(v)$ iterations. This is formalized in Lemma 2.4.5. Secondly, v is guaranteed to be covered in Step 2b of iteration $\ell(v) + 1$ at the latest, i.e., before any fractional coverage is moved outside v's time window. This is formalized in Lemma 2.4.6. Together these lemmas ensure feasibility.

Lemma 2.4.5. *Suppose a node v of level $\ell(v) = j$ has not been added to any set S_t for $t \in F_v$ at the beginning of some iteration $i \leq j + 1$. Then $\sum_{t \in F_v} x_v^t \geq 1$.*

Proof. By induction on i. It is true for $i = 1$ because the input solution is feasible. Now suppose it is true at the beginning of iteration $i \leq j$, we will show it holds for iteration $i + 1$ as well. If v does not get covered in iteration i, it is clear that $\sum_{t \in F_v} x_v^t$ remains unchanged after Step 2a.

So, we only need to ensure that the merging in Step 2b does not cause any problems. Suppose for contradiction that the assignment in Equation (2.15) moves some coverage from inside F_v to a day before F_v. Then there must be some integer k', such that

$$k' 2^i + 1 < \min F_v \quad \text{and} \quad k' 2^i + 2^{i-1} + 1 \in F_v.$$

But since the level of v is $\ell(v) = j \geq i$, we have that $\min F_v = k 2^j + 1$ for some integer k. This means the inequalities above cannot have a solution, a contradiction. Hence, we prove the induction and therefore the lemma. □

In principle some of the fractional coverage could get moved out of the time window by the merge step of iteration $\ell(v) + 1$. However, as the next lemma shows, we can guarantee that v gets covered before this happens. Intuitively, if v is not covered by the start of iteration $\ell(v) + 1$, all of the fractional coverage is merged into the left endpoint of its time window $t' = \min F_v$. This means that $L_\theta(x^{t'})$ contains v for any θ and ensuring v will be added to $S_{t'}$ in Step 2a.

Lemma 2.4.6. *Suppose a node v has not been added to any set S_t for $t \in F_v$ at the beginning of iteration $\ell(v) + 1$. Then v will be added to $S_{t'}$ where $t' = \min F_v$ in this iteration.*

Proof. By Lemma 2.4.5 $\sum_{t \in F_v} x_v^t \geq 1$. Moreover, by the end of iteration i, for every day t not equal to $k 2^i + 1$ for some k, x^t is set to zero. Since $\min F_v = t' = k 2^{\ell(v)} + 1$ and $\max F_v \leq (k+1) 2^{\ell(v)}$ for some k, all of v's coverage inside F_v must be at time t' by the end of iteration $\ell(v)$.

In other words, at the start of iteration $\ell(v)+1$, it holds that $x_v^{t'} \geq 1$. It follows that $v \in L_\theta(x^{t'})$ for any θ, so v is added to $S_{t'}$ regardless of how we determine the set for day t', finishing the proof. □

So, every node gets covered inside its time window. Lemma 2.4.7 is a direct consequence of Lemma 2.4.6 and stated without further proof.

Lemma 2.4.7. *Procedure 2.4.4 returns a feasible solution to the submodular cover over time problem.*

Cost analysis The cost analysis of the algorithm hinges on a result that roughly states the following. For each day t, at least one of two things must happen. Either we find an $\frac{\alpha}{\log\log T}$-supported set in Step 2a, in which case the reduction in the cost of the relaxation brought by removing $L_\theta(x^t)$ from the instance pays for an $O(1/\log\log T)$ fraction of its cost.

If not, the set $L_1(x^t)$ is guaranteed to be fairly cheap compared to the cost of the relaxation for that day (to be precise, $f(L_1(x^t))$ is $O(\hat{f}(x^t)/\log T)$), and can safely be paid for in every iteration without increasing the cost of the solution by more than a constant.

The following technical lemma provides the corner stone needed to prove this result.

Lemma 2.4.8. *Let $f : V \to \mathbb{R}$ be a monotone, nonnegative, normalized, submodular set function, $\hat{f}$ its Lovasz-extension and and $x \in [0,1]^V$. For any $\alpha \in (0,1]$ at least one of the following holds:*

1. *there exists $\theta \in [0,1]$ such that $L_\theta(x)$ is $\frac{\alpha}{32}$-supported.*
2. $2^{1/\alpha} f(L_1(x)) \leq \hat{f}(x)$

Before we formally prove this lemma, let us generate some graphical intuition. Recall that $L_\theta(x)$ is α-supported if

$$\alpha f(L_\theta(x)) \leq \hat{f}(x) - \hat{f}(x^{|\theta}).$$

In Figure 2.7, the plots of the Lovász extension on two different inputs are shown. On the left we have Case 1 of the lemma. If we pick the set $L_{\theta'}(x)$ as indicated, the cost is the area of the light blue square. However, the dark blue area containing $\hat{f}(x) - \hat{f}(x^{|\theta'})$ covers a large fraction of that cost. On the right, no such set exists. This implies the graph must decrease quite quickly everywhere and therefore lie entirely above some exponential curve. Consequently the cost of the set $L_1(x)$, which is the light green rectangle, is small compared to $\hat{f}(x)$, which is the area between the graph and the axes.

Proof of Lemma 2.4.8. Let $k \in \mathbb{N}$ be such that $\frac{1}{16}\alpha \leq \frac{1}{k} \leq \frac{1}{8}\alpha$. Note that this implies $k \geq 8$. Let $a = f(L_1(x))$.

Claim. *If $2^{1/\alpha} a > \hat{f}(x)$, there exists $m \in \mathbb{N}$ such that*

$$f(L_{\frac{k-m}{k}}(x)) < 2^m a. \tag{2.16}$$

Before we prove the claim, let's see how it proves the lemma. If property 2 holds we are done. So suppose it is not. Then the condition of the claim is true. Take the smallest m that satisfies (2.16), and let $\theta = \frac{k-m}{k}$. We claim that:

$$\frac{\alpha}{32} f(L_\theta(x)) \leq \hat{f}(x) - \hat{f}(x^{|\theta}).$$

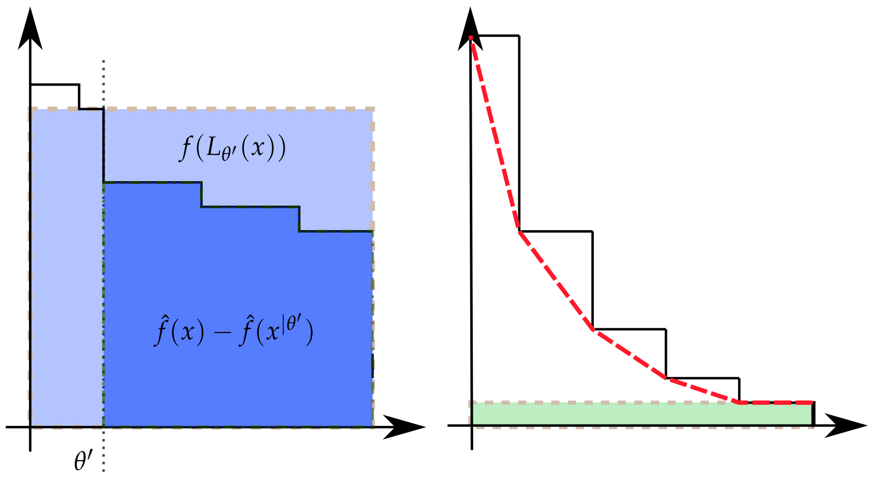


Figure 2.7: Illustration of Lemma 2.4.8. Either there is a set $L_{\theta'}$ whose cost gets covered for a large fraction by $\hat{f}(x) - \hat{f}(x^{|\theta'})$, or, the plot of $f(L_\theta(x))$ lies above some exponential curve and therefore $f(L_1(x))$ is small compared to $\hat{f}(x)$.

To see this we first rewrite the right hand side as an integral.

$$\hat{f}(x) - \hat{f}(x^{|\theta}) = \int_0^1 f(L_\eta(x))\, d\eta - \int_0^1 f(L_\eta(x^{|\theta}))\, d\eta = \tag{2.17}$$
$$= \int_0^1 f(L_\eta(x))\, d\eta - \int_0^\theta f(L_\eta(x))\, d\eta = \int_\theta^1 f(L_\eta(x))\, d\eta$$

Recall that $f(L_\eta(x))$ is monotonically decreasing and that $m \geq 1$ so that $\theta + \frac{1}{k} = \frac{k-m+1}{k} \leq 1$.

$$\int_\theta^1 f(L_\eta(x))\, d\eta \geq \int_\theta^{\theta+\frac{1}{k}} f(L_\eta(x))\, d\eta \geq \frac{1}{k} f(L_{\theta+\frac{1}{k}}(x)) \tag{2.18}$$

Finally, we use the fact that m is minimal, which implies that $f(L_{\frac{k-m+1}{k}}(x)) \geq 2^{m-1}a$, together with (2.17) and (2.18):

$$\hat{f}(x) - \hat{f}(x^{|\theta}) \geq \frac{1}{k} 2^{m-1} a = \frac{1}{2k} 2^m a > \frac{\alpha}{32} f(L_\theta(x)). \tag{2.19}$$

In the final inequality of (2.19) we use that the fact that we chose m to satisfy $2^m a > f(L_{\frac{k-m}{k}}(x))$ and $\frac{1}{k} \geq \frac{1}{16}\alpha$.

Now we proceed to prove our claim. Suppose for contradiction that the condition of the claim holds but no m satisfies inequality (2.16). Then, in particular it must hold that $f(L_{\frac{1}{k}}(x)) \geq 2^{k-1}a$ and therefore we get that:

$$\hat{f}(x) \geq \int_0^{\frac{1}{k}} f(L_\eta(x))\, d\eta \geq \frac{1}{k} f(L_{\frac{1}{k}}(x)) \geq \frac{1}{k} 2^{k-1} a. \tag{2.20}$$

Using the fact that $k \geq 8$ we get that

$$\frac{1}{k} 2^{k-1} a = 2^{\frac{k}{2}} a (\frac{1}{k} 2^{\frac{k}{2}-1}) \geq 2^{\frac{k}{2}} a (\frac{1}{8} 2^{8/2-1}) = 2^{\frac{k}{2}} a \tag{2.21}$$

Concatenating the previous two results and inserting $\frac{1}{k} \leq \frac{1}{8}\alpha$, we get:

$$\hat{f}(x) \geq 2^{k/2} a \geq 2^{4/\alpha} a \geq 2^{1/\alpha} a,$$

contradicting that $2^{1/\alpha} a > \hat{f}(x)$. This proves the claim, and hence the lemma.

□

Lemma 2.4.8 allows us to derive the next result.

Lemma 2.4.9. *For α_{smp} a sufficiently small constant, Procedure 2.4.4 satisfies the following.*

If during Step 2a for day $t \in [T]$, no θ exists such $L_\theta(x^t)$ is $\frac{\alpha_{smp}}{\log\log T}$-supported, then the cost of the set $f(L_1(x^t)) = O(\hat{f}(x^t)/\log T)$.

Proof. We prove the contrapositive. Take $\alpha = \frac{1}{\log\log T}$. Suppose x^t satisfies Case 2 of Lemma 2.4.8 for α, i.e. $2^{1/\alpha} f(L_1(x^t)) \geq \hat{f}(x^t)$. This implies:

$$f(L_1(x^t)) \geq \hat{f}(x^t)/2^{\log\log T} = \hat{f}(x^t)/\log T,$$

and we are done.

So suppose not, then Case 1 of Lemma 2.4.8 holds, and there must be some θ for which $L_\theta(x^t)$ is $\frac{\alpha}{32}$-supported. It follows that if we set the constant such that $\alpha_{smp} \leq \frac{1}{32\log\log T}$, we are guaranteed to find an $\frac{\alpha_{smp}}{\log\log T}$-supported set in Step 2a, proving the lemma. □

2.4.3 Main result

We are now ready to prove the main results. [15]

[15] Recall Theorem 2.4.2: There is a polynomial time algorithm that takes a solution to relaxation (2.11) of cost C_{LOV}, and produces a feasible solution to the submodular cover over time problem of cost $\log\log(T)C_{LOV}$.

Proof of Theorem 2.4.2. We claim that the Procedure 2.4.4 describes an algorithm that proves the theorem.

Lemma 2.4.7 proves that Procedure 2.4.4 returns a feasible solution. The total cost of the $\frac{\alpha_{smp}}{\log\log T}$-supported sets selected in Step 2a is $O(\log\log T)$ times the cost of the input solution to the relaxation C_{LOV}, as for every unit we pay for these sets, the cost of the relaxation gets reduced by a $O(1/\log\log T)$ fraction. Lemma 2.4.9 ensures that whenever we do not select such a set, the cost of the set $L_1(x^t)$ is at most $O(\hat{f}(x^t)/\log T)$. So, summing over all days and iterations the cost of these sets does not exceed $O(\sum_t \hat{f}(x^t)) = O(C_{LOV})$. It follows that the total cost of the solution produced by Procedure 2.4.4 is $O(\log\log(T)C_{LOV})$ as required.

It remains to argue that we can implement the procedure in polynomial time. The number of iterations is $O(\log T)$ and all of the inner steps require us to loop over at most T days. Since all other operations are trivial assignments, we only need to worry about the finding an $\frac{\alpha_{smp}}{\log\log T}$-supported set in Step 2a. There are at most N unique values of θ at which $L_\theta(x^t)$ changes, and we only need to check these. We conclude that this step can also be implemented in polynomial time, which proves the claim and the theorem. □

Theorem 2.4.10. *There is a polynomial time $O(\log\log T)$-approximation for the submodular cover over time problem.*

Proof. As we argued at the beginning of this section, we can solve Relaxation 2.11 to any required precision in polynomial time, and therefore this theorem is implied by Theorem 2.3.2. □

Recall Theorem 2.1.2 There is a polynomial time $O(\log\log\min(N,T))$-approximation algorithm for the submodular joint replenishment problem.

Proof of Theorem 2.1.2. Follows from combining Theorem 2.2.16 with Theorem 2.4.10. □

2.5 Appendix

2.5.1 Approximately solving the subadditive cover over time LP.

In this section we describe how we can (approximately) solve LP (2.5) when appropriate conditions are met.

Recall that LP (2.5) is the following LP.

$$\begin{aligned} \text{minimize} \quad & \sum_{t\in[T]} \sum_{S\subseteq[N]} f(S) y_t^S \\ \text{subject to} \quad & \sum_{r\in[s,t]} \sum_{S:i\in S} y_r^S \geq 1 \qquad \forall [s,t] \in W_i, \quad \forall i \in [N] \\ & y_t^S \geq 0 \qquad \forall t, S \end{aligned} \tag{(2.5)}$$

Its dual is given by (2.22).

$$\begin{aligned} \text{maximize} \quad & \sum_{i\in[N]} \sum_{[s,t]\in W_i} a_{s,t}^i \\ \text{subject to} \quad & \sum_{i\in S} \sum_{\substack{[s,t]\in W_i: \\ r\in[s,t]}} a_{st}^i \leq f(S) \qquad \forall S \subseteq [N], r \in [T] \\ & a_{st}^i \geq 0 \qquad \forall i \in [N], [s,t] \in W_i \end{aligned} \tag{2.22}$$

Since the dual has a polynomial number of variables, we only need an efficient way of finding violated inequalities and apply the ellipsoid method [64] to solve it in polynomial time. We can treat the constraints in (2.22) for each value of r separately. Fix one such r and write

$$a^i := \sum_{\substack{[s,t]\in W_i: \\ r\in[s,t]}} a_{s,t}^i$$

to simplify notation. The generic problem then becomes finding a set S such that $\sum_{i\in S} a^i > f(S)$.

If we write $a(S) = \sum_{i\in S} a^i$, this simplifies to finding a set S such that

$$f(S) - a(S) < 0. \tag{2.23}$$

Now suppose f is a submodular function. As a is a modular function, $f - a$ is also submodular function. Minimizing a submodular function can be done in strongly polynomial time [64], so we can find a violated inequality in polynomial time.

When f is routing cost this is not the case, however. The separation problem amounts to a prize collecting Steiner tree or TSP, depending on the routing cost formulation. All these problems are APX-hard, though a constant factor approximation with performance guarantee (just) smaller than 2 is known [5]. We need a stronger type of approximation though, called a *lagrangian multiplier preserving* (LMP) approximation algorithm, which we will now introduce in general terms.

Definition 2.5.1. Given a ground set $[N]$, a set function $f : 2^{[N]} \to \mathbb{R}_+$ and penalties a^i for all $i \in [N]$, the prize collecting problem asks to find a set $S \subseteq [N]$ that minimizes:

$$f(S) + a([N] \setminus S).$$

Intuitively, an LMP algorithm constructs a solution S that may lose some factor on the cost of the set $f(S)$ compared to OPT, but nothing on the penalties.

Definition 2.5.2. A Lagrangian multiplier preserving α-approximation algorithm for the prize collecting problem is an algorithm which outputs a solution such that:

$$\frac{1}{\alpha}f(S) + a([N] \setminus S) \leq OPT.$$

The reason we need this property is that to convert our separation problem to a prize collecting problem, we add a constant $a([N])$ to the objective function. The separation problem now becomes determining if the minimum of (2.24) is smaller than $a([N])$.This does not affect the minimizer, but it does affect the minimum and therefore the approximation ratio.

$$\min f(S) - a(S) + a([N]) = \min f(S) + a([N] \setminus S) \tag{2.24}$$

A Lagrangian multiplier preserving 2-approximation for prize collecting Steiner tree and TSP can be found using Goemans and Williamson's celebrated primal dual algorithm [45].

From LMP to approximately solving the LP Suppose we have an LMP α-approximation for a class of functions f[16]. We can use it find an α-approximate solution to LP (2.5) as follows.

[16] Assume that this class is closed under multiplication by a constant factor, which is no real restriction in our context.

Let $\tilde{f} = \alpha f$, let D be the dual polytope under cost function f, and $\tilde{D}$ the dual polytope if we replace the cost function by $\tilde{f}$. Clearly $OPT_{\tilde{D}} = \alpha OPT_D$.

Initially we run the ellipsoid method to maximize over D, using the LMP algorithm to generate violated inequalities as long as we can find any. Let a' be the solution found when no more violated inequalities can be found. We claim that:

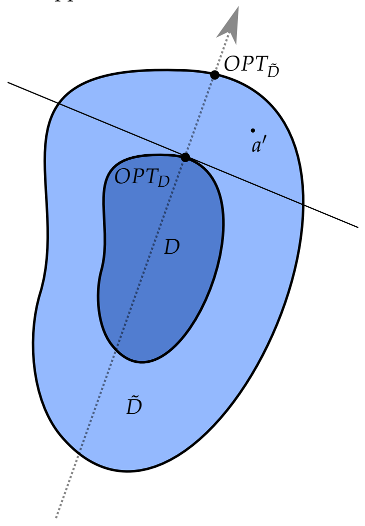
1. $a' \in \tilde{D}$, and

2. the value of a' is at least OPT_D.

This implies that the value of a' is at most αOPT_D and therefore an α-approximation of the optimum of LP (2.5). Finally, we find a feasible point to the primal of equal value: restrict the solution to variables corresponding to constraints active at a', of which there are polynomially many.

To finish the proof we prove the claims.

Figure 2.8: Illustration of ellipsoid method with approximate separation oracle. The polytope $\tilde{D}$ is essentially D blown up by a factor α, so D is entirely contained in $\tilde{D}$. As long as a is outside $\tilde{D}$, we generate a violated inequality that is valid for D. If no more violated inequalities are found, we know a' is 'in between' D and $\tilde{D}$ and therefore an α-approximation of OPT_D.



Claim 1. Suppose $a' \notin \tilde{D}$. We claim the LMP algorithm finds a violated inequality for D, which means the ellipsoid run does not terminate on a', a contradiction.

Note $a' \notin \tilde{D}$ implies there exists S such that:

$$\begin{aligned} \tilde{f}(S) - a'(S) &< 0 \\ \tilde{f}(S) + a'([N] \setminus S) &< a'([N]) \end{aligned} \tag{2.25}$$

The LMP algorithm is guaranteed to find a set S' such that:

$$\frac{1}{\alpha}\tilde{f}(S') + a'([N] \setminus S') \leq \tilde{f}(S) + a'([N] \setminus S) \tag{2.26}$$

Inserting the definition of $\tilde{f}$ and (2.25) gives

$$f(S') + a'([N] \setminus S') < a'([N]) \tag{2.27}$$

hence this set S' defines a violated inequality for $\mathcal{D}$ as required. $\square$

Claim 2. This follows from the fact that every violated inequality generated by the LMP procedure is a valid inequality. Hence, the constraints set generated by the LMP procedure forms a relaxation of D and the bound follows. $\square$

3
Replenishment problems with fixed turnover times

Parts of this chapter appeared in:
[16] Thomas Bosman, Martijn van Ee, Jiao Yang, Alberto Marchetti-Spaccamela, R. Ravi and Leen Stougie. "Approximation Algorithms for Replenishment Problems with Fixed Turnover Times". *Latin American Symposium on Theoretical Informatics (LATIN)*, 2018.

Abstract We introduce and study a class of optimization problems we coin replenishment problems with fixed turnover times: a very natural model that has received little attention in the literature. Nodes with capacity for storing a certain commodity are located at various places; at each node the commodity depletes within a certain time, the turnover time, which is constant but can vary between locations. Nodes should never run empty, and to prevent this we may schedule nodes for replenishment every day. The natural feature that makes this problem interesting is that we may schedule a replenishment (well) before a node becomes empty, but then the next replenishment will be due earlier also. This added workload needs to be balanced against the cost of routing vehicles to do the replenishments. In this chapter, we focus on the aspect of minimizing routing costs. However, the framework of recurring tasks, in which the next job of a task must be done within a fixed amount of time after the previous one is much more general and gives an adequate model for many practical situations.

Note that our problem has an infinite time horizon. However, it can be fully characterized by a compact input, containing only the location of each node and a turnover time. This makes determining its computational complexity highly challenging and indeed it remains essentially unresolved. We study the problem for two objectives: MIN-AVG minimizes the average tour length and MIN-MAX minimizes the maximum tour length over all days. For MIN-MAX we derive a logarithmic factor approximation for the problem on general metrics and a 6-approximation for the problem on trees, for which we have a proof of NP-hardness. For MIN-AVG we present a $O(\log \log n)$-approximation on general metrics by reducing it to the inventory routing problem, a 2-approximation for trees, and a pseudopolynomial time algorithm for the line. Many intriguing problems remain open.

3.1 Introduction

Imagine the following particular inventory replenishment problem. A set of automatic vendor machines are spread over a country or a city. They have a certain turnover time: the number of days in which a full machine will be sold out. Replenishment is done by vehicles. Let us assume that turnover times are machine dependent but not time dependent, and that it is highly undesirable to have an empty machine. However, the holding costs of the machine are negligible, so that we will always fill the machine to capacity. There is nothing against replenishing a machine before it has become empty, but then the next replenishment will be due earlier as well. That is, the deadline of the next replenishment is always within the turnover time after the last replenishment. Equivalently, in any consecutive number of days equal to the turnover time, at least one replenishment has to take place. Replenishing a machine earlier to combine it with the replenishment of another machine that is due earlier may lead to cost savings. The feature that makes this problem so special w.r.t. existing literature, is that it can be compactly modeled by only specifying for every machine its location and the turnover time. The feature is very natural but has hardly been studied in the existing literature. There are intriguing basic open complexity questions, and some highly non-trivial results.

The motivation for studying this problem comes linea recta from a business project for the replenishment of ATMs in the Netherlands, in which some of the co-authors are involved. Of course the real-life ATM replenishment problem is not as stylized as described above; the turnover time is not strictly the same over time but subject to variability, there are restrictions on the routes for the vehicles, etc. But the feature that is least understood in the ATM-problem is exactly the problem of how to deal with the trade-off between replenishing an ATM earlier than its due date leading to a higher frequency of replenishments and the savings on vehicle routing costs.

Formally, an instance of the problem that we study in this chapter, which we baptize the REPLENISHMENT PROBLEM WITH FIXED TURNOVER TIMES (RFTT), consists of a pair (G, τ), where $G = (V \cup \{s\}, E, c)$ is a weighted graph with a designated depot vertex s and weights on the edges $c : E \to \mathbb{R}_+$, and turnover times $\tau \in \mathbb{N}^{|V|}$, indicating that $v_j \in V$ should be visited at least once in every interval of τ_j days.

A solution consists, for each day k, of a tour T_k in G starting in and returning to the depot s and visiting a subset of the vertices $J_k \subseteq V$. It is feasible if $v_j \in \bigcup_{k=t+1}^{t+\tau_j} J_k$, $\forall t$ and $\forall v_j \in V$. We will focus on solutions that repeat themselves after a finite amount of time, that is, in which $(T_k, ..., T_{k+\ell}) = (T_{k+\ell+1}, ..., T_{k+2\ell})$ for some ℓ, and all k. Since all turnover times are finite, this is no real restriction.

We consider two versions of RFTT. In the first version, called MIN-AVG, the goal is to find a feasible solution that minimizes the average tour length. In the MIN-MAX problem, we want to find a feasible solution that minimizes the maximum tour length over all days.

Related work. Our problem is strongly related to the INVENTORY ROUTING PROBLEM (IRP) from Chapter 2. In particular, the MIN-AVG objective can be seen as periodic variant of the IRP, and we will make use of this relation later on. However, an important difference between the IRP and RFTT problems, is that the latter has a compact input description — consisting only of a location and turnover time — which makes it incomparable to the inventory routing problem from a complexity point of view. Indeed it is unclear if the decision version of our problem is in NP or in co-NP.

Another closely related problem is the PERIODIC LATENCY PROBLEM [26], which features the recurring visits requirement of RFTT. We are given recurrence length q_i for each client i and travel distances between clients. Client i is considered *served* if it is visited every q_i time units. The difference is that the server does not return to the depot after each tour, but keeps moving continuously between clients at uniform speed. Coene et. al. [26] study two versions of the problem: one that maximizes the number of served clients by one server, and one that minimizes the number of servers needed to serve all clients. They determine the complexity of these problems on lines, circles, stars, trees, and general metrics.

Yet another problem that shares the compact input size and is in fact a very special case of our problem is known under the guise of PINWHEEL SCHEDULING (PS). It has been introduced to model the scheduling of a ground station to receive information from a set of satellites without data loss. In terms of our problem no more than one node can be replenished per day and all distances to the depot are the same; the interesting question here is if there exists a feasible schedule for replenishing the vertices. Formally, a set of jobs $\{1, ..., n\}$ with periods $p_1, ..., p_n$ is given, and the question is whether there exists a schedule $\sigma : \mathbb{N} \to \{1, .., n\}$ such that $j \in \bigcup_{k=t+1}^{t+p_j} \sigma_k$, $\forall t \geq 0$ and $\forall j$.

PINWHEEL SCHEDULING was introduced by Holte et al. [54], who showed that it is contained in PSPACE. The problem is in NP if the schedule σ is restricted to one in which for each job the time between two consecutive executions remains constant throughout the schedule. In particular this must hold for feasible schedules if the instance has *density* $\rho = \sum_j 1/p_j = 1$ [54]. They also observed that the problem is easily solvable when $\rho \leq 1$ and the periods are harmonic, i.e. p_i is a divisor of p_j or vice versa for all i and j. As a corollary, every instance with $\rho \leq \frac{1}{2}$ is feasible: simply round turn over times down to powers of 2.

Chan and Chin [21] improved the latter result by giving an algorithm that produces a feasible schedule for PS whenever $\rho \leq \frac{2}{3}$. In [20], they improved this factor to $\frac{7}{10}$. Later, Fishburn and Lagarias [38] showed that every instance with $\rho \leq \frac{3}{4}$ has a feasible schedule. All these chapters work towards the conjecture that there is a feasible schedule if $\rho \leq \frac{5}{6}$. That no sharper bound is possible, can be seen by the instance with $p_1 = 2$, $p_2 = 3$ and $p_3 = M$, with M large. This instance has density $\frac{5}{6} + \frac{1}{M}$, but cannot be scheduled: every other day we need to schedule job 1, but then job 2 also needs to be scheduled every other day (every fourth day is no option) which means no free day remains for job 3.

The complexity of PINWHEEL SCHEDULING has been open since it was introduced. It was only recently shown by Jacobs and Longo [57] that there is no pseudopolynomial time algorithm solving the problem unless SAT has an exact algorithm running in expected time $n^{O(\log n \log \log n)}$, implying for example that the randomized exponential time hypothesis fails to hold [19, 27]. Since the latter is unlikely, one could conclude that PINWHEEL SCHEDULING is not solvable in pseudopolynomial time. It remains open whether the problem is PSPACE-complete.

Our problem shares the property that a constant input size can specify an infinite sequence of jobs with the PERIODIC MAINTENANCE PROBLEM [10, 31]. It was shown that even in the case of a single server and $c_j = 1$ for all j the problem remains coNP-hard [8]. Other related problems with a compact input representation include real-time scheduling of sporadic tasks [9, 13], where we are given a set of recurrent tasks. On a single machine, EDF (Earliest Deadline First) is optimal. However, we remark that the complexity of deciding whether a given set of tasks is feasible has been open for a long time and only recently proved that it is coNP-hard to decide whether a task system is feasible on a single processor even if the utilization is bounded [33].

This chapter. We investigate the computational complexity of both the MIN-MAX and the MIN-AVG version of RFTT. Mostly we will relate their complexity to the complexity of PINWHEEL SCHEDULING. Some interesting inapproximability results follow from this relation. After that, we will start with some special cases. In Section 3.3, we give our most interesting result, a constant factor approximation for MIN-MAX on a tree, next to a less remarkable constant approximation for the MIN-AVG version on the tree. In the same section, we show for MIN-AVG that the problem can be solved to optimality in pseudopolynomial time on line metrics. Finally, in Section 3.4, we present a logarithmic factor approximation for the MIN-MAX objective and $O(\log\log n)$-approximation for MIN-AVG by reduction to the IRP problem.

3.2 *Complexity*

In this section, we investigate the computational complexity for both objectives. Since our problem requires finding a shortest tour visiting some subset of vertices for every day, it is at least as hard as the TRAVELING SALESMAN PROBLEM (TSP). However, it is also interesting to note that the problems are at least as hard as PINWHEEL SCHEDULING. This is an additional 'source of hardness', independent of the embedded TSP problem, as it holds on graph classes where TSP is not hard.

For the MIN-MAX objective there is a straightforward gap reduction from PS to the RFTT on star graphs. Create an unweighted star graph with the depot at the center and a leaf for each job in the PS instance with turnover time equal to the job's period. This instance has value 2 only if there exists a pinwheel schedule and at least 4 otherwise.

Theorem 3.2.1. *Approximating* MIN-MAX RFTT *within a factor* 2 *on star graphs, is at least a hard as* PINWHEEL SCHEDULING.

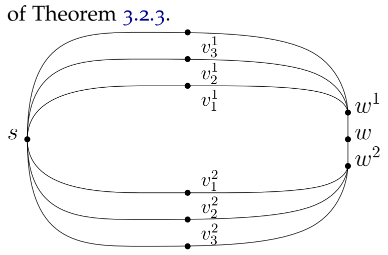
A similar reduction works to show strong NP-hardness. Now we reduce from a 3-PARTITION[1] instance $a_1, ..., a_{3m}$. We construct a weighted star graph with the depot at the center. For each element a_i create a leaf vertex attached by an edge of length $\frac{1}{2}a_i$. The turnover time of every leaf is set to m. A valid 3-PARTITION then corresponds to an RFTT schedule of cost $\sum_{i=1}^{3m} a_i/m$ and vice versa.

[1] In 3-PARTITION, we are given $3m$ integers $a_1, \dots, a_{3m}$ and an integer $B = \frac{1}{m}\sum_i a_i$ such that $\frac{B}{4} < a_i < \frac{B}{2}$ for all i. The question is whether we can partition the integers into m sets of three integers that add up to B [43].

Theorem 3.2.2. MIN-MAX RFTT *is NP-hard on star graphs.*

For the MIN-AVG RFTT the reduction is a bit more involved.

Figure 3.1: Instance created in the proof of Theorem 3.2.3.



Theorem 3.2.3. *On series-parallel graphs,* MIN-AVG RFTT *is at least as hard as* PINWHEEL SCHEDULING.

Proof. Given an instance $p_1, ..., p_n$ of PINWHEEL SCHEDULING, create an instance (G, τ) of MIN-AVG RFTT. We define

$$V = \{s, w^1, w, w^2\} \cup V^1 \cup V^2,$$

where $V^i = \{v_1^i, ..., v_n^i\}$ for $i = 1, 2$,

$$E = \{(s,v) | v \in V^1 \cup V^2\} \cup \{(w^i, v_j^i) | \forall j; i = 1,2\} \cup \{(w^1, w), (w^2, w)\},$$

and $\tau_{w^1} = \tau_{w^2} = \tau_w = 1$, $\tau_{v_j^1} = \tau_{v_j^2} = p_j$. All edge weights are 1. See Figure 3.1 for an illustration.

We claim that the instance (G, τ) has a solution of cost 6 if and only if $p_1, ..., p_n$ is a feasible Pinwheel Scheduling instance.

For the 'if'-direction, suppose we have a feasible pinwheel schedule. Assume that on every day some job is scheduled in the pinwheel schedule (otherwise add arbitrary jobs to the empty timeslots). Now, we create a replenishment schedule as follows: on day k we visit the set of vertices $J_k = \{v_j^1, w^1, w, w^2, v_j^2\}$, where j is the job scheduled on day k in the pinwheel solution. The pinwheel schedule then guarantees that the periods of jobs in $V^1 \cup V^2$ are satisfied, while jobs w^1, w, w^2 are visited every day, as required. Now since any tour $(s, v_j^1, w^1, w, w^2, v_j^2, s)$ has length 6, we can do this within the claimed cost.

For the 'only if'-direction, note that any replenishment schedule must have cost 6 at least, since no tour that visits w^1, w, w^2 costs less than that. Moreover, any tour visiting those three vertices that is not of the form $(s, v_j^1, w^1, w, w^2, v_j^2, s)$, will cost strictly more than 6. It follows that if the cost of the replenishment schedule is 6, every tour visits at most one job from V^1 (and the same for V^2). Since $\tau_{v_j^1} = p_j$ for all j, this directly implies that the Pinwheel Scheduling instance is feasible.[2] □

[2] It should be noted that an optimal schedule might visit vertices $v_j^1, v_{j'}^2$ on the same day for $j \neq j'$. In this case the v_j^1-nodes and the v_j^2-nodes induce different feasible pinwheel schedules.

3.3 *Approximation on trees*

In this section we give a 2-approximation for min-avg and a 6-approximation for min-max on trees.

We start out with a simplifying result, which will be of use in the next sections.

Lemma 3.3.1. *Given an instance (G, τ) of* rftt, *let τ' be found by rounding every turnover time in τ down to a power of 2. Then $OPT(G, \tau') \leq 2OPT(G, \tau)$.*

Proof. Let $(G, \bar{\tau})$ denote the instance found from (G, τ) by rounding every turnover time up to a power of 2. Since any schedule remains feasible if we round up the turnover times, we have that $OPT(G, \bar{\tau}) \leq OPT(G, \tau) \leq OPT(G, \tau')$.

Suppose we have an optimal solution for $(G, \bar{\tau})$ in which $\bar{T}_k$ is scheduled on day k. We can construct a feasible schedule for (G, τ') by scheduling the concatenation of $\bar{T}_{2k-1}$ and $\bar{T}_{2k}$ on day k. The maximum tour length in this schedule is at most twice that of the optimal solution for $(G, \bar{\tau})$ and every tour from the original schedule is visited exactly twice in the new schedule, so this yields a factor 2 increase in both the min-max and the min-avg objective. □

In the remainder we assume w.l.o.g. that G is rooted at s and that turnover times are increasing on any path from the depot to a leaf node in G.[3] Furthermore, for an edge e in E we define $D(e)$ to be the set of vertices that are a descendant of e. We also need the following definition.

[3] If some node v_j, on the path from root s to another node v_k, has $\tau_j > \tau_j$, we visit v_j sufficiently often as long as we visit v_k sufficiently often, so we can remove v_j from the instance.

Definition 3.3.2 (tt-weight of an edge)**.** For any edge in G we define:

$$q(e) = \min_{j \in D(e)} \tau_j.$$

We call this quantity the tt-weight (turnover time-weight) of e.

So, under the assumption we make on G, $q(e)$ is equal to the turnover time of the incident vertex of e furthest from the root. This definition allows us to express the following lower bound.

Lemma 3.3.3 (tt-weighted tree)**.** *For an instance (G, τ) of the* RFTT *on trees it holds that the average tour length is at least:*

$$L(G, \tau) := 2 \sum_{e \in E} \frac{c(e)}{q(e)}.$$

Proof. This follows immediately from the fact that $\frac{2}{q(e)}$ lower bounds the number of times edge e must be traversed on *average* in any feasible solution. □

Since the maximum tour length is at least the average tour length, Lemma 3.3.3 also provides a lower bound for the MIN-MAX objective.

An approximation for MIN-AVG RFTT is thus found by rounding all turnover times to powers of 2 and then visit each client j on every day that is a multiple of τ_j. Since in that case the lower bound of Lemma 3.3.3 is exactly attained on the rounded instance, Lemma 3.3.1 implies the following theorem.

Theorem 3.3.4. *There is a 2-approximation for* MIN-AVG RFTT *on trees.*

3.3.1 MIN-MAX

We will now show that we can achieve a 6-approximation for MIN-MAX RFTT on trees. The main ingredient is a 3-approximation algorithm for the case that all turnover times are powers of 2, the general case then follows by applying Lemma 3.3.1.

The algorithm produces a solution by recursively fixing the schedules of nodes with increasing turnover times. At the top level, it fixes all nodes j with $\tau_j = 1$, who need to be visited daily. It then splits the remaining nodes into two new instances, each containing only clients with turnover time at least 2.

One instance will use only odd days and the other only even days. So, in each instance we can shrink the timeline by deleting every other day and end up with an instance where the minimum turnover time is again 1. We then apply the same trick recursively to each of the new instances.

We want the recursive splitting of the subtrees to distribute the clients evenly over the days, i.e. each day must get a subtree of approximately equal weight. To guide this we use a TSP tour in the tree we compute prior to the start of the algorithm. This tour is split in each recursive step, and each part passed on to one of the child calls of the algorithm. The area covered by the part of the tour a child receives determines which clients are in its instance. Since every child receives a connected part of the tour, this area will form a single connected component, implying it is sensible to visit nodes there simultaneously.

The most important aspect of making this work is to ensure that the tour is split in a balanced way. In particular, we want its length to be balanced, taking into account the tt-weights. Before we describe more precisely what this means, we will introduce some formal definitions to avoid confusion later on.

Preliminaries In the context of this section, a walk $W = (v_1, e_1, v_2, \dots, v_k)$ is a sequence of vertices interspersed by edges in G, such that each consecutive pair of vertices is connected by the edge occurring between them in the walk. A TSP tour is a walk that contains every vertex in G and starts and ends with the root. We use $v \in W$ or $e \in W$ to denote that a vertex or edge occurs (at least once) in the walk.

Definition 3.3.5. Let W be a walk in the tree G. A split of W is a pair of walks $W_{pre} = (v_1, e_2, v_2, \dots, v_j)$ and $W_{suf} = (v_{j+1}, e_{j+1}, v_{j+2}, e_{j+2}, \dots, v_k)$, that partition W, i.e.:

$$(v_1, \dots, v_j, e_j, v_{j+1}, \dots, v_k) = W.$$

We call these walks the prefix and the suffix respectively.[4]

[4] Note that the edge connecting the end and start of the walks in the split is not part of either of the walks.

Definition 3.3.6. A split is *tt-balanced* if:

$$\sum_{e \in W_{pre}} c(e)/q(e) \leq \frac{1}{2} \sum_{e \in W} c(e)/q(e) \quad \text{and} \quad \sum_{e \in W_{suf}} c(e)/q(e) \leq \frac{1}{2} \sum_{e \in W} c(e)/q(e).$$

Since the tt-weights are a measure of how often an edge is visited, a balanced split essentially partitions the work equally between the two parts.

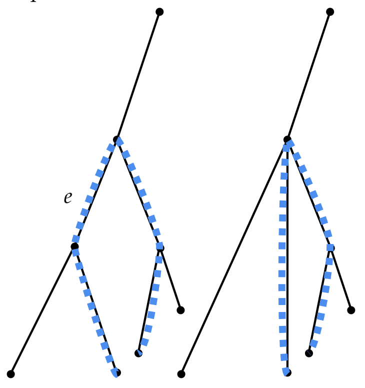
We also need the concept of edge contraction.

Definition 3.3.7. Let $e \in G$ be an edge, with endpoints u, v. The operation of contracting e in G, merges u and v into a single vertex, that is adjacent to every vertex that was adjacent to u or v.

We denote the contracted graph M/e. If F is an edge set in G, M/F denotes the graph found by contracting all edges in F (in any order). Similarly if W is a walk in G, we denote W/F the walk in the contracted graph G/F.

Uncontraction is simply the operation that reverses a previously executed edge contraction.

Figure 3.2: Illustation of contracting edge e. Note that we only allow contraction of adjacent vertices. This means that for every walk W and edge e in G, there is an obvious mapping to a walk in G/e, as shown by the blue dotted path here.



In each recursive step, the algorithm receives a walk W from its parent. It looks at the unscheduled clients in the walk, schedules every client j for which $\tau_j = 1$ to be visited daily, and removes them from the instance. Next, it contracts edges in G with $q(e) = 1$. Finally it adjusts all turnover times and tt-weights by dividing them by 2. It now computes a tt-balanced split W_0, W_1 of W', where W' is the walk in the contracted metric, and passes each one to a separate recursive call of the algorithm.

Each child call returns a schedule that visits nodes on a daily basis. These schedules get 'stretched' out and added to the odd and even days respectively. In other words, all clients scheduled at day t by the first child get scheduled at day $2t - 1$, and all clients scheduled at day t by the second child get scheduled at day $2t$.

Procedure 3.3.8 provides a detailed description. Note that it only constructs a schedule for the first τ_{max} days; after day τ_{max} the schedule is continuously is repeated.

Procedure 3.3.8 (Approximation algorithm for MIN-MAX on trees).
Input: An instance (G, τ) with τ powers of two
Output: schedule $S_1, \dots, S_{\tau_{max}}$ for the first τ_{max} days

1. Compute a TSP tour W in G, the tt-weights q and set $U \leftarrow V$ the set of clients in G.

2. Return the schedule generated by MAXTREESCHEDULE(G, U, W, τ, q)

MAXTREESCHEDULE
Input: graph G, uncovered clients U, walk W, turnover times τ, tt-weights q
Output: schedule $S_1, \dots, S_{\tau_{max}}$ for the first τ_{max} days

1. Determine the set of clients with turnover time 1 and edges connecting them
$$X \leftarrow \{j \in U : j \in W \text{ and } \tau_j = 1\} \quad \text{and} \quad Y \leftarrow \{e \in W : q(e) = 1\}.$$

2. Initialize the schedule
$$S_t \leftarrow X \text{ for } t = 1, \dots, \tau_{max}.$$

3. Remove the covered clients and contract the edges connecting them
$$U' \leftarrow U \setminus X \quad \text{and} \quad G' \leftarrow G/Y.$$

4. Half the turnover times and tt-weights of the remaining clients and edges:
$$\tau_j' \leftarrow \frac{1}{2}\tau_j \quad \text{for all } j \in U \quad \text{and} \quad q(e)' \leftarrow \frac{1}{2}q(e) \quad \text{for all } e \in G'.$$

5. Let W_0, W_1 be a *tt*-balanced split of W/Y

6. Compute the child schedules recursively
$$T^i \leftarrow \text{MAXTREESCHEDULE}(G', U', W_i, \tau', q') \qquad \text{for } i = 0, 1 : W_i \neq \emptyset$$
Merge them into the schedule
$$S_{2t-i} \leftarrow S_{2t-i} \cup T_t^i \qquad \text{for } t = 1, \dots, \frac{1}{2}\tau_{max}, \quad \text{for } i = 0, 1 : W_i \neq \emptyset$$

7. Return $S = (S_1, \dots, S_{\tau_{max}})$

Feasibility Feasibility of the schedule is easy to verify. Every client with turnover time 2^k is scheduled daily by a recursive call at depth k. As the schedule propagates back up the call chain, it gets stretched out a by a factor 2 in each step. So, in the end the client is replenished every 2^k days, as required. Hence we will focus on the cost bound.

Cost analysis The height of a tree Height(G) is the maximum weight of a root leaf path in G. We assume that every leaf of the tree is a client (otherwise they can be deleted), and hence 2Height(G) is a lower bound on OPT. We will prove that the cost of a tree connecting the nodes on any day t is at most:

$$L(G, \tau) + \text{Height}(G).$$

After doubling the edges of the tree to get a tour this gives a cost of $2L(G, \tau) + 2\text{Height}(G) \leq 3OPT$, as required.

It will be more convenient to look at leaves of the recursion instead of days though. Note that every set S_t scheduled at some day by the procedure, is generated from the set X scheduled in a leaf call of the recursion together with all the sets scheduled by ancestor calls of that leaf call. Hence, if we can bound the cost of every set of clients generated by a leaf call and its ancestors, we immediately bound the cost of the sets scheduled on every day.

Note: in the rest of this section, we will use the following conventions. G^{un} will always refer to the original (uncontracted) metric. Given a call to MaxTreeSchedule, X and Y refer to the sets defined in Step 1 of that call.

We will start with the following (visually intuitive) observation.

Lemma 3.3.9. *In any call to* MaxTreeSchedule, *Y spans X (in the contracted tree).*

Proof. Assume $|X| > 1$ otherwise there is nothing to be done. Any shortest path between vertices in X must be completely made up of edges e with $q(e) = 1$, and therefore is completely contained in $Y = \{e \in W : q(e) = 1\}$. So, Y spans X. □

Lemma 3.3.10. *In any call to* MaxTreeSchedule, *let Y^{anc} be the set of edges that have been contracted so far by ancestor calls. Then $Y^{anc} \cup Y$ spans X in G^{un}.*

Proof. By Lemma 3.3.9, Y spans X in the contracted tree G. Now, if we uncontract any single previously contracted edge e, then $Y \cup \{e\}$ again spans X in the uncontracted graph. The lemma follows by repeatedly applying this principle to every edge in Y^{anc}. □

So, we only need to connect the connected components in $Y^{anc} \cup Y$, at most one for each ancestor call, to the root. The next lemma will help in showing that we can do this at the cost of single root leaf path.

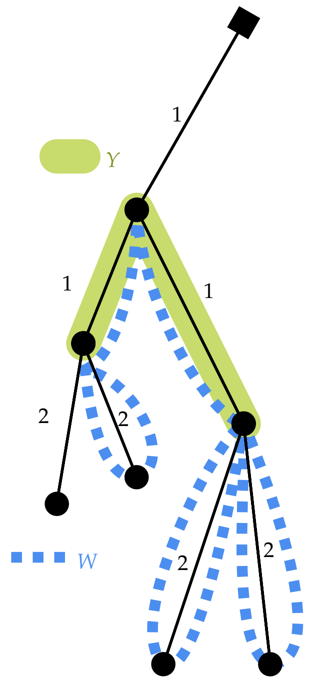


Figure 3.3: Illustration of Lemma 3.3.9. The number next to each edge e represents its tt-weight $q(e)$. The blue dotted indicates the walk W received by the call. The emphasized fat green edges are in Y and will be contracted in the current iteration.

Lemma 3.3.11. *In any call to* MaxTreeSchedule, *if a nonempty set of edges Y is contracted, there is a vertex j in the subtree spanned by Y, such that all clients $k \in W$ with $\tau_k > 1$ are descendants of j in G.*

Proof. Note that the vertex $j \in W$ which is closest to the root, is also in the subtree spanned by Y. The result then immediately follows by picking this j. □

Lemma 3.3.12. *In any call to* MaxTreeSchedule, *that contracts a nonempty set of clients X, let k be an arbitrary client in X and P a shortest path from k to the root in G^{un}. Let Y^{anc} be the set of edges contracted and X^{anc} the set of clients scheduled in ancestor calls.*

Then $Y^{anc} \cup Y \cup P$ contains a tree connecting $X \cup X^{anc}$ to the root of G^{un}.

Proof. By Lemma 3.3.11, any previously contracted set of edges is incident to a vertex j that is an ancestor of k, and therefore lies on P. It follows that P connects every connected component in $Y^{anc} \cup Y$ to the root. Using Lemma 3.3.10 it now follows that $Y^{anc} \cup Y \cup P$ contains a tree spanning every client scheduled by an ancestor plus the root. □

It remains to show that the cost of $Y^{anc} \cup Y \cup P$ is at most $L(G^{un}, \tau) + \text{Height}(G^{un})$. Since the cost of P is at most $\text{Height}(G^{un})$, it is enough to show that the cost of the contracted edges $Y^{anc} \cup Y$ is at most $L(G, \tau)$. We prove a slightly stronger statement in Lemma 3.3.13, to facilitate a proof by induction.

Lemma 3.3.13. *In any call to* MaxTreeSchedule*, let Y^{anc} be the set of edges that have been contracted so far by ancestor calls. Then*

$$\sum_{e \in Y^{anc}} c(e) + \sum_{e \in W} c(e)/q(e) \leq L(G^{un}, \tau).$$

Proof. By induction on the depth of the recursion. For the base case it clearly holds since Y^{anc} is empty and W is a TSP tour, which has tt-weighted cost exactly $L(G^{un}, \tau)$ as it contains every edges twice.

Suppose it holds for depth h. We will prove it holds for depth $h + 1$. To do this, it is enough to show that

$$\sum_{e \in W} c(e)/q(e) - \sum_{e \in W_i} c(e)/q'(e) \geq \sum_{e \in Y} c(e) \quad \text{for } i = 0, 1.$$

That is, the cost decrease in the walk sent to each of the child calls, pays for the cost of the edge set Y that is contracted in the current call. Since $q(e) = 1$ for all $e \in Y$:

$$\sum_{e \in Y} c(e) = \sum_{e \in Y} c(e)/q(e) = \sum_{e \in W} c(e)/q(e) - \sum_{e \in W/Y} c(e)/q(e). \qquad (3.1)$$

Finally, recall the definition of tt-balanced split 3.3.6 and that $q'(e) = 2q(e)$ for $e \in W/Y$. We continue:

$$\sum_{e \in W} c(e)/q(e) - \sum_{e \in W/Y} c(e)/q(e) \geq \sum_{e \in W} c(e)/q(e) - \sum_{e \in W_i} c(e)/q'(e). \qquad (3.2)$$

Now connecting (3.1) and (3.2) gives the required bound, finishing the proof. □

Now to show the required inequality

$$\sum_{e \in Y^{anc} \cup Y} c(e) \leq L(G, \tau),$$

we use Lemma 3.3.13 and, because $q(e) = 1$ for all $e \in Y$, the fact that $\sum_{e \in Y} c(e) \leq \sum_{e \in W} c(e)/q(e)$ in any call to MaxTreeSchedule.

This brings us to the main result of our cost bound.

Lemma 3.3.14. *Let S be the schedule returned by Procedure 3.3.8 on instance (G, τ). The maximum cost of a shortest tour visiting all clients S_t any day t, is at most*

$$2L(G, \tau) + \mathit{Height}(G).$$

Time and space complexity The number of recursive calls is polynomial in the input size as every split of a walk removes one edge, and once the walk is empty no further calls are generated.

However, the calls themselves cannot trivially be implemented in polynomial space, as τ_{max} is not polynomially bounded in the input size. This can easily be fixed by using a more compact representation of the schedule than explicitly storing all sets for day 1 to τ_{max}. One example would be to save the sets of clients scheduled in each call of the recursion, which gives enough information to compute the entries of the schedule on the fly.

Finally, recall that we assumed that turnover times are powers of two. To apply Procedure 3.3.8 to general instances, we need to round the turnover times, losing a factor of two on the bound implied by Lemma 3.3.14. Taking all this into account, we state our main result in Theorem 3.3.15.

Theorem 3.3.15. *There is a 6-approximation for* MIN-MAX RFTT *on trees.*

3.3.2 *MIN-AVG on the line*

As an even more special underlying metric, we might consider the MIN-AVG problem on the line (on a path). For the MIN-MAX version this case is trivial, but for the MIN-AVG version its complexity is unclear: we do not know whether it is in NP, although we expect it to be NP-hard.

On the positive side we can show that the problem is not strongly NP-hard.

Theorem 3.3.16. MIN-AVG *on the line can be solved in pseudopolynomial time.*

Since we are minimizing the average, it is easy to see that we can reduce this problem to two times the MIN-AVG problem on the half-line, i.e. where the depot is an endpoint of the path. On the half-line each vertex i is completely defined by a distance $d_i \in \mathbb{N}$. Suppose vertices are numbered such that $d_1 \leq \ldots \leq d_n$. We present a pseudopolynomial time dynamic programming algorithm for this problem, based on the following observations.

Like in the tree case, we may assume that $\tau_i \leq \tau_j$ for $i < j$. Thus, after visiting j, all $i \leq j$ have a remaining turnover time of τ_i. For the dynamic program to work, we guess L, the day on which vertex n is visited for the first time and try all guesses between 1 and τ_n.

The DP works as follows. We first guess the time L at which vertex n is visited. After this time we might as well cycle the schedule so we only need a schedule for the first L days. We recursively computes the following values. For $i \in \{1, ..., n\}$ and $k \in \{1, ..., L\}$, let $\phi_L(i,k)$ denote the minimum cost of a schedule for the first k days that is feasible for all vertices $1, \ldots, i$ during that period[5]. We want to compute $\phi_L(n, L)$ recursively.

Suppose we want to compute $\phi(i,k)$ and we have computed $\phi_L(i',k')$ for all $i' \leq i$ and $k' \leq k$ with at least one of the inequalities strict.

Note that if $k < \tau_i$ we don't need to visit i during the first k days at all and hence $\phi(i,k) = \phi(i-1,k)$. If not, suppose we visit i for the first time on day $\ell \leq \tau_{max}$. This partitions the schedule in two independent periods $[1, \ell - 1]$ and $[\ell + 1, L]$.

Before day ℓ, we need a schedule that works for the first $i-1$ vertices, then we visit i, so after day ℓ all the vertices up to i have effectively been reset.

[5] That is, we check if every vertex i is visited every interval of length τ_i that fits inside the first k days, and we only count the cost of visiting vertices during the first k days.

Hence, for the days from $\ell + 1$ to k, we can use any schedule that is feasible for days 1 to $k - \ell$ (for the first i vertices). So, the recursion becomes:

$$\phi_L(i,k) = \begin{cases} \phi_L(i-1,k), & \text{if } k < \tau_i \\ \min_{\ell \leq \tau_i} \phi_L(i-1,\ell-1) + d_i + \phi_L(i,k-\ell), & \text{else} \end{cases}. \tag{3.3}$$

We initialize by setting $\phi_L(0,k) = \phi_L(i,0) = 0$ for all i. The recursion can then be computed in time polynomial in L and n. Moreover, we need to check the DP for every possible guess of L. The average daily cost of the optimal schedules is given by:

$$\min_{L \leq \tau_n} \phi_L(n,L)/L, \tag{3.4}$$

which can be computed in time polynomial in n and τ_n, implying Theorem 3.3.16.

3.4 *Approximation on general graphs*

We start with logarithmic approximations for the MIN-MAX objective. The MIN-AVG case, for which we achieve a better performance guarantee, follows in the rest of the section.

3.4.1 *Approximating* MIN-MAX *on general graphs*

Our results for the MIN-MAX objective rely on a reduction to the k-tsp problem. Here, we have a graph with designated depot vertex, and the goal is to cover all other vertices in the graph using k tours, each of which must contain the depot. For this problem a $\frac{5}{2}$-approximation is known [40].

Theorem 3.4.1. MIN-MAX RFTT *has an $O(\log \tau_{max})$-approximation.*

Proof. By Lemma 3.3.1 we may assume every τ_i is a power of 2 so that there are at most $\log \tau_{max}$ different turnover times. We create an instance for each $\tau' \in \{1, 2, 4, \ldots, \tau_{max}\}$, containing the clients i with $\tau_i = \tau'$. We solve each independently and concatenate the solutions to find a solution for the original instance. Our result then follows from the fact that for all these instances a constant factor approximation is available. Indeed, every instance corresponds to an instance of the k-tsp problem, where k is equal to the turnover time of the vertices in the instance. □

Now, using a little more thorough analysis, we can turn this result into an $O(\log n)$-approximation.

Theorem 3.4.2. MIN-MAX RFTT *has an $O(\log n)$-approximation.*

Proof. Again, assume all turnover times are powers of 2. Next, we split up the instance into two new instances. To this end we first define a turnover time k to be *saturated* if $|\{j \in V | \tau_j = k\}| \geq k$. In the first instance we retain the set of vertices V_1 with saturated turnover times, and in the second all vertices V_2 with unsaturated turnover times. Now if all turnover times are saturated, then $\tau_{max} = O(n)$ and we can find a $O(\log n)$-approximation using Theorem 3.4.1. So what remains is to find a $O(\log n)$-approximation for the second instance.

Since no turnover time is saturated, it is easy to see that we can partition the vertices in V_2 into $\lceil \log n \rceil$ sets $W_1, W_2, W_4, \ldots, W_{2^{\lceil \log n \rceil}}$, such that $|W_i| \leq i$, and such that $\tau_j \geq i$ for all $j \in W_i$. For example sort the vertices by turnover time in ascending order, and add them to the sets greedily. We now take any schedule that visits the clients in a set W_k on different days, for each k. This is feasible and implies that at most $\log n$ clients are visited on a given day, which leads to $O(\log n)$-approximation factor, as required. □

3.4.2 *Approximating* MIN-AVG *on general graphs*

We now turn our attention to the MIN-AVG case. Our result relies on a reduction to the Steiner tree over time problem (STOT) that loses only a constant factor in optimality. As an $O(\log \log n)$-approximation is known for this problem (Theorem 2.3.15, Chapter 2), we immediately get an $O(\log \log n)$-approximation for the MIN-AVG RFTT problem. While it is relatively straightforward to prove the relation between the optimal values of the instances, the STOT problem does not have a compact formulation like the RFTT problem. Hence, we must take care not to blow up the instance size. To achieve this, we will show that we can ensure that τ_{max} is polynomially bounded in n.

Lemma 3.4.3. *For each instance I^R of* MIN-AVG *RFTT, there is a corresponding instance I^S of STOT, such that every solution to I^S induces a feasible solution to I^R. Furthermore, the optimal average daily cost of I^S is within a constant of the optimal average daily cost of I^R, and I^S has size polynomial in n and τ_{max} of the instance I^R.*

Proof. Assume that the instance $I^R = (G, \tau)$ of RFTT has turnover times that are powers of 2. We create the instance I^S of STOT as follows. The depot is the same as in the RFTT instance. For all $j \in V$ let $\tau'_j = \max(\frac{1}{2}\tau_j, 1)$. The time horizon of the STOT instance has length τ'_{max}. We will create a copy of j for each *consecutive* time window of length τ'_j inside the time horizon. To be precise: for each j create τ'_{max}/τ'_j copies $j_1, j_2, \ldots$ of j. Set the time window[6] F_{j_i} of the ith copy to

$$F_{j_i} = [(i-1)\tau'_j + 1, i\tau'_j].$$

[6] Note that, since the turnover times are powers of 2, the time windows created here form a laminar family. In particular a subfamily of the dyadic intervals (see Chapter 2, Definition 2.2.7).

Any feasible schedule to the STOT instance I^S can be turned into a feasible schedule to the RFTT instance by repeating it indefinitely. This is true since every period of τ_j days must contain at least one of the consecutive periods of τ'_j days in its entirety, and so each node is visited on time.

Moreover, we have $OPT_{I^S} \leq 2OPT_{I^R}$, since any schedule for I^R can be turned into one for I^S by running it at double speed.[7] Finally, since the time horizon has length $\tau'_{max} \leq \tau_{max}$, and we create at most $n\tau'_{max}$ clients in the STOT instance, the instance size is polynomially bounded in n and τ_{max}. □

[7] I.e. on day t connect the nodes that are replenished on day $2t-1$ and $2t$ in the RFFT schedule.

Bounding the time horizon In the RFTT instance τ_{max} is not guaranteed to be polynomially bounded in the input size though. However, at the loss of a constant factor, we can decompose the STOT instance into a collection of instances such that for each instance the maximum turnover time is $\tau_{max} \leq n^2$. [8]

[8] Note that we cannot directly use the results from Section 2.2.3 in Chapter 2; it presumes we have a solution to the LP which could have exponential size.

First we split the instance into (at most n) instances for which the ratio between the maximum and minimum distance is at most n. By Lemma 2.2.12 in Chapter 2, this loses only a constant factor. Let c_{min} denote the minimum distance to the depot.

Next, we shrink the time line to ensure that all minimal time windows have size one (as all the STOT instances we create have time windows in a laminar family, this can be achieved by deleting days).

Now, in any feasible schedule, on every day at least one node must be visited (as $\tau_{\min} = 1$), so c_{min} is a lower bound on the cost on any given day. In particular, the schedule for the first n^2 days must cost at least $n^2 c_{min}$. Suppose we visit all nodes in the instance on day n^2. This costs at most $nc_{max} \leq n^2 c_{min}$, which is at most as much as the cost the first n^2 days. By day $n^2 + 1$ the turnover times have been reset and we repeat the schedule for the first n^2 days. Hence, we get the following result:

Lemma 3.4.4. *At the loss of a constant factor, we can split I^S into a collection of instances with $T \leq n^2$.*

Combining Lemma 3.4.3, 3.4.4 and 2.3.15 then leads to the main theorem of this section.

Theorem 3.4.5. *There is a polynomial time $O(\log \log \min(N, \tau_{max}))$-approximation algorithm for the inventory routing problem.*

3.5 *Conclusion*

In this chapter, we considered replenishment problems with fixed turnover times, a natural inventory-routing problem that has not been studied before. We formally defined the RFTT problem and considered the objectives MIN-AVG and MIN-MAX. For the MIN-AVG RFTT, we showed that it is at least as hard as the intractable PINWHEEL SCHEDULING PROBLEM on series-parallel graphs and we gave a 2-approximation for trees. For the MIN-MAX objective we showed NP-hardness on stars and gave a 6-approximation for tree metrics. We also presented a DP that solved the MIN-AVG RFTT in pseudopolynomial time on line graphs. Finally, we gave a $O(\log n)$-approximation for the MIN-MAX objective and an $O(\log \log n)$-approximation for the MIN-AVG objective on general metrics.

The results that we present should be considered as a first step in this area and many problems remain open. An intriguing open problem is the complexity of the of RFTT on a tree. Namely, for MIN-AVG variant we conjecture that the problem is hard, and we ask whether the simple 2-approximation we provide can be improved. For the MIN-MAX variant it is open whether the problem is APX-hard and whether we can improve the 6-approximation,

Next to replenishing locations with routing aspects as we studied in this chapter, scheduling problems modeling maintenance or security control of systems, form a class of problems to which this model naturally applies. It would be interesting to study such fixed turnover time problems in combination with scheduling. Would this combination allow for more definitive results?

4
Fixed order scheduling on parallel machines

Abstract We consider the following natural scheduling problem: Given a sequence of jobs with weights and processing times, one needs to assign each job to one of m identical machines in order to minimize the sum of weighted completion times. The twist is that for each machine the jobs assigned to it must obey the order of the input sequence, as is the case in multi-server queuing systems. We establish a constant factor approximation algorithm for this (strongly NP-hard) problem. Our approach is necessarily very different from what has been used for similar scheduling problems without the fixed-order assumption. We also give a QPTAS for the special case of unit processing times.

The contents of this chapter appeared in:
[14] Thomas Bosman, Dario Frascaria, Neil Olver, René Sitters and Leen Stougie.
" Fixed-Order Scheduling on Parallel Machines".
International Conference on Integer Programming and Combinatorial Optimization (IPCO), 2018.

4.1 Introduction

We consider an extremely simple, yet challenging, scheduling principle that arises in many logistic and service applications. Given a sequence of jobs and a set of machines, we need to dispatch the jobs one by one over the machines, where for each machine the ordering of the original sequence is preserved. Hence, each machine must handle the jobs in a first-in first-out (FIFO) order. Each job has a processing time p_j and weight w_j and the goal is to minimize the sum of weighted completion times, where the completion time of a job j is the total processing time of the jobs preceding j (including j) on the same machine.

The FIFO-ordering restriction is common in queuing theory, where the *task assignment problem* [52, 51, 35] is concerned with the same question, except that jobs arrive stochastically over time. Our problem can be seen as asking for the optimal way of dispatching jobs from a single queue over m server queues under complete information of the processing times, essentially *unzipping* a single queue into m queues. The reverse problem of *zipping* m queues into a single queue, is the classic single machine scheduling problem: $1|\text{chains}|\sum w_jC_j$ (in the 3-field notation by Graham et al. [47]) and can be solved efficiently by greedily selecting a prefix of jobs with largest ratio of total weight over total processing time [61].

In the scheduling literature, the fixed-ordering scheduling problem can be seen as a special case of *scheduling problems with sequence dependent setup times*, where for each pair of jobs there is a changeover cost c_{ij} that is paid if job j is the immediate successor of job i on a machine. Such setup times occur naturally in many industrial applications [1]. Our problem is precisely this, in the special case where $c_{ij} = \infty$ if i is later than j in the ordering, and $c_{ij} = 0$ otherwise. While the problem has received substantial attention (see [2, 3, 1] for a comprehensive literature review), almost nothing is known from a theoretical perspective (an exception is [53], but this is concerned with a rather unusual objective function). We believe our work may shed light on this more general problem.

We remark that the online version of the problem, where we need to assign a job before we get to know the next job in the sequence, does not admit a constant-competitive algorithm. Consider, for example, two machines and a sequence of three jobs (where job 1 should be completed first and job 3 last on any machine) with $p_1 = k^2, p_2 = k, p_3 = 1$ and $w_1 = 1, w_2 = k$ and w_3 is either zero or k^3, depending on the schedule of the first two jobs. Here, k is an arbitrarily large number. It is easy to see that a good schedule requires knowledge of the weight of job 3 before deciding whether to put jobs 1 and 2 on the same machine.

Our contribution. Scheduling problems with weighted completion times objective *without* ordering constraints typically admit good approximations algorithms. For identical machines there is a PTAS [75], while a slightly better than $\frac{3}{2}$-approximation [7] for unrelated machines is known. But even simpler approaches yield a constant factor approximation. On identical machines, it is a classic result due to Kawaguchi and Kyan [59] (see [73] for a modern proof) that scheduling greedily according to Smith ratio (see Section 4.2) is a $\frac{1+\sqrt{2}}{2}$-approximation. For unrelated machines, independent rounding of both a natural time-indexed LP [70] and a (nontrivial) convex quadratic program [74] works, and achieve approximation ratios of $\frac{3}{2} + \epsilon$ and $\frac{3}{2}$ respectively. Another approach is *α-point scheduling* [71], where jobs are sorted according to the time by which an α fraction of the job has been processed in some LP relaxation. The jobs are then scheduled greedily in that order. This method has enabled many algorithmic improvements in scheduling, since it can be modified to deal with additional complications, such as precedence constraints and release times [68].

Fixed-order scheduling appears highly resistant to all these techniques. A big obstacle is that moving even a single pair of jobs onto the same machine can have a catastrophic effect on the objective function if the order is fixed: think of a job with large processing time but minuscule weight, followed by a job with large weight and minuscule processing time. Thus in order to have any hope of a non-trivial performance guarantee, jobs must be assigned to machines in a highly dependent way. To achieve this, our approach radically departs from earlier ones.

We define an important partial order $\prec$: essentially, $j \prec k$ means that not only is j earlier than k in the FIFO ordering, but also this ordering is opposite to what would be preferred according to Smith's rule. Thus as alluded to earlier, if $j \prec k$ we should be particularly careful about assigning j and k to the same machine. Order the machines arbitrarily. A key idea is that at only a constant factor loss, we can restrict our attention to a class of schedules we call *Smith-monotone*, meaning that if $j \prec k$, then j is assigned to an earlier machine (or the same machine) as the one that k is assigned to. Next, we relax the problem by computing each job's completion times partially, based only on jobs preceding it in the partial order $\prec$. While this potentially distorts completion times a lot, we can ensure the amortized effect is not too large by appropriately rounding weights and processing times. Finally, we formulate a new LP using these partial completion times and enforcing the strong structure imposed by Smith monotonicity. This LP can be rounded in a way that completely respects the pairwise probabilities of jobs being assigned to the same machine; Smith monotonicity is crucial here. The rounding is very appealing and natural, and can be seen as an analog of α-point scheduling with respect to machine index rather than time.

Finally, we remark that for the case of unit processing times, the complexity of the problem is unknown, but it is unlikely to be APX-hard: we present a QPTAS in Section 4.6.1 of the appendix.

4.2 *Problem Definition, Notation and NP-hardness*

We have a set of identical machines $M = \{1, \ldots, m\}$, each of which can process one job at a time, and a totally ordered set of jobs $J = 1, ..., n$, where each job $j \in J$ has weight $w_j \in \mathbb{N}$ and processing time $p_j \in \mathbb{N}$. Because of its frequent use we also define notation for the so called *Smith ratio* $s_j := \frac{w_j}{p_j}$ of job j. A feasible schedule $\mu : J \to M$ assigns to every job j a machine $\mu(j)$. Each machine processes the jobs assigned to it in order of their number. The cost of the schedule μ is the sum of weighted completion times, i.e.,

$$\Gamma(\mu) = \sum_{k \in J} \sum_{\substack{j \in J : j \leq k, \\ \mu(j) = \mu(k)}} p_j \cdot w_k. \tag{4.1}$$

The objective is to minimize the cost of the schedule. We denote by OPT the optimal cost, by $\sigma_\mu : J \to \mathbb{N}$ the function that maps every job to its completion time under μ and by $\prec$ a partial order on J such that $j \prec k$ if and only if $j < k$ and $s_j \leq s_k$.

Note that we explicitly disallow $p_j = 0$. This is for convenience, and ensures that the Smith ratio w_j / p_j is always well defined. Our results easily extend to the case where jobs with zero processing time are allowed.

The problem of minimizing the sum of weighted completion times without ordering constraints is a classic problem that has long been known to be strongly NP-hard [43]. This result extends to fixed-order scheduling as well: given an assignment of jobs to a machine, it is always optimal to schedule them in decreasing order of Smith ratio, so the ordering constraints become redundant if $s_1 \geq s_2 \geq \cdots \geq s_n$.

4.3 *Structural properties of optimal solutions*

In this section we will provide a characterization of an optimal solution which will help us construct a constant-factor approximation algorithm in Section 4.4.

Unit processing times

Suppose all jobs have equal processing time and hence that the relation $j \prec k$ indicates that $j < k$ and $w_j \leq w_k$. W.l.o.g. we may then as well assume that the processing times are 1. An initial simplification is that we will assume that schedules are *staircase shaped*, in the sense that for every prefix of the jobs $1, \ldots, k$, the number of jobs assigned to each machine decreases monotonically with the machine index. We will use the following equivalent definition.

Definition 4.3.1. A schedule μ is **staircase shaped** if for each job k, $\mu(k) = |\{j < k : \sigma_\mu(j) = \sigma_\mu(k)\}| + 1$.

Given any schedule μ, we can clearly turn it into a staircase shaped schedule without changing the completion time of any job.

Clearly we want jobs with high weights to be completed early, but this may not always be possible because of the ordering on the jobs. Intuitively, it seems like a good idea to 'reserve' some of the good spots early in the schedule, for higher weight jobs later in the sequence. In staircase shaped schedules this means that early and low weight jobs should be put on low index machines as much as possible. Lemma 4.3.3 below makes this more precise.

Definition 4.3.2. A schedule μ is **Smith-monotone** if, for every $j \prec k$, it holds that $\mu(j) \leq \mu(k)$.

Lemma 4.3.3. *For unit processing times, there exists an optimal schedule that is Smith-monotone and staircase shaped.*

Proof. Let us define the potential function $\sum_{j \in J} \mu(j) j$. Let μ be a solution maximizing this potential function among those that are optimal and staircase shaped. Suppose μ is not Smith-monotone. We obtain a contradiction by showing there is a staircase shaped schedule with a higher potential but no higher cost.

Since μ is not Smith-monotone, there exists a pair $j \prec k$ that violates Smith monotonicity. Pick j, k so that there is no other violating pair $j' \prec k'$ between it, i.e with $j \leq j'$ and $k' \leq k$ and at least one of the inequalities strict. We call such a pair *tight*. For $g = j, k$, let $S_g = \{h \in \{j+1, \ldots, k-1\} : \mu(h) = \mu(g)\}$ be the set of jobs between j and k on the machine of job g. It follows that $h \in S_k \implies w_h \leq w_j$ and that $h \in S_j \implies w_h \geq w_k$ as, otherwise, the pair $j \prec k$ would not be tight. (In fact, $<$ holds.) By assigning j to the machine of k and vice-versa we get a schedule μ' that improves our potential function. We first show is that the new schedule μ' does not incur a higher cost than μ.

Since only the starting times (whence completion times) of j, k and jobs in S_j and S_k may change, all others remaining equal, it follows that

$$\sum_{h \in \{j\} \cup S_k} \sigma_{\mu'}(h) + \sum_{h \in \{k\} \cup S_j} \sigma_{\mu'}(h) = \sum_{h \in \{j\} \cup S_k} \sigma_\mu(h) + \sum_{h \in \{k\} \cup S_j} \sigma_\mu(h),$$

hence

$$\sum_{h \in \{j\} \cup S_k} (\sigma_{\mu'}(h) - \sigma_\mu(h)) = \sum_{h \in \{k\} \cup S_j} (\sigma_\mu(h) - \sigma_{\mu'}(h)). \tag{4.2}$$

Now note that the completion times of jobs j and S_k increase, while those of k and S_j decrease, since the schedule was staircase shaped. Combining this with the fact that $w_h \leq w_j$ for $h \in S_k$ and $w_h \geq w_k$ for $h \in S_j$, we can bound the increase in the cost as follows:

$$\sum_{h\in\{j,k\}\cup S_j\cup S_k} w_h(\sigma_{\mu'}(h)-\sigma_\mu(h)) =$$
$$\sum_{h\in\{j\}\cup S_k} w_h(\sigma_{\mu'}(h)-\sigma_\mu(h)) - \sum_{h\in\{k\}\cup S_j} w_h(\sigma_\mu(h)-\sigma_{\mu'}(h)) \leq$$
$$w_j \sum_{h\in\{j\}\cup S_k} (\sigma_{\mu'}(h)-\sigma_\mu(h)) - w_k \sum_{h\in\{k\}\cup S_j} (\sigma_\mu(h)-\sigma_{\mu'}(h)) \leq 0 \quad \text{by (4.2).}$$

So the new schedule has a higher potential function and no higher cost. The schedule may not be staircase shaped, however. But simply sorting the jobs fixes this without undoing our work. To see this, note that the set of timeslots occupied on each machine did not change when we modified the schedule. So the fact that the old schedule was staircase shaped, implies that the new schedule still has the following weaker property: if exactly k jobs are scheduled at time t, they are scheduled on the first k machines. By sorting all the jobs assigned to one timeslot by number and assigning them to the first available machine in that order, the potential function can only increase further, while completion times stay the same. Hence, we have found an optimal staircase shaped schedule with higher potential function than we started with, contradicting our choice of the original schedule and concluding the proof. □

General processing times

Unfortunately, for general processing times we cannot hope for an equally nice structural result. Indeed, there may not be an optimal schedule that is Smith-monotone. However, as we will now show, we may impose this structure with the loss of only a constant factor in the objective.

The proof works by reducing a general instance to one with unit processing times, finding an optimal Smith-monotone schedule, and then rounding it back. Although this reduction is not polynomial time, it will suffice to prove the bound on the optimality ratio. Instead, in the next section we will find that we can bypass the reduction, and approximate such a schedule directly.

Given an instance I to the general problem, let I^{unit} be the instance with unit processing times obtained from I by replacing every job $j \in J$ with a set $U(j) = \{j^1, \dots, j^{p_j}\}$ of p_j consecutive jobs, each having unit processing time and weight w_j/p_j. Let J^{unit} be the set of jobs of I^{unit} and OPT^{unit} be the optimal cost for I^{unit}.

Lemma 4.3.4. $\text{OPT}^{unit} \leq \text{OPT} - \sum_{j\in J} \frac{1}{2}(p_j - 1)w_j$

Proof. The statement follows for the schedule μ^{unit} obtained from the optimal schedule for I by putting all the jobs in $U(j)$ on the machine $\mu(j)$, for all $j \in J$. The completion times of the jobs in $U(j)$ run from $\sigma(j) - p_j + 1$ to $\sigma(j)$, and hence the average completion time is $\sigma(j) - \frac{1}{2}(p_j - 1)$. The proof follows from multiplying by the total weight of the jobs in $U(j)$. □

Lemma 4.3.5. *An optimal Smith-monotone schedule for I has cost at most* $\text{OPT}^{unit} + \sum_{j\in J}(p_j - 1)w_j$.

Proof. Given an optimal schedule μ^{unit} for I^{unit}, we can create a schedule for I by putting job j on machine i with probability $|\{h \in U(j) : \mu^{unit}(h) =$

$i\}|/|U(j)|$, for all $j \in J$. The expected time spent processing job $j \in J$ on machine i is exactly $|\{h \in U(j) : \mu^{unit}(h) = i\}|$. As a consequence, the expected starting time of a job $j \in J$ is at most the average starting time of the jobs in $U(j)$, and thus the expected completion time is at most the expected completion time of jobs in $U(j)$ plus $p_j - 1$ (which is the difference between the processing time of job j and any job in $U(j)$).

What remains to show is that μ is Smith-monotone. By Lemma 4.3.3 we can assume, w.l.o.g., that the optimal solution of I^{unit} satisfies Smith monotonicity. Consider an arbitrary pair $j \prec k$. We have that $j' \prec k'$ for all jobs $j' \in U(j), k' \in U(k)$. This implies that, if any job in $U(j)$ is scheduled on machine i, all jobs in $U(k)$ are scheduled on machines with index not smaller than i. Hence it holds that the machine with highest index to which j may be assigned cannot have index larger than any machine to which k may be assigned. Therefore, for every possible realization of the random schedule, Smith monotonicity is satisfied, completing the proof. □

Lemma 4.3.6. *An optimal Smith-monotone schedule has cost at most* $\frac{3}{2}$OPT.

Proof. By Lemma 4.3.5, we have that an optimal Smith-monotone schedule has cost at most $\text{OPT}^{unit} + \sum_{j \in J}(p_j - 1)w_j$, which in turn, by Lemma 4.3.4, is at most $\text{OPT} + \frac{1}{2}\sum_{j \in J}(p_j - 1)w_j \leq \frac{3}{2}\text{OPT}$. □

Though the bound in Lemma 4.3.6 is unlikely to be tight, it cannot be improved much further. The Kawaguchi-Kyan bound of $\frac{1+\sqrt{2}}{2} \approx 1.207$ is known to be tight [59, 73], and this lower bound carries over to fixed-order scheduling: the worst-case example uses jobs with equal Smith ratios, and in that case reordering the jobs assigned to a machine does not change the cost.

4.4 *A constant factor approximation algorithm*

In this section we will describe an algorithm that proves our main result: Theorem 4.4.1. Our approach is as follows: first we round the instance such that all Smith ratios are powers of $\frac{1}{3}$ (by rounding up the weights as appropriate). Given that, we show that a certain relaxed objective function is always within a constant of the original objective function. We then use LP rounding to find a Smith-monotone schedule that is optimal with respect to the relaxed objective function.

Theorem 4.4.1. *Fixed-order scheduling can be approximated to within a factor* $\frac{27}{2} + 9\sqrt{3} < 29.1$.

From hereon assume that all Smith ratios are powers of some fixed $\gamma \in (0, 1)$. Suppose we relax the objective function (4.1) to disregard the contribution of processing times of jobs with $j < k$ and $w_j > w_k$ (hence, $j \not\prec k$) in the completion time of job k. We claim this relaxation loses only a factor $(\frac{4}{1-\sqrt{\gamma}} - 3)$.

Definition 4.4.2. The **partial completion time** of a job $k \in J$ under a schedule μ is

$$\tilde{c}_k = \sum_{j \preceq k : \mu(j) = \mu(k)} p_j .$$

The **partial cost** of a schedule μ is $\tilde{\Gamma}(\mu) = \sum_{k \in J} w_k \tilde{c}_k$.

It is crucial that the Smith ratios are powers of γ; this ensures that either $j \preceq k$ or w_j is relatively large compared to w_k. Intuitively, in the latter case we don't care too much about the effect of j's processing time on the lighter-weight job k.

Theorem 4.4.3. *Take an instance where all Smith ratios are positive powers of $\gamma \in (0,1)$. Consider any schedule μ and denote its cost by $\Gamma(\mu)$. It holds that*

$$\tilde{\Gamma}(\mu) \leq \Gamma(\mu) \leq \left(\frac{4}{1-\sqrt{\gamma}} - 3\right) \cdot \tilde{\Gamma}(\mu).$$

Since it is obvious that $\tilde{\Gamma}(\mu) \leq \Gamma(\mu)$ we prove the upper bound.

To simplify notation assume the instance has only one machine; the result can be applied to each machine individually. For $d \in \mathbb{N}$, let N_d be the set of jobs j with $s_j = \gamma^d$; let W_d and P_d be, respectively, the total weight and total processing time of jobs in N_d. We denote by $H_d = \frac{1}{W_d}\sum_{k \in N_d} w_k \tilde{c}_k$ the weighted average of the partial cost in N_d. It follows that

$$\tilde{\Gamma}(\mu) = \sum_{k \in J} w_k \tilde{c}_k = \sum_{d \in \mathbb{N}} W_d H_d = \sum_{d \in \mathbb{N}} \gamma^d P_d H_d. \tag{4.3}$$

Our goal is to bound $\Gamma(\mu)$ in the same terms. Since H_d only accounts for the contribution of jobs with Smith ratio at most γ^d in the completion time of jobs in N_d, we need to correct for the other jobs (that are in $N_1, \dots, N_{d-1}$). In the worst case, all these jobs are scheduled first and hence their processing times need to be added. Therefore we get:

$$\Gamma(\mu) \leq \sum_{d \in \mathbb{N}} W_d (H_d + \sum_{i=1}^{d-1} P_i) \leq \sum_{d \in \mathbb{N}} \gamma^d P_d (H_d + \sum_{i=1}^{d-1} P_i). \tag{4.4}$$

We will prove that this value can be bounded by the desired constant times $\tilde{\Gamma}(\mu)$. Globally our strategy is to show that every newly introduced term $\gamma^d P_d P_i$ can be charged to a term in the expression for $\tilde{\Gamma}(\mu)$ such that no term gets charged more than a $\frac{4}{1-\sqrt{\gamma}} - 4$ fraction of its value.

Before we can proceed we will need the inequality in Lemma 4.4.4 below. It says that the average weighted completion time in N_d is at least half the total processing time in N_d, which can intuitively be seen as follows: think of all jobs $j \in N_d$ as blocks of equal width, length p_j and mass w_j. So, all blocks have equal density. Now stack the boxes on top of each other: then P_d corresponds to the total length, and H_d approximately to the center of mass, which is in the middle.

Lemma 4.4.4. $P_d \leq 2H_d$.

Proof.

$$H_d = \frac{1}{W_d} \sum_{k \in N_d} w_k \sum_{j \leq k \wedge s_j \leq s_k} p_j \geq \underbrace{\frac{1}{W_d} \sum_{k \in N_d} w_k \sum_{j \leq k \wedge j \in N_d} p_j}_{:=Q}$$

Suppose now that $Q < \frac{P_d}{2}$. Since $W_d = \gamma^d P_d$, it follows that:

$$Q = \frac{1}{P_d} \sum_{k \in N_d} p_k (P_d - \sum_{h > k \wedge h \in N_d} p_h) = P_d - \frac{1}{P_d} \sum_{h \in N_d} p_h \sum_{k < h \wedge k \in N_d} p_k \geq P_d - Q,$$

implying that $Q \geq P_d/2$, a contradiction. □ □

Lemma 4.4.5. $\max\{\gamma^d P_d H_d, \gamma^i P_i H_i\} \geq \frac{1}{2}(\gamma^d P_d P_i)\gamma^{\frac{1}{2}(i-d)}$.

Proof. Suppose that $\gamma^d P_d H_d < \frac{1}{2}\gamma^d P_d P_i \gamma^{\frac{1}{2}(i-d)}$. This implies that

$$H_d < \frac{1}{2} P_i \gamma^{\frac{1}{2}(i-d)}. \tag{4.5}$$

So we obtain

$$\gamma^d P_d P_i \gamma^{\frac{1}{2}(i-d)} \leq \gamma^d 2 H_d P_i \gamma^{\frac{1}{2}(i-d)} \overset{(4.5)}{<} \gamma^{d-i}\gamma^i P_i \gamma^{\frac{1}{2}(i-d)} P_i \gamma^{\frac{1}{2}(i-d)} \leq 2\gamma^i H_i P_i,$$

where the first and third inequalities follow from Lemma 4.4.4. □ □

We are now ready to prove Theorem 4.4.3.

Proof of [. Theorem 4.4.3] We will prove the following inequality, implying the theorem by (4.4):

$$\sum_{d\in\mathbb{N}} \gamma^d P_d H_d \left(\frac{4}{1-\sqrt{\gamma}} - 4\right) \geq \sum_{d\in\mathbb{N}} \gamma^d P_d \sum_{i=1}^{d-1} P_i.$$

Applying Lemma 4.4.5 and replacing the max by a sum we get:

$$\begin{aligned}
\sum_{d\in\mathbb{N}} \gamma^d P_d \sum_{i=1}^{d-1} P_i &\leq \sum_{d\in\mathbb{N}} \sum_{i=1}^{d-1} \gamma^{\frac{1}{2}(d-i)} 2 \max(\gamma^d P_d H_d, \gamma^i P_i H_i) \\
&\leq \sum_{d\in\mathbb{N}} \sum_{i=1}^{d-1} \gamma^{\frac{1}{2}(d-i)} 2(\gamma^d P_d H_d + \gamma^i P_i H_i) \\
&\leq \sum_{d\in\mathbb{N}} \sum_{i=1}^{\infty} \gamma^{\frac{1}{2}i} 4(\gamma^d P_d H_d) = \sum_{d\in\mathbb{N}} \gamma^d P_d H_d \left(\frac{4}{1-\sqrt{\gamma}} - 4\right). \quad \square
\end{aligned}$$

□

Linear Programming Relaxation

Suppose all Smith ratios are positive powers of $\gamma \in (0,1)$. The following mixed-integer program captures the problem of finding a Smith-monotone ordering that minimizes the modified objective $\tilde{\Gamma}(\mu)$.

$$\begin{aligned}
\min \quad & \sum_{k\in J} w_k \tilde{c}_k & & \\
\text{s.t.} \quad & u_k \geq u_j + z_{jk} & \forall j \prec k \quad & (4.6)\\
& u_k \leq m & \forall k \in J \quad & (4.7)\\
& \tilde{c}_k \geq p_k + \sum_{j\prec k}(1 - z_{jk}) p_j & \forall k \in J \quad & (4.8)\\
& u_k \in \mathbb{N}, z_{jk} \in \{0,1\} & \forall k \in J, j \prec k \quad &
\end{aligned}$$

Here, z_{jk} is the indicator variable for the event that j and k are assigned to different machines. The variable u_k indicates which machine job k is assigned to. Finally, $\tilde{c}_k$ represents the partial completion time of job k. The constraint (4.6) is valid since we require a Smith-monotone ordering.

Now consider the natural LP relaxation of this mixed-integer program, where we drop the integrality requirements and instead require $1 \leq u_k \leq m$, $0 \leq z_{jk} \leq 1$. Denote this relaxation by (LP). Let $(z^*, u^*, \tilde{c}^*)$ be an optimal solution to (LP), with cost OPT_{LP}.

Definition 4.4.6. For $\beta \in (0,1)$, the β**-point schedule** associated to u^* is the schedule obtained by assigning job j to machine $\lceil u_j^* - \beta \rceil$.

From now on, β will be chosen uniformly at random from $(0,1)$, making the β-point schedule a random schedule.

Let N_d be the set of jobs with Smith ratios γ^d and let $\tilde{C}_k$ be the (random) partial completion time of job k under the β-point schedule. The following statements are easy to verify.

Proposition 4.4.7. *For any pair of jobs $j \prec k$, the probability that jobs j and k are assigned to the same machine under the β-point schedule is at most $1 - z_{jk}^*$.*

Proof. This follows immediately from the constraint (4.6). □ □

Proposition 4.4.8. *For any $k \in J$, $\mathbb{E}[\tilde{C}_k] \le \tilde{c}_k^*$. Hence*

$$\mathbb{E}\Big[\sum_{k \in J} w_k \tilde{C}_k\Big] \le \sum_{k \in J} w_k \tilde{c}_k^* = \text{OPT}_{\text{LP}}.$$

Proof. This follows from Proposition 4.4.7 and (4.8). □ □

Note that a β-point schedule can be derandomized in polynomial time: a job j is always assigned to u_j^* when u_j^* is integral and to either $\lfloor u_j^* \rfloor$ or $\lceil u_j^* \rceil$ otherwise. Let b_j be the maximum value of β for which $\lceil u_j^* - \beta \rceil = \lfloor u_j^* \rfloor$. It follows that all possible schedules are the ones obtained by assigning to β the values in $\{b_j | j \in J\} \cup \{1\}$.

Constant factor approximation

We are now ready to complete the main proof of this section.

Proof of [. Theorem 4.4.1] Given an instance I where the jobs have arbitrary Smith ratios, let I^γ be the instance obtained from I by rounding up the weights such that the Smith ratios are powers of γ, where γ will be defined later, and then scaled to be in $(0,1)$. Let OPT_I and OPT_{I^γ} be the the optimal cost for the instances I and I^γ respectively. Clearly the cost of any schedule μ on I^γ is at most γ^{-1} times the cost of μ on I and $\text{OPT}_I \le \text{OPT}_{I^\gamma}$ since the cost of a schedule is linear in the weights (see (4.1)) and weights do not affect feasibility.

We denote by μ and OPT_{LP}, respectively, the β-point schedule and the optimal cost of (LP) on I^γ. By Lemma 4.3.6, Proposition 4.4.8 and Theorem 4.4.3, the cost of μ on I is at most

$$\frac{3}{2}\left(\frac{4}{1-\sqrt{\gamma}} - 3\right) \cdot \text{OPT}_{\text{LP}} \le \frac{3}{2}\left(\frac{4}{1-\sqrt{\gamma}} - 3\right) \cdot \text{OPT}_{I^\gamma} \le \frac{3}{2}\left(\frac{4}{1-\sqrt{\gamma}} - 3\right) \cdot \text{OPT}_I \gamma^{-1}.$$

Minimizing over γ, this yields an approximation ratio of

$$\min_{\gamma \in (0,1)} \frac{3}{2}\left(\frac{4}{1-\sqrt{\gamma}} - 3\right) \cdot \gamma^{-1} = \frac{3}{2}(9 + 6\sqrt{3}) = \frac{27}{2} + 9\sqrt{3}$$

when $\gamma = 1/3$, concluding the proof. □ □

4.5 Epilogue

Our work suggests many further interesting and natural directions. One is to find a PTAS (or even a polynomial time exact algorithm) for unit processing times, perhaps expanding on the QPTAS in the appendix. Good approximation algorithms for all of the following problems remain open questions.

(1) There are k arrival lines that need to be dispatched over m servers, such that the FIFO ordering in each of the arrival lines is obeyed in each of the server queues.

(2) An arbitrary partial order on the jobs is given, and we require that if two jobs are assigned to the same machine, then the partial order is respected. (1) is exactly this problem, where the partial order is described by k disjoint chains.

(3) Instead of requiring that the order is exactly preserved, a natural relaxation is to allow a *reordering buffer* (see, e.g. [69]) of limited size. Jobs enter the buffer in the given FIFO order, but any job in the buffer can be chosen and assigned to one of the machines.

4.6 Appendix

4.6.1 A QPTAS for unit processing times

In this section we sketch a simple quasipolynomial time approximation scheme (QPTAS) for the problem under unit processing times. Note that we do not know if this version of the problem remains NP-hard. However, it seems to capture most of the difficulty, so we feel that tackling this case will help substantially in improving the upper bound for the general case. The QPTAS works by solving a relaxed problem by dynamic programming. We round the completion times to geometric intervals and then we consider schedules in which at any time point only one machine per completion time can accept jobs. This sufficiently reduces the solution space to get a quasipolynomial time algorithm.

The first step is to consider only a logarithmic number of distinct completion times. Let $R = \{\lfloor (1+\epsilon)^i \rfloor : i \in \mathbb{N}\}$ be the set of integers found by rounding down a geometric series growing with rate $1+\epsilon$. Then order the elements $1 = R_1 < R_2 < \dots$ and take $R_0 = 0$ by convention. Assume that s is the smallest index such that $R_s \geq n$, and note that $s = O(\log_{1+\epsilon}(n))$. Now consider the objective of minimizing the weighted sum of *rounded* completion times, where each completion time is rounded up to the nearest R_i. Call this the *rounded objective*; clearly the rounded objective of any schedule is at most $1+\epsilon$ times the actual objective. So, if we can find an optimal segmented schedule for the rounded objective, we immediately get a $(1+\epsilon)$-approximation to the original problem.

Now we define a restricted type of schedule, which we call a *segmented staircase schedule*. A segmented staircase schedule is similar to a staircase shaped schedule, except that the "steps" are now defined in terms of the rounded completion times. When j is assigned to a machine, it is assigned to the leftmost machine that gives it the same rounded completion time. In other words, if a job j gets assigned to $\mu(j)$ and gets completed at time $t \in (R_i, R_{i+1}]$, then $\mu(j)$ is the lowest index machine for which the number of jobs $k < j$ assigned to it does not exceed $R_{i+1} - 1$.

Lemma 4.6.1. *There is an optimal solution to the problem of minimizing the rounded objective that is a segmented staircase schedule.*

Proof. Take μ to be a schedule for which $(\mu(1), \mu(2), \dots, \mu(n))$ is lexicographically minimal amongst all solutions that are optimal for the rounded objective. Notice that μ must be staircase shaped; otherwise, transforming it into a staircase shaped schedule would yield a schedule μ' in which every job has the same completion time, but which is lexicographically smaller than μ.

Suppose for a contradiction that this schedule is not a segmented staircase schedule. Let j be the last (maximum index) job that violates the rule for a segmented staircase schedule: j is assigned to machine h', but $h < h'$ is the smallest index machine that gives it the same rounded completion time, ignoring all jobs after j. Let k be the first job $k > j$ scheduled on machine h. (If there is no such job, then moving j to h' reduces the lexicographical value and does not increase the total rounded completion time.) Note that since j was chosen maximally, no job ℓ with $j < \ell < k$ is scheduled on machine h'. Moreover, $\sigma_\mu(k) > \sigma_\mu(j)$, since μ is staircase shaped, so it must be that j and k are both in the same segment $(R_i, R_{i+1}]$ for some i. So we can simply swap k and j to obtain a lexicographically smaller schedule of the same rounded objective value. $\square$

For segmented staircase schedules, we can compactly describe the loads on the machines just prior to assigning job j. Let X_i^j be the number of jobs on machines with load currently in the interval $[R_i, R_{i+1})$ just prior to assigning job j. Since only one of the machines with load in that interval can have strictly more than R_i jobs on it, this number also completely determines how many machines there are with loads exactly R_i. For each j, we have $n^{O(\log_{1+\epsilon} n)}$ options for the values of $X_1^j, X_2^j, \dots, X_s^j$. Once we know the minimum cost of a schedule attaining each of those options, we can compute the cost of all the schedules up to job $j + 1$. Hence, we get the main result of this section.

Theorem 4.6.2. *For any $\epsilon > 0$, there is a $(1 + \epsilon)$-approximation algorithm for fixed-order scheduling with unit processing times with running time $n^{O(\log_{1+\epsilon} n)}$.*

4.6.2 Negative results

In this section we will show that some common approaches to approximating scheduling problems are doomed to fail on the fixed-order scheduling problem.

Independent rounding A non-trivial LP rounding algorithm that works by independently assigning jobs to machines cannot give an approximation ratio sub-linear in the number of jobs. This holds even when all processing times are 1, there are 3 machines, and if we start with a convex combination of optimal schedules that satisfy Smith monotonicity.

For $k \in \mathbb{N}$, define an instance on 3 machines with unit processing times consisting of $n = 2k + 2$ jobs in two consecutive batches of $2k$ and 2. The first

$2k$ jobs all have weight $\frac{1}{3k^2}$ and the last two have weight $\frac{1}{3}$. It is easy to see that there are two optimal solutions: either assign the first batch to machine 1 and balance the other batch on the last 2 machines, or balance the first batch among the first 2 machines, and assign the second batch to the last machine. In either case the cost of the solution is $\frac{1}{3k^2} \cdot \frac{1}{2}(2k+1)(2k) + 2 \cdot \frac{1}{3} = \frac{1}{3k} + \frac{4}{3} = 2 \cdot \frac{1}{3k^2} \cdot \frac{1}{2}k(k+1) + 3 \cdot \frac{1}{3} = O(1)$.

Now suppose we have a convex combination weighting both these solutions equally, and try to randomly round the jobs to the machines accordingly. Both the last k jobs of the first batch and first job of the last batch are scheduled on the middle machine, half of the time. Hence, the expected completion time of the first job in the second batch is at least $\frac{1}{2}k$ and since its weight is $\frac{1}{3}$, the cost of the schedule will be $\Omega(k)$, while the number of jobs is linear in k.

It is apparent from this example that independent rounding of any LP that assigns jobs to machines is not going to be fruitful. This is even the case when all processing times are 1, and with only 3 machines.

Removing machines We now give an example that shows that decreasing the number of available machines from m to $m-1$ may increase the cost of the solution by an arbitrarily large factor. This holds even when all processing times are 1.

Take an instance with unit processing times, m machines, and $n = k^m + \cdots + k$ jobs divided into m consecutive batches of increasing total weight and decreasing cardinality. In particular, batch i contains k^i jobs with weight $\frac{1}{k^{2i}}$. An optimal Smith-monotone schedule simply puts every batch i on its own machine i. The cost of this solution is $\sum_{i=1}^{m} k^i(k^i+1)/2 \cdot \frac{1}{k^{2i}} \approx \frac{1}{2}m$.

Now suppose we decrease the number of machines from m to $m-1$. By the pigeonhole principle, there must be a machine that contains a $\frac{1}{m-1}$ fraction of the jobs of 2 different batches. Call the fraction of jobs coming from the earlier (respectively later) of the 2 batches S (respectively S'), and suppose S' comes from a batch with k^i jobs. Thus, $|S'| = \frac{1}{m-1}k^i$, $|S| \geq \frac{1}{m-1}k^{i+1}$ and the weight of those jobs in S' is $\frac{1}{k^{2i}}$. Since the jobs in S are scheduled before those in S', all the completion times of jobs in S' are at least $|S| = \frac{1}{m-1}k^{i+1}$, and therefore the total cost of jobs in S' is at least $\frac{1}{k^{2i}} \frac{1}{(m-1)^2} k^i k^{i+1} = \Omega(\frac{k}{m^2}) = \Omega(\frac{k}{m^3}\text{OPT})$. Since we can make k as large as we want, this completes the argument.

Summary

Three chapters of this thesis are based on published papers. One chapter contains unpublished results.

In Chapter 1 we study the masked VPN problem. This problem deals with routing uncertain and varying traffic patterns in a network. We are given a set of terminals in a network G, together with a second graph H on the terminals (the *mask* graph) that encodes which terminal pairs might communicate with eachother. Every terminal has a standardized, unit capacity connection with the network it may use to communicate with other terminals at any time. For every communicating terminal pair (according to H), we need to pick a connecting path through the network G. Subsequently we must install sufficient capacity in the network such that every traffic pattern that is feasible under these constraints can be routed via the predetermined paths. The masked VPN problem asks for a minimum cost solution to this problem.

The masked VPN problem generalizes the VPN problem, which can be seen as a special case where every terminal pair communicates, or H is a clique. This problem has an elegant polynomial time solution that looks like a star embedded into the network.

While such results are out of the question for general H (when H is star, the problem reduces to the APX-hard Steiner tree problem), we establish an analogous theorem for H a cycle. In this case, the optimal solution looks like a cycle with spokes embedded into the network, a topology that can be optimized in polynomial time using dynamic programing. We also show that the problem is solvable in polynomial time for H a tree of bounded degree, generalizing a known result for Steiner tree with a bounded number of terminals.

In Chapter 2 we look at subadditive inventory problems, including the well-studied inventory routing problem (IRP) and the submodular joint replenishment problem (SJRP). In subadditive inventory problems, N nodes need to be served over some finite time horizon $1, ..., T$. Multiple nodes may be served simultaneously, and the cost for this is expressed by some subadditive ordering cost function. Demand can be served exactly on time, or at any point prior to that, in which case a holding cost needs to be paid for the intervening period. The goal is to balance these cost factors so as to minimize the total cost of the schedule. The IRP is the special case of this problem where the ordering cost are induced by the cost of a minimum length route serving all nodes from some depot.

The submodular joint replenishment problem is the case where the ordering cost function is submodular. We provide the first sublogarithmic approximation algorithm for these two problems, both attaining an approximation ratio of $O(\log\log\min(N,T))$. This improves the previous best known $O(\log N)$,

$O(\log T)$ approximation ratios (using two different algorithms), as well as the $O(\log T/\log\log T)$ approximation algorithm for the special case of polynomial holding cost. We also provide general purpose results for the subadditive inventory problem, bounding the length of the time horizon in terms of the number of nodes N, unifying approximation ratios for both parameters in a single algorithm.

In Chapter 3 we study replenishment problems with fixed turnover times (RFTT). Here, we consider a metric with client that need to be replenished (repeatedly) over an infinite time horizon. After each replenishment, the nodes will need to be revisited within a fixed time period. Consequently, the effect of visiting a client early propagates, i.e. the next visit will also need to happen earlier. This property differentiates the RFTT from models typically studied in literature. The study of the RFTT is motivated by applications in practice for which the fixed turnover time is more natural, for example the replenishment of ATMs. After every replenishment an ATM has a full stock of cash, which starts depleting at a steady rate immediately. The ATM will need to be revisited before it runs out again.

We consider the objectives of minimizing the average distance per day as well as the maximum distance per day. We provide constant factor approximation for both objectives when distance is given by a tree metric, and logarithmic approximations for the general case.

In Chapter 4 we look at fixed order scheduling on parallel machines. In this setting, jobs with processing times and weights need to be processed on identical machines so as to minimize the total weighted completion time. The caveat is that the jobs arrive with a predetermined order, and this order must be respected on every machine (but not across machines). This prevents us from processing the jobs according to Smith's ratio on the individual machines, as would have been optimal otherwise.

We derive a constant factor approximation for this problem and proof a number of structural results along the way. A powerful concept here is a natural partial order we can define on the jobs, found as the intersection of the input ordering and the order of the Smith ratios. We can show that there is always an optimal schedule whose machine assignment is monotone with respect to this partial order. This drastically reduces the degrees of freedom one has in setting up a schedule. Not only does this make reasoning about the problem much simpler, we also show that by dropping some terms from the objective (at the loss of a constant factor), the problem becomes polynomial time solvable. Additionally we give a QPTAS for the special case of unit processing times, and provide some evidence that techniques previously successful on other scheduling problems with completion time objective are bound to fail on this problem.

Bibliography

[1] Ali Allahverdi. "The third comprehensive survey on scheduling problems with setup times/costs". In: *European Journal of Operational Research* 246(2) (2015), pp. 345–378.

[2] Ali Allahverdi, Jatinder N.D Gupta, and Tariq Aldowaisan. "A review of scheduling research involving setup considerations". In: *Omega* 27(2) (1999), pp. 219–239.

[3] Ali Allahverdi, C.T. Ng, T.C.E. Cheng, and Mikhail Y. Kovalyov. "A survey of scheduling problems with setup times or costs". In: *European Journal of Operational Research* 187(3) (2008), pp. 985–1032.

[4] A Altın, Edoardo Amaldi, Pietro Belotti, and MÇ Pınar. "Provisioning virtual private networks under traffic uncertainty". In: *Networks* 49(1) (2007), pp. 100–115.

[5] Aaron Archer, MohammadHossein Bateni, MohammadTaghi Hajiaghayi, and Howard Karloff. "Improved approximation algorithms for prize-collecting Steiner tree and TSP". In: *SIAM journal on computing* 40(2) (2011), pp. 309–332.

[6] Esther Arkin, Dev Joneja, and Robin Roundy. "Computational complexity of uncapacitated multi-echelon production planning problems". In: *Operations Research Letters* 8(2) (1989), pp. 61–66.

[7] Nikhil Bansal, Aravind Srinivasan, and Ola Svensson. "Lift-and-round to improve weighted completion time on unrelated machines". In: *Proceedings of the 48th Annual ACM Symposium on Theory of Computing*. 2016, pp. 156–167.

[8] Sanjoy K. Baruah, Rodney R. Howell, and Louis E. Rosier. "Feasibility Problems for Recurring Tasks on one Processor". In: *Theor. Comput. Sci.* 118(1) (1993), pp. 3–20.

[9] Sanjoy Baruah and Joël Goossens. "Scheduling real-time tasks: Algorithms and complexity". In: *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. CRC Press, Boca Raton, 2003. Chap. 28.

[10] Sanjoy Baruah, Louis Rosier, Igor Tulchinsky, and Donald Varvel. "The complexity of periodic maintenance". In: *Proceedings of the International Computer Symposium*. 1990, pp. 315–320.

[11] Walid Ben-Ameur and Hervé Kerivin. "New Economical Virtual Private Networks". In: *Commun. ACM* 46(6) (2003), pp. 69–73.

[12] Marcin Bienkowski, Jaroslaw Byrka, Marek Chrobak, Łukasz Jeż, Dorian Nogneng, and Jiří Sgall. "Better approximation bounds for the joint replenishment problem". In: *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*. SIAM. 2014, pp. 42–54.

[13] Vincenzo Bonifaci and Alberto Marchetti-Spaccamela. "Feasibility Analysis of Sporadic Real-Time Multiprocessor Task Systems". In: *Algorithmica* 63(4) (2012), pp. 763–780.

[14] Thomas Bosman, Dario Frascaria, Neil Olver, René Sitters, and Leen Stougie. "Fixed-Order Scheduling on Parallel Machines". In: *International Conference on Integer Programming and Combinatorial Optimization*. Springer. 2019, pp. 88–100.

[15] Thomas Bosman and Neil Olver. "Exploring the Tractability of the Capped Hose Model". In: *25th Annual European Symposium on Algorithms (ESA 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik. 2017.

[16] Thomas Bosman, Martijn Van Ee, Yang Jiao, Alberto Marchetti-Spaccamela, R Ravi, and Leen Stougie. "Approximation algorithms for replenishment problems with fixed turnover times". In: *Latin American Symposium on Theoretical Informatics*. Springer. 2018, pp. 217–230.

[17] Jaroslaw Byrka, Fabrizio Grandoni, Thomas Rothvoß, and Laura Sanità. "An improved LP-based approximation for Steiner tree". In: *Proceedings of the forty-second ACM symposium on Theory of computing*. ACM. 2010, pp. 583–592.

[18] Jarosław Byrka, Fabrizio Grandoni, Thomas Rothvoss, and Laura Sanità. "Steiner tree approximation via iterative randomized rounding". In: *Journal of the ACM (JACM)* 60(1) (2013), p. 6.

[19] Chris Calabro, Russell Impagliazzo, Valentine Kabenets, and Ramamohan Paturi. "The complexity of Unique *k*-SAT: An Isolation Lemma for *k*-CNFs". In: *Journal of Computer and System Sciences* 74(3) (2008), pp. 386–393.

[20] Mee Yee Chan and Francis Y. L. Chin. "General schedulers for the pinwheel problem based on double-integer reduction". In: *IEEE Transactions on Computers* 41(6) (1992), pp. 755–768.

[21] Mee Yee Chan and Francis Y. L. Chin. "Schedulers for larger classes of pinwheel instances". In: *Algorithmica* 9(5) (1993), pp. 425–462.

[22] Chandra Chekuri and Alina Ene. "Approximation algorithms for submodular multiway partition". In: *52nd Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE. 2011, pp. 807–816.

[23] Chandra Chekuri and Alina Ene. "Submodular cost allocation problem and applications". In: *International Colloquium on Automata, Languages, and Programming*. Springer. 2011, pp. 354–366.

[24] Maurice Cheung, Adam N Elmachtoub, Retsef Levi, and David B Shmoys. "The submodular joint replenishment problem". In: *Mathematical Programming* 158(1-2) (2016), pp. 207–233.

[25] Leandro C Coelho, Jean-François Cordeau, and Gilbert Laporte. "Thirty years of inventory routing". In: *Transportation Science* 48(1) (2013), pp. 1–19.

[26] Sofie Coene, Frits C. R. Spieksma, and Gerhard J. Woeginger. "Charlemagne's Challenge: The Periodic Latency Problem". In: *Operations Research* 59(3) (2011), pp. 674–683.

[27] Holger Dell, Thore Husfeldt, Dániel Marx, Nina Taslaman, and Martin Wahlén. "Exponential Time Complexity of the Permanent and the Tutte Polynomial". In: *ACM Transactions on Algorithms* 10(4) (2014), 21:1–21:32.

[28] Stuart E Dreyfus and Robert A Wagner. "The Steiner problem in graphs". In: *Networks* 1(3) (1971), pp. 195–207.

[29] N. G. Duffield, Pawan Goyal, Albert Greenberg, Partho Mishra, K. K. Ramakrishnan, and Jacobus E. van der Merwe. "A flexible model for resource management in virtual private networks". In: *Proc. SIGCOMM*. 1999, pp. 95–108.

[30] Friedrich Eisenbrand, Fabrizio Grandoni, Gianpaolo Oriolo, and Martin Skutella. "New Approaches for Virtual Private Network Design". In: *SIAM J. Comput.* 37(3) (2007), pp. 706–721.

[31] Friedrich Eisenbrand, Nicolai Hähnle, Martin Niemeier, Martin Skutella, José Verschae, and Andreas Wiese. "Scheduling periodic tasks in a hard real-time environment". In: *Proceedings of the 37th International Colloquium on Automata, Languages, and Programming*. Springer. 2010, pp. 299–311.

[32] Friedrich Eisenbrand and Edda Happ. "Provisioning a virtual private network under the presence of non-communicating groups". In: *Italian Conference on Algorithms and Complexity*. Springer. 2006, pp. 105–114.

[33] Pontus Ekberg and Wang Yi. "Schedulability analysis of a graph-based task model for mixed-criticality systems". In: *Real-Time Systems* 52(1) (2016), pp. 1–37.

[34] Alina Ene, Jan Vondrák, and Yi Wu. "Local distribution and the symmetry gap: Approximability of multiway partitioning problems". In: *Proceedings of the twenty-fourth annual ACM-SIAM symposium on Discrete algorithms*. SIAM. 2013, pp. 306–325.

[35] Hanhua Feng, Vishal Misra, and Dan Rubenstein. "Optimal state-free, size-aware dispatching for heterogeneous M/G/-type systems". In: *Performance Evaluation* 62(1) (2005), pp. 475–492.

[36] J Andrew Fingerhut, Subhash Suri, and Jonathan S Turner. "Designing Least-Cost Nonblocking Broadband Networks". In: *J. Algorithms* 24(2) (Aug. 1997), pp. 287–309.

[37] Samuel Fiorini, Gianpaolo Oriolo, Laura Sanità, and Dirk Oliver Theis. "The VPN problem with concave costs". In: *SIAM J. Discrete Math.* 24(3) (2010), pp. 1080–1090.

[38] Peter C. Fishburn and Jeffrey C. Lagarias. "Pinwheel scheduling: Achievable densities". In: *Algorithmica* 34(1) (2002), pp. 14–38.

[39] Alexandre Fréchette, F. Bruce Shepherd, Marina K. Thottan, and Peter J. Winzer. "Shortest Path versus Multi-Hub Routing in Networks with Uncertain Demand". In: *Proc. INFOCOM*. 2013, pp. 710–718.

[40] Greg N. Frederickson, Matthew S. Hecht, and Chul E. Kim. “Approximation algorithms for some routing problems”. In: *Proceedings of the 17th International Symposium on Foundations of Computer Science*. 1976, pp. 216–227.

[41] Satoru Fujishige. *Submodular functions and optimization*. Vol. 58. Elsevier, 2005.

[42] Takuro Fukunaga, Afshin Nikzad, and R Ravi. “Deliver or hold: approximation algorithms for the periodic inventory routing problem”. In: *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM)*. 2014.

[43] Michael R Garey and David S Johnson. *Computers and Intractability. A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman, 1979.

[44] James Gleick. *Genius: The life and science of Richard Feynman*. Vintage, 1993.

[45] Michel X Goemans and David P Williamson. “A general approximation technique for constrained forest problems”. In: *SIAM Journal on Computing* 24(2) (1995), pp. 296–317.

[46] Navin Goyal, Neil Olver, and F. Bruce Shepherd. “The VPN Conjecture Is True”. In: *J. ACM* 60(3) (2013), 17:1–17:17.

[47] Ronald L Graham, Eugene L Lawler, Jan Karel Lenstra, and AHG Rinnooy Kan. “Optimization and Approximation in Deterministic Sequencing and Scheduling: a Survey”. In: *Discrete Optimization II*. Vol. 5. Annals of Discrete Mathematics. Elsevier, 1979, pp. 287–326.

[48] Fabrizio Grandoni, Volker Kaibel, Gianpaolo Oriolo, and Martin Skutella. “A Short Proof of the VPN Tree Routing Conjecture on Ring Networks”. In: *Oper. Res. Lett.* 36(3) (2008), pp. 361–365. eprint: 0710.3044.

[49] Fabrizio Grandoni, Thomas Rothvoß, and Laura Sanità. “From uncertainty to nonlinearity: Solving virtual private network via single-sink buy-at-bulk”. In: *Mathematics of Operations Research* 36(2) (2011), pp. 185–204.

[50] Anupam Gupta, Amit Kumar, Martin Pál, and Tim Roughgarden. “Approximation via cost sharing: Simpler and better approximation algorithms for network design”. In: *J. ACM* 54(3) (2007), p. 11.

[51] Mor Harchol-Balter. *Performance Modeling and Design of Computer Systems: Queueing Theory in Action*. 1st. New York, NY, USA: Cambridge University Press, 2013.

[52] Mor Harchol-Balter, Mark E. Crovella, and Cristina D. Murta. “On Choosing a Task Assignment Policy for a Distributed Server System”. In: *IEEE Journal of Parallel and Distributed Computing* 59(2) (1999), pp. 204–228.

[53] Kunihiko Hiraishi, Eugene Levner, and Milan Vlach. “Scheduling of parallel identical machines to maximize the weighted number of just-in-time jobs”. In: *Computers and Operations Research* 29(7) (2002), pp. 841–848.

[54] Robert Holte, Al Mok, Louis Rosier, Igor Tulchinsky, and Donald Varvel. "The pinwheel: A real-time scheduling problem". In: *Proceedings of the 22th Annual Hawaii International Conference on System Sciences*. Vol. 2. 1989, pp. 693–702.

[55] Cor AJ Hurkens, Judith Cornelia Maria Keijsper, and Leen Stougie. "Virtual Private Network Design: A Proof of the Tree Routing Conjecture on Ring Networks". In: *SIAM J. Discrete Math.* 21(2) (2007), pp. 482–503.

[56] Jennifer Iglesias, Rajmohan Rajaraman, R. Ravi, and Ravi Sundaram. "Designing Overlapping Networks for Publish-Subscribe Systems". In: *Proc. APPROX*. 2015, pp. 381–395.

[57] Tobias Jacobs and Salvatore Longo. "A new perspective on the windows scheduling problem". In: *arXiv preprint arXiv:1410.7237* (2014).

[58] Kamal Jain. "A factor 2 approximation algorithm for the generalized Steiner network problem". In: *Combinatorica* 21(1) (2001), pp. 39–60.

[59] Tsuyoshi Kawaguchi and Seiki Kyan. "Worst case bound of an LRF schedule for the mean weighted flow-time problem". In: *SIAM Journal on Computing* 15(4) (1986), pp. 1119–1129.

[60] Lap Chi Lau, Ramamoorthi Ravi, and Mohit Singh. *Iterative methods in combinatorial optimization*. Vol. 46. Cambridge University Press, 2011.

[61] Eugene. L. Lawler. "Sequencing jobs to minimize total weighted completion time subject to precedence constraints". In: *Annals of Discrete Mathematics* 2 (1978), pp. 75–90.

[62] Retsef Levi, Robin O Roundy, and David B Shmoys. "Primal-dual algorithms for deterministic inventory problems". In: *Mathematics of Operations Research* 31(2) (2006), pp. 267–284.

[63] Retsef Levi, Robin Roundy, David Shmoys, and Maxim Sviridenko. "A constant approximation algorithm for the one-warehouse multiretailer problem". In: *Management Science* 54(4) (2008), pp. 763–776.

[64] Laszlo Lovász, Martin Grotschel, and Alexander Schrijver. *Geometric algorithms and combinatorial optimization*. Springer, 1988.

[65] Viswanath Nagarajan and Cong Shi. "Approximation algorithms for inventory problems with submodular or routing costs". In: *Mathematical Programming* 160(1-2) (2016), pp. 225–244.

[66] Neil Olver. "A note on hierarchical hubbing for a generalization of the VPN problem". In: *Oper. Res. Lett.* 44(2) (2016), pp. 191–195.

[67] Neil Olver and F Bruce Shepherd. "Approximability of robust network design". In: *Proc. SODA*. 2010, pp. 1097–1105.

[68] Maurice Queyranne and Andreas S Schulz. "Approximation bounds for a general class of precedence constrained parallel machine scheduling problems". In: *SIAM Journal on Computing* 35(5) (2006), pp. 1241–1253.

[69] Harald Räcke, Christian Sohler, and Matthias Westermann. "Online Scheduling for Sorting Buffers". In: *Proceedings of 10th Annual European Symposium on Algorithms*. Ed. by Rolf Möhring and Rajeev Raman. 2002, pp. 820–832.

[70] Andreas S. Schulz and Martin Skutella. “Scheduling Unrelated Machines by Randomized Rounding”. In: *SIAM Journal on Discrete Mathematics* 15(4) (2002), pp. 450–469.

[71] Andreas S Schulz and Martin Skutella. “The power of α-points in preemptive single machine scheduling”. In: *Journal of Scheduling* 5(2) (2002), pp. 121–133.

[72] Petra Schuurman and G Woeginger. “Approximation schemes-a tutorial”. In: *Lectures on scheduling* (2000), p. I1.

[73] Uwe Schwiegelshohn. “An alternative proof of the Kawaguchi-Kyan bound for the Largest-Ratio-First rule”. In: *Operations Research Letters* 39(4) (2011), pp. 255–259.

[74] Martin Skutella. “Convex quadratic and semidefinite programming relaxations in scheduling”. In: *Journal of the ACM* 48 (2001), pp. 206–242.

[75] Martin Skutella and Gerhard J Woeginger. “A PTAS for minimizing the total weighted completion time on identical parallel machines”. In: *Mathematics of Operations Research* 25(1) (2000), pp. 63–75.

[76] Marius M Solomon. “The minimum spanning tree problem with time window constraints”. In: *American Journal of Mathematical and Management Sciences* 6(3-4) (1986), pp. 399–421.

[77] Harvey M Wagner and Thomson M Whitin. “Dynamic version of the economic lot size model”. In: *Management Science* 5(1) (1958), pp. 89–96.

[78] David P Williamson and David B Shmoys. *The design of approximation algorithms*. Cambridge university press, 2011.

List of Figures

An introduction to combinatorial optimization

This chapter is meant as a gentle introduction to the topics in this thesis, targeted towards readers with no background in computer science, mathematics, operations research or any other technical field for that matter. After a rough sketch of the problems that the core chapters are trying to solve[1], we will explore their common theme: *combinatorial optimization*. I have tried to give a taste of what this term entails though examples and pictures, introducing some of the important concepts from my field in the process.

[1] In the interest of accessibility, I have simplified and left out some details. For an accurate description of the results, I refer to the introductory sections of the chapters.

On the application areas

Chapter 1 A virtual private network (VPN) provider offers a service to clients that allows digital devices in multiple locations — servers, laptops, smartphones etc. — to *virtually* connect as if they were part of the same local network. To guarantee quality of service it needs to install connections with sufficient bandwidth to carry data between all devices.

A crude solution would be to reserve some bandwidth for every connection a client might potentially make, but this does not take into account some basic restrictions. Every device has a maximum download and upload rate (based on hardware limitations, or their internet subscription for example) and this means that if a device sends a lot of data to A, it has cannot send as much data to B.

By making clever use such knowledge, capacity can be reused for multiple connections. This reduces the amount of bandwidth required (which is expensive) and therefore the cost of the network. Chapter 1 looks at algorithms for calculating how to do this most effectively.

Chapters 2 and 3 In inventory management — think stocking supermarket warehouses with enough soup cans to serve demand — there is an interesting dynamic between the two quantities that largely drive expenses: square meters of storage space, and litres of fuel.

Ideally, inventory would arrive just as it is about to sell out, going from truck to shelf directly. But this would require deliveries to be made constantly. From a transportation point of view sending everything at once and far ahead of time would be more sensible, but this raises the required storage space, an expensive proposition in today's urban areas with their ever rising property prices. Chapters 2 and 3 contain algorithms to balance these forces and generate cheaper restocking schedules.

The former chapter considers a specific scenario, where inventory needs to be replenished within a predictable number of days after each delivery, motivated by an application in cash replenishment at ATM's. The latter chapter studies a more general model, providing high level procedures usable in a wider array of inventory settings.

Chapter 4 Imagine a small workshop with multiple production lines, that just received a batch of job orders. The orders have different priorities, and ideally the high priority orders get completed first. However, job orders arrive sequentially at the workshop and there is no time/space to reorder them before sending them to the machines, causing low priority orders that arrive early to delay high priority orders that come in later.

We can assign the jobs to the different production lines as we see fit, though. Is there a good way to exploit this? A reasonable sounding approach is to reserve some production lines for high priority jobs, trading off idle time for more flexibility. Our analysis shows that this is indeed a good idea and distills the intuition in an algorithm for computing efficient production schedules in Chapter 4.

Introduction to combinatorial optimization

THE RESEARCH presented in this dissertation is all motivated in one way or another by the study of optimal planning and logistics. Its application area is only half the story though. The technical challenges involved are at least as interesting, in my opinion. A concise description of the technical contribution of my thesis would be something like:

The papers bundled in this thesis contain proofs of results about the complexity of, and worst case performance guarantees of algorithms for, combinatorial optimization problems in the area of network design and scheduling.

Unfortunately, it is not easy to translate this summary into more everyday language; at least not without losing a lot of the meaning. Instead, I have tried to illustrate some of the jargon with examples.

THE WORD PROBLEM means something very specific to computer scientists and forms a natural companion to the world *algorithm*. A problem is simply a well defined set of requirements, that can potentially be met by multiple solutions. In the sort of problems I work on, people are typically interested in getting the best among all possible solutions, or maybe a very good one, although oftentimes finding any solution that meets the requirements is all that is needed. An algorithm on the other hand, is a well defined list of instruction on how to find a (best/good/any) solution to a given problem. That is all really. Many important algorithms involve a lot of complicated math and computer code, but in principle that part is optional.

One example of an algorithm is Feynman's general purpose problem-solving algorithm[2]. It's named after famous 20th century physicist Richard Feynman [44, pg. 315].

[2] This one helped me produce most of the work in this thesis, incidentally.

Algorithm 1: Feynman's Algorithm

1. Write down the problem.
2. Think very hard.
3. Write down the solution.

Though this algorithm is mainly known for its satirical value, it perfectly illustrates the core concept of an algorithm. This particular algorithm also highlights of one of the most important factors in the design of algorithms: the *running time*, i.e. the time it takes to completely execute the instructions of the algorithm. By the looks of it, it might take a long time to finish step 2. We say the algorithm is not *efficient*. In fact this algorithm could take an infinite amount of time. Ideally we would like our algorithms to be fast.

Algorithm 2: Feynman's Heuristic

1. Write down the problem.
2. Think very hard for 10 minutes.
3. Write down the best solution you found.

Feynman's heuristic takes care of the running time problem, but creates a new one. Who is to say that we have found a reasonable solution after just 10 minutes of thinking? The algorithm does not have a *performance guarantee*. This is typical of a class of algorithms called *heuristics*. When they work, they are fast, but you can never really be sure they will. Ideally, we would like an algorithm to combine the speed of heuristics with a good performance guarantee. Such an algorithm is called an *approximation algorithm*, and is the main type of algorithm studied in this thesis. We will revisit this type of algorithm later in the chapter.

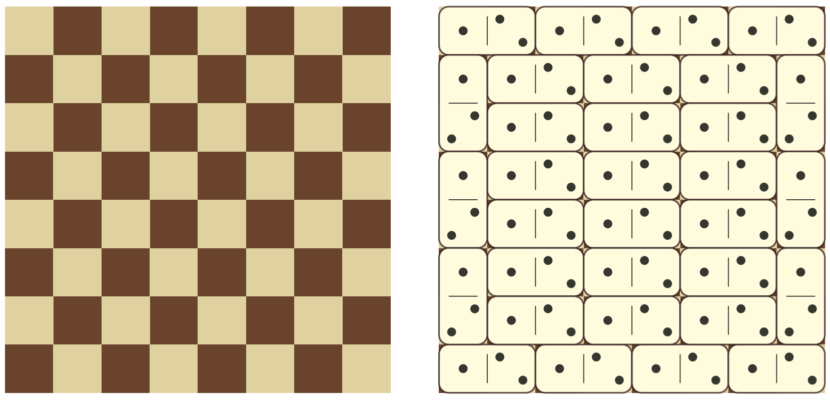


Figure 1: Covering a chess board with dominoes.

COMBINATORIAL PROBLEMS arise when you have a small collection of objects that can be combined or structured in many different ways. For example, suppose you have a chess board, an 8×8 grid of alternating dark and light squares, and 32 identical dominoes that fit exactly on two adjacent squares of the chess board. You can cover the entire chessboard perfectly with the dominoes such that no piece sticks out or overlaps another. There are many,

many ways you can do this. In fact, there are well over 50 quadrillion (50 million billion, a 5 with 16 zeroes) ways. This means that if every human that ever lived would have produced one new pattern to cover the chessboard for every day of their life, we would not be close to exhausting all possibilities yet [3].

[3] There are about 12.99 million ways to tile a chessboard with 2x1 rectangles. Since the orientation of the dominoes is distinguishable, we multiply by $2^{32} \approx 4.2 \cdot 10^9$ for a grand total of over $55 \cdot 10^{15}$. An estimated 108 billion people have ever lived. Even if we work with an optimistic life expectancy of a 100 years, or about 37 thousand days, this sums to less than $4 \cdot 10^{15}$ days.

An interesting question is what happens if we chop off one corner from the chessboard (see Figure 2). How many ways are there to perfectly cover the board now?

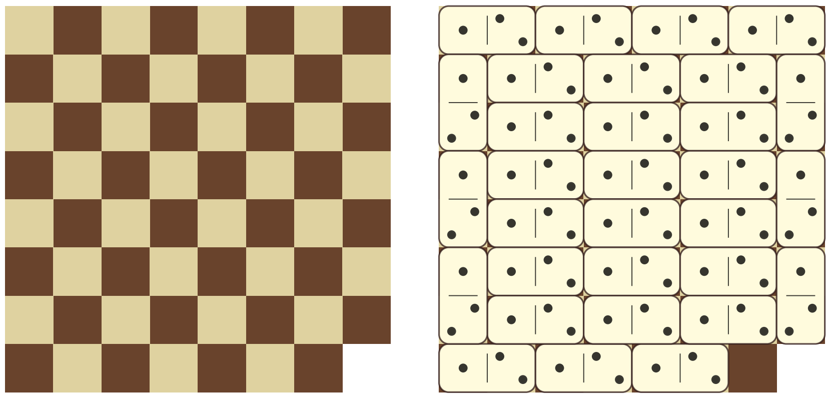


Figure 2: Covering a chess board with one corner cut off.

If you play around with it for a while you'll probably figure out that the answer is 0. Suppose you want to be absolutely sure that it is truly **impossible** to perfectly cover the board. You could go through all possible patterns and see if you always end up with an uncovered square, but given how many ways there are to do this, this will likely take too long.

A better way to look at it is this: as every domino is made out of two squares, any pattern of dominoes that does not overlap or stick out on the sides must cover an even number of squares of the chessboard. Because we removed a single corner from the board though, the board has an odd number of squares, meaning we can never cover all of them in this way.

THIS TYPE OF ARGUMENTATION IS CALLED A PROOF. Even though we didn't try every possibility, we can be sure none of them work out. This makes a proof fundamentally different from *evidence*. Every pattern that does not work out is evidence that covering the board is impossible, but evidence can only give us some confidence that the statement might be true. On the other hand, a proof completely rules out the possibility that evidence to the contrary exists. So, in a way we have just compressed 50 quadrillion pieces of evidence into a three sentence argument.

Proofs are what define theoretical research[4], the type of research included in this thesis. The idea of establishing something with absolutely certainty is of course a very attractive proposition, which begs the question why not all research is theoretical. The problem is that proving things can be much harder than merely finding evidence, and in practice not all things that are true can be proven[5].

[4] In computer science, at least. It should be noted that the word theory has a different connotation outside the exact sciences.

[5] There is a mathematical way to define this statement, but I mean it in a soft sense here.

Theoretical guarantees can be very conservative compared to what the answer should be intuitively (we will see a real life example of this later), and this is a big drawback of theoretical work. On the other hand, theoretical results can be strengthened as time goes on, and some of the work in this thesis is of that flavour. It should also be remarked that sometimes we cannot prove things *precisely because they are not true*, and in that case it might be good to know why.

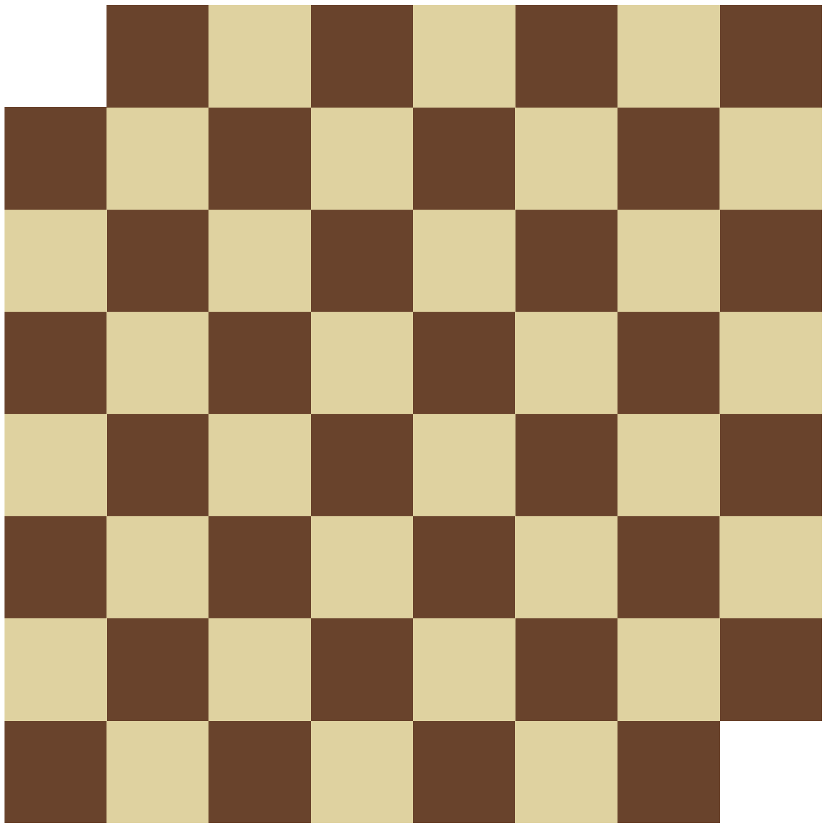


Figure 3: A chess board with two corners cut off. Can you tile it with dominoes?

A final variant of this chessboard problem is show in Figure 3. Suppose we cut off opposite corners of the board. Is there a perfect covering? If not, can you give a proof? I leave it to the interested readers to try this one for themselves. A solution is given at the end of the chapter but you may want to look at the hint in the sidenote first[6].

[6] Though the colors of the squares do not matter for the answer, they do highlight an important structure within the chessboard. What makes the two cut off squares special? What about the two squares covered by a domino?

COMBINATORIAL OPTIMIZATION PROBLEMS form a special class of combinatorial problem which require us to find the best solution according to some criterion. So, the cheapest, quickest, shortest, most profitable, safest etc. The chessboard problems presented in the previous paragraphs are combinatorial but not really optimization problems.

We will now look at a real world optimization problem anyone who has ever flown to the European union may have been confronted with. At the time of writing, customs regulations stipulate that anyone entering the EU through an airport may import at most 430 EUR worth of goods free of tax[7]. Individual items cannot be split, meaning that if you have, say, three items of 300, 100 and 50 euros, the first two items can be imported freely, but the 50 EUR item will be taxed for its entire value. Even though you only imported $300 + 100 = 400$ euros worth of goods, the remaining 30 euros of tax free allowance cannot be deducted from the value of the third item and is effectively lost. Suppose now that you are carrying a lot of items. Which items exactly should you count towards your limit so as to maximize the total value you can declare tax free?

[7] Retrieved from: `https://ec.europa.eu/taxation_customs/individuals/travelling/entering-eu_en`.

This problem is called the *knapsack problem*, and it is a well-known problem[8] that belongs to the *NP-hard* class of problems. This essentially means that there is likely no efficient algorithm for it[9]. If you really want to be sure that you get the right answer every time, you might as well run Feynman's algorithm, as there is no substantially faster way to solve it that try every possibility and take the best one. If you value your time this is not an appealing idea.

[8] Well-known to computer scientists that is, but perhaps not to lawmakers.

[9] This is widely assumed to be true, but hinges on resolving the P versus NP problem, one of the most important open problems in mathematics and computer science.

There is a compromise though, in the form of an *approximation* algorithm. It is fast to execute and guaranteed to be at least somewhat close to the optimal solution every time.

Algorithm 3: Greedy algorithm airport knapsack

```
1. Sort the items from high to low value
2. Go through your items one by one in this order
3. Add every item to the set of tax free items if it does not cause the total
   value to exceed 430 EUR
```

In the case of the earlier example we would add the items of 300 and 100, but not the item of 50 since that would make the total value jump from 400 to 450.

So how well does this algorithm perform? You can prove that the best possible solution never achieves more than twice as much tax free value as the solution produced by Algorithm 3, regardless of how many items you are carrying. As such it is a *constant factor* approximation algorithm. Constant, because the performance guarantee does not get worse if you add more items.

Is the bound of twice the value too conservative? It feels weak but imagine this scenario: you are carrying three items, one of value 216 euros and two of value 215 euros. If we run Algorithm 3, we pick the item of 216 first but then we cannot add any more items. Clearly we would have been happier had we picked the two items of 215, since then we would have utilized the full 430 tax free allowance, almost twice as much as 216.

Once you have seen the value of the items, finding the optimal set of items to declare in this 'bad' example is not particularly difficult, of course. And by making the algorithm more sophisticated the performance guarantee can be improved a lot, the details of which are out of scope for now. The point is that you want to be sure that you covered all your bases, so that you do not run into any nasty surprises, and this is what the theory of approximation algorithms is all about.

Solution to chessboard problem in Figure 3: even though the number of squares is even, there is no way to cover the board perfectly. Since there are two light squares missing, and every domino covers a dark and a light square, we will always have two dark squares left over that we cannot cover.