Non-clairvoyant problems with Deadlines or Delay

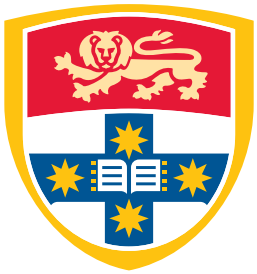
MAI LE
SID: 480398998

Supervisor: Dr. William Umboh

This thesis is submitted in partial fulfillment of
the requirements for the degree of
Bachelor of Advanced Computing (Honours)

School of Computer Science
The University of Sydney
Australia

14 November 2021



THE UNIVERSITY OF SYDNEY

Student Plagiarism: Compliance Statement

I certify that:

I have read and understood the University of Sydney Student Plagiarism: Coursework Policy and Procedure;

I understand that failure to comply with the Student Plagiarism: Coursework Policy and Procedure can lead to the University commencing proceedings against me for potential student misconduct under Chapter 8 of the University of Sydney By-Law 1999 (as amended);

This Work is substantially my own, and to the extent that any part of this Work is not my own I have indicated that it is not my own by Acknowledging the Source of that part or those parts of the Work.

Name: Mai Le

Signature:  **Date**: 14 November 2021

Abstract

Most online problems with deadlines or delay assume clairvoyance, which is defined as the ability to see the deadline or future delay of pending requests. The non-clairvoyant variant has only been studied for the set cover with delay problem (Azar et al., 2020). In this thesis, we explore the power of clairvoyance for the joint replenishment problem (JRP) and the multi-level aggregation problem (MLAP).

On the lower end, we show that the competitive ratio of any non-clairvoyant algorithm for JRP is at least $\sqrt{n}$. This is in contrast to the best known clairvoyant algorithms for JRP, which all have constant competitive ratios. On the upper end, we present $O(\sqrt{n})$-competitive non-clairvoyant algorithms for JRP with deadlines and JRP with delay, which are optimal up to a constant factor. We also give an $O(\sqrt{n} + D)$-competitive non-clairvoyant algorithm for MLAP with deadlines.

For the set cover with delay problem, we slightly improve on the only previously known non-clairvoyant algorithm (Azar et al., 2020) by derandomizing it, yielding a $O(\log n \log k)$ deterministic competitive ratio. Additionally, we show that there exists an $O(\log n)$-competitive algorithm for the clairvoyant case. This means that clairvoyance provides at least a $\log k$ factor improvement.

Acknowledgements

I would would first like to thank my supervisor, William Umboh, for reaching out to me to work on this project and for his time and valuable guidance this past year. This research would not have been possible without your help.

To Brian, thank you for your unwavering love and support, especially during stressful times.

To my parents, I'm deeply grateful for your constant support. I would not be where I am without you.

Finally, I'm thankful to God for everything in my life and for the many opportunities bestowed on me.

CONTENTS

List of Figures

CHAPTER 1

Introduction

In online minimization problems with deadlines or delay, requests are released over time. For many problems, the algorithm is initially given a weighted ground set of items and serves each request by transmitting a set of items. The set of items that would satisfy a request is problem-specific. An online algorithm is not aware of future requests.

In the deadline model, each request has a deadline it must be served by. The cost of each transmission is the sum over the weight of each item included in the transmission. We want to minimize the total cost of all transmissions.

The delay model is a generalisation of the deadline model, whereby each request is associated with a non-decreasing delay function of time. In addition to the cost of transmitting items, the algorithm also pays the delay cost of a request, which is defined as the value of the delay function when the request is served. We want to minimize the delay penalty plus the transmission cost.

Most competitive algorithms for problems of this nature assume clairvoyance. This means the online algorithm knows a request's deadline or delay function at its release time. On the other hand, a non-clairvoyant algorithm does not know a request's deadline ahead of time and is only informed when a request expires. Similarly, in the non-clairvoyant delay variant, the algorithm can only see the delay accumulated up to the current time point.

In this thesis, we consider the non-clairvoyant variant of classical online problems and analyse the competitive gap between them and their clairvoyant counterparts. The benefit of working in the non-clairvoyant setting is that we can solve a wider range of problems, where the algorithm is given little information about the future. In addition to having less information, most

techniques in literature rely heavily on the clairvoyant property in their analyses and thus do not translate to a non-clairvoyant setting.

1.1 Problems

We focus on the following problems in this thesis. All require transmitting weighted items to serve requests.

Joint replenishment (JRP): The algorithm is given a set of weighted items and a fixed ordering fee K. Each arriving request is associated with one item and is served by transmitting that item. The cost of a transmission is the sum of K plus the total cost of all items included. The ordering fee can be viewed as another item that must be included in every transmission. Buchbinder et al. (2008) present a deterministic 3-competitive algorithm for the online clairvoyant variant of this problem.

Multi-level aggregation (MLAP): The algorithm is given a node-weighted[1] rooted tree. Each request arrives on a node and is served by transmitting the path from the root to that node. Competitive algorithms exist when assuming clairvoyance. Buchbinder et al. (2017) show a $O(D)$ competitive algorithm for the deadline version, where D is the depth of the input tree. In the delay case, Azar and Touitou (2019) design a $O(D^2)$ competitive algorithm.

Set cover with delay (SCD): The algorithm is given a set cover instance ahead of time. Each request arrives on an element and is served by transmitting any set the element is in. Azar et al. (2020) present the first non-clairvoyant randomized polylog-competitive algorithm in the size of the set cover instance. The paper also presents a lower bound of $\Omega(\sqrt{\log n})$ and $\Omega(\sqrt{\log m})$ for the clairvoyant variant of the problem.

1.2 Motivation

We give one motivation for working in a non-clairvoyant setting.

Real-time data processing: Consider a network where a single server needs to deliver data to a set of clients and each client independently processes the data received. To save bandwidth,

[1] the edge-weighted variant can be efficiently reduced to node-weighted.

the server initially sends a snapshot to each client and from then on sends smaller updates that can be applied to the existing snapshot to obtain a newer snapshot.

Sometimes, the clients process data slower than the speed at which it comes in. For a client to catch up, the server needs to send a new snapshot to it.

Each client periodically informs the server of how behind it is based on the timestamp of the last update processed. The server receives this information and decides when to send a new snapshot. The server aims to minimize the number of snapshots it has to send out and the lag accumulated by each client. The non-clairvoyant aspect stems from the fact that the server cannot predict each client's lag time in the future as it could remain stable or quickly increase over time.

1.3 Contributions

In this thesis, we study the joint replenishment problem and the multi-level aggregation problem in the non-clairvoyant setting and give near-optimal algorithms to solve them. Interestingly, we prove that there exists a square root lower bound for both problems. Considering that there exists a $O(1)$-competitive clairvoyant algorithm for JRP (Buchbinder et al., 2008), the power of clairvoyance is at least a $\sqrt{n}$ factor for both JRP and MLAP.

THEOREM 1.1. *Any randomized non-clairvoyant algorithm for JRP with deadlines is $\Omega(\sqrt{n})$-competitive.*

We show that the competitive ratio of any algorithm is lower bounded for a particular JRP instance. In this instance, the adversary initially releases requests on all items and sets their deadlines to be ∞. At any point in time, the adversary sets a pending request's deadline to be the current time, forcing the algorithm to serve it immediately by starting a new transmission. The adversary repeats this as long as there remain pending requests, penalizing the algorithm for not serving requests together. On the other hand, the adversary also penalizes the algorithm for serving requests that have not reached their deadlines by keeping their deadlines at ∞.

All the following algorithms run in polynomial time unless otherwise stated.

The following upper bounds for JRP have s in the competitive ratio, which denotes the maximum number of items requested at any point in time. In an instance where requests only arrive on one item, $s = 1 \ll n$. Theorem 1.1 implies that the upper bounds are optimal up to a constant factor.

THEOREM 1.2. *There exists a $(\sqrt{s}+1)$-competitive deterministic non-clairvoyant algorithm for JRP with deadlines.*

The algorithm that gives the above theorem can be easily extended to the delay model.

THEOREM 1.3. *There exists a $O(\sqrt{s})$-competitive deterministic non-clairvoyant algorithm for JRP with delay.*

Both algorithms work by aggregating items that are 'cheap' together to save on the fixed ordering fee and serving 'expensive' items separately. The intuition behind not aggregating expensive items is that these can pay for themselves. We extend this technique to give a non-clairvoyant algorithm for MLAP with deadlines. Any JRP instance can be converted into a MLAP instance where the root node has n children, each corresponding to an item in the JRP instance.

We show that only an additive term of D is lost when working with MLAP.

THEOREM 1.4. *There exists a $O(D+\sqrt{n})$-competitive deterministic non-clairvoyant algorithm for MLAP with deadlines.*

A key optimization technique in the above algorithm is to fractionally invest into nodes and then aggregate nodes that have been fully paid for via investments.

Previous techniques used in the clairvoyant setting consist of reducing the input tree into a forest of hierarchically well-separated trees (HST) and designing competitive algorithms for HSTs. Knowing requests' deadlines or delay in advance is key in such algorithms (Buchbinder et al. 2017 and Azar and Touitou 2019). Hence these algorithms don't translate to the non-clairvoyant setting.

Additionally, our algorithm does not require reduction into HSTs, which we believe makes the analysis more intuitive.

The following result is an implication of Theorem 1.4.

THEOREM 1.5. *There exists a $O(\sqrt{n}\log n)$-competitive randomized non-clairvoyant algorithm for edge-weighted Steiner tree with deadlines.*

We can simply embed the input graph into a tree using metric tree embedding and run the algorithm for MLAP that gives Theorem 1.4. In the clairvoyant setting, the optimal competitive ratio for edge-weighted Steiner tree with either deadlines or delay is $O(\log n)$ (Azar and Touitou, 2020).

For the set cover with delay problem, we further improve the results given in (Azar et al., 2020) by de-randomizing their $O(\log k)$-competitive fractional algorithm. The method of conditional probabilities is used to achieve this (Alon and Spencer, 2008). As the delay case is a more general version of the deadline case, the result holds for set cover with deadlines.

THEOREM 1.6. *There exists a $O(\log n \log k)$-competitive deterministic non-clairvoyant algorithm for set cover with delay.*

This algorithm matches the existing lower bound of $\Omega(\log n \log m)$ (Azar et al., 2020) on any randomized polynomial time randomized algorithm for non-clairvoyant SCD. Even if we consider exponential running time algorithms, our upper bound is nearly tight as the best lower bound is $\Omega(\frac{\log n \log m}{\log n + \log m})$, which holds for most interesting values of n and m (Alon et al., 2003).

Finally, we apply the algorithm given in Azar and Touitou (2019) with some modifications to obtain a better upper bound for clairvoyant SCD.

THEOREM 1.7. *There exists a $O(\min\{\log n, \log m\})$-competitive deterministic clairvoyant algorithm for SCD which runs in exponential time.*

This bridges the competitive gap between the upper bound and the lower bound of $\Omega(\sqrt{\min\{\log n, \log m\}})$ (Azar et al., 2020) for clairvoyant SCD to a quadratic gap.

1.4 Related work

In this thesis, we provide deterministic algorithms to solve various deadline or delay problems. Deterministic algorithms must make decisions based sorely on the input provided, allowing an adversary to completely predict the algorithm's output. In contrast, a randomized algorithm also uses a stream of random data as an input. Therefore an oblivious adversary is unaware of the state of the algorithm (Ben-David et al., 1990). In many online problems, randomization proves to be a powerful tool. The online metric matching problem admits a $O(\log^2 k)$-competitive randomized algorithm but no deterministic algorithm can have a competitive ratio better than $2k - 1$ (Kalyanasundaram and Pruhs, 1993).

For the multi-level aggregation problem, we only provide an algorithm for the deadline case. Even though the deadline model is a special case of the delay model, often the best known algorithms for these problems have the same asymptotic competitive ratios in both models. This is the case for the edge-weighted Steiner tree/forest, multi-cut, edge-weighted Steiner network, node-weighted Steiner tree/forest, facility location, where all problems are in the clairvoyant setting (Azar and Touitou, 2019). However for MLAP, the best known clairvoyant algorithm for the delay model is $O(D^2)$-competitive (Azar and Touitou, 2019), whereby the best known algorithm for the deadline model is $O(D)$-competitive.

The special case of MLAP, where the input tree is a path, has been mostly settled. There is a simple 4-competitive algorithm for the deadline variant and a 5-competitive algorithm for the delay variant (Bienkowski et al., 2016). This is accompanied by a lower bound of 4 on any deterministic algorithm for the deadline variant (Bienkowski et al., 2016). The algorithms above don't require clairvoyance to make decisions.

The set cover with delay problem was first studied in (Carrasco et al., 2018) as the online set aggregation problem. The paper presents a $O(\log N)$-competitive clairvoyant algorithm, where N is the number of requests.

1.5 Outline

Chapter 2 provides some background and review of past work and techniques.

In Chapter 3, we study the multi-level aggregation problem. We first work with the easier joint replenishment problem and extend the technique to provide competitive algorithms for MLAP. A simple lower bound is given which applies for JRP and thus also MLAP.

Chapter 4 gives results for set cover with delay, in both non-clairvoyant and clairvoyant settings.

Chapter 5 concludes the thesis and highlights related open questions.

CHAPTER 2

Background

2.1 Problem definition

General model. We are given a set of items $\mathcal{U}$, with costs $c : \mathcal{U} \to \mathbb{R}^+$. Requests are released over time and can only be served after its release time. We call a request *pending* if it has been released but not yet served.

Each request q can be served by an *upwards-closed* collection of subsets $S_q \subseteq 2^{\mathcal{U}}$, that is, if $X_1 \subseteq X_2 \subseteq \mathcal{U}$ and $X_1 \in S_q$, then $X_2 \in S_q$. When the algorithm transmits a set of items $X \subseteq \mathcal{U}$, any request q such that $X \in S_q$ is served by the transmission. Items are only transmitted momentarily and cannot be reused for future transmissions.

An example problem of this abstract model is the edge-weighted Steiner tree problem. In this problem, the items $\mathcal{U}$ are the edges of the input tree. Each request q is on a node u and S_q is the set of sub-graphs in the original input graph such that the designated root node and u are connected in the sub-graph.

All problems considered in this thesis will be special cases of how S_q is defined for each request.

Joint replenishment problem. $\mathcal{U}$ includes all items in the original JRP instance and an additional item o with cost equal to the fixed ordering fee. Each request q is on an item x and S_q includes any subset of $\mathcal{U}$ that contains x and o.

Multi-level aggregation problem. $\mathcal{U}$ includes all nodes of the input tree. Each request q is on a node u and S_q is the set of sub-trees in the original input tree such that the root node and u are connected in the sub-tree.

Set cover problem. $\mathcal{U}$ includes all subsets in the set cover instance. Each request q is on an element e and is served by buying a set containing e. Hence S_q includes any sub-collection $X \subseteq \mathcal{U}$ that contains some set S such that $e \in S$. For example consider the elements $\mathcal{E} = \{a, b, c\}$ and the collection of subsets $\mathcal{S} = \{\{a, b\}, \{b, c\}\}$. Then for any request q on element a, $S_q = \Big\{\{\{a, b\}\}, \{\{a, b\}, \{b, c\}\}\Big\}$. This problem should not be confused with the classical offline set cover problem.

The general model can be of two variants.

2.1.1 Deadline variant

In this variant, each request q is associated with a deadline time d_q. A feasible algorithm must serve every released request before its deadline.

2.1.2 Delay variant

In this variant, each request q is associated with a continuous, monotone non-decreasing delay function $d_q(t)$, defined for every time $t \geq r_q$, where r_q is the release time of q. If q is served at time t, the algorithm must pay $d_q(t)$ in delay penalty. The algorithm can choose not to serve q, in which case it pays $\lim_{t\to\infty} d_q(t)$ in delay penalty.

2.2 Online setting

In the online setting, the algorithm has no knowledge of requests that have not been released yet. The algorithm also makes transmissions online, which means transmissions cannot be made in retrospect.

This is in contrast to the offline setting, where all requests are known in advance and the algorithm makes decisions based on all the input.

2.2.1 Measuring performance

To measure the performance of an online algorithm, competitive analysis is used whereby the performance of the online algorithm is compared to an optimal offline algorithm. As a

convention, the cost of the online algorithm is denoted *ALG* and the cost of the optimal offline algorithm is denoted *OPT*.

DEFINITION 2.1. *(Competitive ratio). An online algorithm is c-competitive (minimization problems only) if*

$$ALG \leq c \cdot OPT + O(1) \tag{2.1}$$

for all possible inputs.

In our analyses, we usually find an upper bound on the cost of the online algorithm and divide this by a lower bound on the optimal offline cost to calculate the competitive ratio.

2.2.2 Running time

Usually, the running time of the online algorithm is not as important as its competitive ratio. However, it's still desirable that the algorithm has polynomial running time. Allowing super-polynomial running time could yield more competitive online algorithms.

2.2.3 Deterministic vs Randomized

Most algorithms used in practice are *deterministic*, that is, given some fixed input data, the algorithm always outputs the same result. Definition 2.1 only works for this class of algorithms.

An algorithm can also be *randomized* whereby the algorithm has access to an additional stream of random data that it uses when making decisions. To analyse such an algorithm, we take the expectation of the cost over the distribution of random data used. The competitive ratio is thus defined as:

DEFINITION 2.2. *(Competitive ratio). A* randomized *online algorithm is c-competitive (minimization problems only) if*

$$\mathbb{E}[ALG] \leq c \cdot OPT + O(1) \tag{2.2}$$

for all possible inputs.

Randomized algorithms are more powerful than deterministic algorithms against an *oblivious adversary*.

DEFINITION 2.3. *(Ben-David et al., 1994) An* oblivious adversary *is one who constructs the input sequence in advance knowing the online algorithm, but pays for it optimally.*

In the deterministic case, a malicious adversary can generate an input sequence that maximizes the ratio between the online algorithm's cost and the optimal cost as it knows which steps the algorithm will take. On the other hand, introducing random data allows a randomized algorithm to hide which steps it will take from an oblivious adversary. This often results in improved competitive ratios. Analysis of online randomized algorithms typically assumes the oblivious adversarial model.

2.2.4 Clairvoyance

An online algorithm is *clairvoyant* if, upon a request's arrival time, its deadline or delay function is immediately revealed to the algorithm. Otherwise the algorithm is *non-clairvoyant*.

Clearly a clairvoyant algorithm is at least as powerful as a non-clairvoyant algorithm as it's given more information.

Clairvoyance does not necessarily make a problem strictly easier. For the non-clairvoyant MLAP problem on paths, there is a 4-competitive algorithm and a lower bound of 4 applies even for the clairvoyant case (Bienkowski et al., 2016).

For the deadline variant, non-clairvoyance implies that the algorithm is only informed of a request's deadline at the deadline time. When this occurs the algorithm must immediately serve the expiring request.

For the delay variant, non-clairvoyance means that for any request q at time $t \geq r_q$, the algorithm only knows $d_q(t')$ for $r_q \leq t' \leq t$. However, it's impractical to reveal the delay function continuously. Hence we assume the existence of an oracle that, given a set of requests Q and a value d, notifies the algorithm when the total delay of Q reaches d. If the algorithm makes a polynomial number of calls to the oracle and the rest of the algorithm has polynomial running time, we say that the algorithm has polynomial running time overall.

Most algorithms presented in this thesis are non-clairvoyant. We will show that for all problems considered, there's a clear competitive gap between the non-clairvoyant and the clairvoyant setting.

2.3 TCP acknowledgement

The TCP acknowledgement problem is motivated by the requirement in the TCP protocol whereby a host must acknowledge arriving data. In this problem, requests arrive at a host. At any point in time, the host can send out a single acknowledgement to serve all currently pending requests. The acknowledgement cost is fixed regardless of the number of pending requests.

It is easy to see that this is a special case of JRP, where all items have cost 0 and the fixed ordering fee is the acknowledgement cost.

TCP acknowledgement with deadlines is equivalent to the *interval hitting problem* where given a set of intervals on the real line, we want to compute a set of points in time H such that every interval intersects at least one point. Considering each request's [arrival, deadline] times as an interval, all requests can be served by acknowledging at every point in H. Interval hitting can be solved optimally by an algorithm that sorts intervals from left to right and continually adds the right end point of every interval to H if that interval has not been covered by another point (Chrobak et al., 2015). Thus an optimal algorithm for TCP acknowledgement works as follows. When any request reaches its deadline, the algorithm sends an acknowledgement serving all pending requests. Observe that this algorithm is non-clairvoyant as it does not need to know deadlines of requests ahead of time.

For online TCP acknowledgement with delay, there exists a 2-competitive algorithm (Dooly et al., 2001). The algorithm acknowledges when the delay of all pending requests reaches the acknowledgement cost. We can see that this algorithm is also non-clairvoyant as it does not need to know the future delay of pending requests.

Dooly et al. (2001) also show that their algorithm is optimal by giving a lower bound of 2 on the competitive ratio of any deterministic online algorithm. This lower bound applies even if clairvoyance was permitted.

A better competitive ratio than 2 is possible by allowing randomization. Karlin et al. (2001) present a $\frac{e}{e-1}$-competitive randomized algorithm for TCP acknowledgement with delay. This algorithm is non-clairvoyant and optimal (Seiden, 2000).

2.4 Greedy approach

As seen in the previous section, waiting until the deadline of a request and then making a transmission appears to be a sensible approach to online problems with deadlines. For the delay variant, this translates to waiting until the delay reaches a prescribed amount. The TCP acknowledgement problem is easy in the sense that serving any number of requests costs the same. This means there's no benefit to not serving all requests together. We now consider the non-clairvoyant joint replenishment problem where aggregating more requests into a transmission increases the transmission cost.

2.4.1 Joint replenishment problem

To decide which requests to aggregate together, an algorithm could set a budget for each transmission. A 2-competitive algorithm for clairvoyant JRP aggregates requests in order of increasing deadlines as long as the cost of aggregated items does not exceed the fixed ordering fee (Bienkowski et al., 2014). In other words, the budget is equal to the ordering fee.

Large budget

We can see that it's beneficial to serve many requests together as every transmission incurs an ordering fee.

Consider an example where serving as many requests as possible is bad. In this instance, every item has cost 1 and the ordering fee is 1. At the beginning, a request arrives on every item. Only one of these requests reaches its deadline.

An algorithm with no budget restrictions would serve all requests at the first deadline at cost $n+1$. However an optimal solution would only serve the expiring request at cost 2. Thus the previous algorithm is $\Omega(n)$-competitive.

Observe that when the first request reaches its deadline, a non-clairvoyant algorithm is not aware that all other requests would never reach their deadlines.

Clearly, serving as many requests as possible is not a suitable approach for non-clairvoyant JRP.

Small budget

The problem with the last algorithm is that it can potentially serve requests that never reach their deadlines. These requests would never be served by an optimal solution.

Another approach is to only serve the expiring request. Essentially this means the budget of each transmission is 0 as the algorithm does not transmit unnecessary items.

Consider an instance where each item has cost 1 and the ordering fee is n. At the beginning, a request arrives on every item. Every request has a distinct finite deadline.

A greedy algorithm with zero budget would only serve the expiring request upon its deadline. Hence it would make n transmissions in total, each at cost $n + 1$. The total cost of the algorithm is $2n^2 + 1$.

An optimal algorithm would serve all requests at the beginning, incurring a cost of $n + n = 2n$. Thus the previous algorithm is $\Omega(n)$-competitive.

As in the previous section, non-clairvoyance implies that the online algorithm is not aware that all pending requests have finite deadlines.

Constant budget

Here we consider a similar approach to the 2-competitive algorithm for clairvoyant JRP (Bienkowski et al., 2014), where the budget is equal to the ordering fee.

Consider an instance where every item has cost 1 except one item, denoted x', which has cost n. The ordering fee is set to be n. Again a request is released on every item at the start. Every request has a distinct finite deadline, except the request on x', which has infinite deadline. Immediate after the deadline of every request, a new request is released on x', which has infinite deadline.

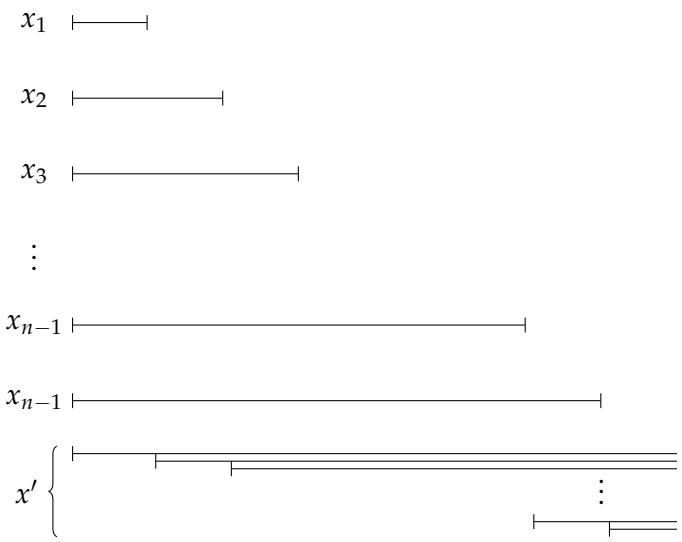


FIGURE 2.1. Instance where the constant budget approach is $\Omega(n)$-competitive

At the deadline of the request on item x_i for $1 \leq i \leq n-1$, an algorithm could arbitrarily pick to aggregate the request on x' with the current transmission. Observe that there is always a request on x' pending at every deadline time. As x' costs n, serving the request on x' immediately exhausts the budget. Hence each transmission has cost $n + c(x_i) + c(x') = 2n + 1$ and there are $n - 1$ transmissions in total, one for every item x_i. The cost of the algorithm is thus at least n^2.

An optimal algorithm simply serves all requests on $\{x_i : 1 \leq i \leq n-1\}$ at the beginning at cost $n + n - 1 = 2n - 1$. Requests on x' never reach their deadlines so do not need to be served. This gives a linear competitive ratio for the online algorithm.

Insights

Both a large budget and a small budget lead to a linear competitive ratio, which is not ideal. The last approach also informs us that arbitrarily picking other requests to serve is not suitable. Unfortunately, (Bienkowski et al., 2014)'s algorithm for the clairvoyant variant serves requests in increasing order of deadlines, which is not possible in our problem due to non-clairvoyance.

In section 3.1.3 we explore a different approach that serves requests depending on the cost of the items they're on and show that a sub-linear competitive ratio is possible for non-clairvoyant JRP.

CHAPTER 3

Multi-level aggregation problem

In this chapter, we show a lower bound on the competitive ratio of any non-clairvoyant algorithm for JRP. Additionally, we present competitive non-clairvoyant algorithms for both JRP and MLAP.

3.1 Joint replenishment problem

3.1.1 Problem definition

In this problem, we are given a set of n items $\mathcal{X}$, a fixed ordering fee K and a cost function $c : \mathcal{X} \to \mathbb{R}_{\geq 0}$. Each request arrives on an item $x \in \mathcal{X}$ and is served by transmitting x. To transmit a set of items $S \subseteq \mathcal{X}$, the algorithm pays $K + \sum_{x \in S} c(x)$. In the deadline variant, each request q has a deadline time d_q it must be served by. We allow $d_q = \infty$ such that q never has to be served. In the delay variant, requests accumulate delay if they are not served.

3.1.2 Lower bound

We describe a fractional relaxation of the integral problem, where requests can be partially served. Let $\gamma(q)$ be the total amount by which q is served, initially 0. q is considered fully served when $\gamma(q) \geq 1$. At any point in time the algorithm can transmit p of a set of items $S \subseteq \mathcal{X}$, where $p \in [0, 1]$. The cost of the transmission is $p \cdot [K + \sum_{x \in S} c(x)]$. Naturally transmitting p of an item increases $\gamma(q)$ by p for every pending request q on x. Observe that an integral algorithm has $p \in \{0, 1\}$.

A randomized algorithm can transmit a set of items $S \subseteq \mathcal{X}$ with probability $p \in [0, 1]$. Hence its expected behaviour is equivalent to transmitting p of S in the fractional algorithm. We

can therefore convert any randomized algorithm into a deterministic fractional algorithm with identical cost. This implies that a lower bound on the competitive ratio of a deterministic fractional algorithm also applies to a randomized integral algorithm.

We next show a $\Omega(\sqrt{n})$ lower bound on the competitiveness of a deterministic fractional algorithm for JRP with deadlines against an integral optimum. Showing this is enough to prove the following theorem.

THEOREM 1.1. *Any randomized non-clairvoyant algorithm for JRP with deadlines is $\Omega(\sqrt{n})$-competitive.*

Proof. Since we're working in the non-clairvoyant setting, an online algorithm does not know pending requests' deadlines and thus an adversary can set deadlines online. At any time t, we say that the adversary *deadlines* a request q when it sets d_q to be t. Observe that the online algorithm is forced to serve q when the adversary deadlines q.

We construct an instance of JRP with deadlines as follows. Let $c(x) = 1$ for all $x \in \mathcal{X}$ and $K = \sqrt{n}$. At the beginning, the adversary releases a request on every item, such that there are n requests in total. While there remains a request q such that $\gamma(q) \leq \frac{1}{2}$, the adversary deadlines q and waits for the algorithm to serve q before deadlining other requests. We will prove that a fractional online algorithm always pays at least $\frac{\sqrt{n}}{2}OPT$, where OPT is the optimal cost.

Let k be the number of requests that the adversary deadlines. An optimal offline algorithm knows all such requests in advance and thus serves them together at a cost of $\sqrt{n} + k$ at the beginning.

Every time the adversary deadlines some request q, $\gamma(q) \leq \frac{1}{2}$. This means that the algorithm must pay an ordering fee of at least $\frac{\sqrt{n}}{2}$ to serve q. In total the algorithm pays $\frac{k\sqrt{n}}{2}$ towards ordering fees. Since the adversary keeps deadlining requests until all of them are at least half served, the online algorithm fractionally transmits every item to an extent of at least $\frac{1}{2}$, incurring $\frac{n}{2}$ in item costs. Overall the online algorithm pays at least $\frac{1}{2}(k\sqrt{n} + n)$.

Dividing the cost of the online algorithm by the optimal cost yields $\frac{\sqrt{n}}{2}$. Hence any deterministic fractional algorithm for JRP is $\Omega(\sqrt{n})$-competitive. The same lower bound exists for any randomized integral algorithm, giving the theorem. □

3.1.3 Deadline variant

We introduce a simple $(\sqrt{n}+1)$-competitive algorithm for JRP with deadlines.

Let $A(t)$ be the number of items with requests on them at time t. The above algorithm can be optimized to be $(\sqrt{s}+1)$-competitive, where $s = \max_t A(t)$. The maximum is well defined as $A(t) \in \{k \in \mathbb{N} : k \leq n\}$, which is a finite set.

Algorithm description

We classify the set of items $\mathcal{X}$ into cheap and expensive items.

DEFINITION 3.1. *(Cheap/ expensive). For an item $x \in \mathcal{X}$, we say that*

- *x is* cheap *if $c(x) \leq \frac{K}{\sqrt{n}}$. Any request on x is a* cheap *request.*
- *x is* expensive *if $c(x) > \frac{K}{\sqrt{n}}$. Any request on x is an* expensive *request.*

Upon the deadline of a request q, let σ_q be the item q is on. q is called the *triggering* request of the current transmission. If x is cheap, the algorithm transmits all cheap items together. Otherwise, the algorithm only transmits $\{\sigma_q\}$.

Algorithm 1 Non-clairvoyant algorithm for JRP with deadlines

```
1: procedure UPONDEADLINE(q)
2:     if c(σ_q) ≤ K/√n then
3:         Transmit {x ∈ X : c(x) ≤ K/√n}
4:     else
5:         Transmit {σ_q}
```

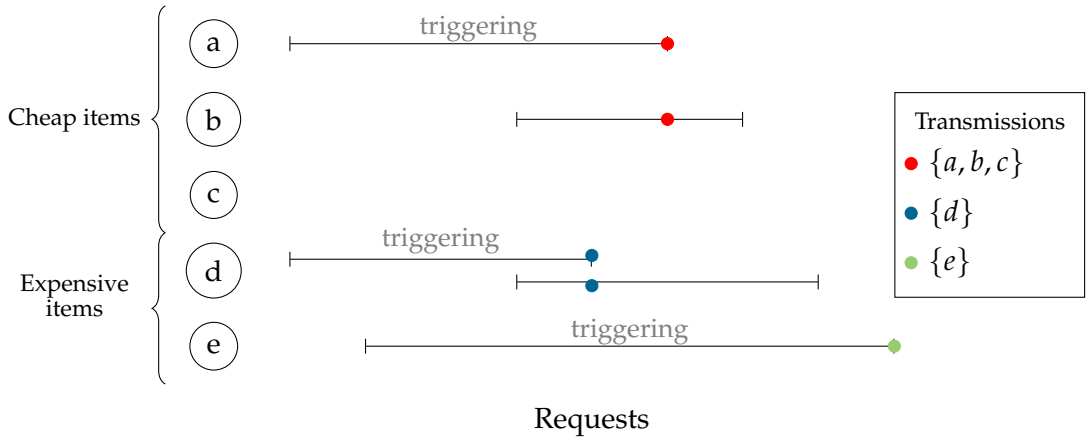


FIGURE 3.1. Example showing transmissions by the algorithm

Analysis

We prove that Algorithm 1 gives the following theorem:

THEOREM 3.2. *There exists a* $(\sqrt{n}+1)$*-competitive deterministic non-clairvoyant algorithm for JRP with deadlines.*

Algorithm 1 is clearly feasible as for every expiring request q, σ_q is transmitted. It remains to bound the cost of the algorithm ALG. First we prove a lower bound on the optimal cost through Lemmas 3.4 and 3.6. Finally we show that ALG is bounded within a $(\sqrt{n}+1)$ ratio of the optimal through Lemmas 3.9 and 3.10.

Split the optimal cost $OPT = OPT_0 + OPT_1$, where OPT_0 is how much the optimal solution pays towards ordering fees and OPT_1 is how much the optimal solution pays towards item costs. Denote Q be the set of triggering requests.

DEFINITION 3.3. *(Disjoint requests). A set of requests is* disjoint *if for every pair of requests, their [arrival, deadline] intervals are pairwise disjoint.*

LEMMA 3.4. *Let* Q' *be a set of disjoint triggering requests, then* $K \cdot |Q'| \leq OPT_0$.

Proof. As every triggering request reaches its deadline, it must be served by an optimal solution. Since Q' is disjoint, all requests in Q' must be served by separate transmissions in

the optimal solution, incurring an ordering fee of K each time. The inequality in the lemma follows. □

PROPOSITION 3.5. *Any two triggering requests on the same item are disjoint.*

Proof. Assume for contradiction that q_1 and q_2 are two triggering requests on x such that q_2 was pending at q_1's deadline. Since they're on the same item, the algorithm would have served q_2 when serving q_1, preventing q_2 from triggering a transmission, which is a contradiction. Similarly, q_1 could not have been pending at q_2's deadline. Hence they are disjoint. □

LEMMA 3.6. $\sum_{q \in Q} c(\sigma_q) \leq OPT_1$.

Proof. Let Λ_x^* be the set of transmissions in the optimal solution containing x. We can rewrite OPT_1:

$$\begin{aligned} OPT_1 &= \sum_{x \in \mathcal{X}} |\Lambda_x^*| \cdot c(x) \\ &\geq \sum_{x \in \mathcal{X}} |\{q \in Q : q \text{ is on } x\}| \cdot c(x) \\ &= \sum_{q \in Q} c(\sigma_q) \end{aligned}$$

where the second last inequality is because triggering requests on the same item are disjoint (Proposition 3.5) and thus must be served by separate transmissions in the optimal solution. □

We now proceed to bound *ALG*.

Let Λ denote the set of transmissions made by the online algorithm. Use Λ_0 to denote the set of transmissions triggered by cheap requests and Λ_1 the set of transmissions triggered by expensive requests, such that $\Lambda = \Lambda_0 \cup \Lambda_1$. Define $c(\Lambda')$ to be the cost of all transmissions in Λ' for any subset $\Lambda' \subseteq \Lambda$.

We first show that $c(\Lambda_0) \leq (\sqrt{n} + 1) \cdot OPT_0$ (Lemma 3.9) by proving that the set of triggering requests for Λ_0 is disjoint (Proposition 3.7)

PROPOSITION 3.7. *All cheap triggering requests are disjoint.*

Proof. Let q be a cheap triggering request. When q expires, the algorithm transmits all cheap items, serving all pending requests on these items. Hence the next cheap triggering request must have been released after q's deadline. □

PROPOSITION 3.8. $c(\Lambda_0) \leq (\sqrt{n}+1) \cdot K \cdot |\Lambda_0|$.

Proof. Every transmission in Λ_0 contains all cheap items. Using that there are at most n items, we get that the total cost of all cheap items is at most $n \cdot \frac{K}{\sqrt{n}} = K\sqrt{n}$. Thus every transmission in Λ_0 has cost at most $K + K\sqrt{n} = (\sqrt{n}+1) \cdot K$. Summing over every transmission in Λ_0 yields $c(\Lambda_0) \leq |\Lambda_0| \cdot (\sqrt{n}+1) \cdot K$. □

LEMMA 3.9. $c(\Lambda_0) \leq (\sqrt{n}+1) \cdot OPT_0$.

Proof. Let Q_0 bet the set of cheap triggering requests. By Proposition 3.7, all cheap triggering requests are disjoint, hence Q_0 is a set of disjoint triggering requests. By Lemma 3.4 $K \cdot |Q_0| \leq OPT_0$. Since every transmission in Λ_0 is due to a distinct cheap triggering request, $|Q_0| = |\Lambda_0|$ and thus $K \cdot |\Lambda_0| \leq OPT_0$. Combine this inequality with Proposition 3.8 we obtain $c(\Lambda_0) \leq (\sqrt{n}+1) \cdot K \cdot |\Lambda_0| \leq (\sqrt{n}+1) \cdot OPT_0$. □

We proceed to bound the cost of Λ_1.

LEMMA 3.10. $c(\Lambda_1) \leq (\sqrt{n}+1) \cdot OPT_1$.

Proof. Let Q_1 be the set of expensive triggering requests. Recall that when an expensive request expires, the algorithm only transmits the single item the request is on. Thus each $q \in Q_1$ triggers a transmission of cost $K + c(\sigma_q)$. We can rewrite $c(\Lambda_1)$ as:

$$
\begin{aligned}
c(\Lambda_1) &= \sum_{q \in Q_1} [K + c(\sigma_q)] \\
&\leq \sum_{q \in Q_1} [\sqrt{n} \cdot c(\sigma_q) + c(\sigma_q)] && \frac{K}{\sqrt{n}} < c(\sigma_q) \text{ as } \sigma_q \text{ is expensive} \\
&= \sum_{q \in Q_1} (\sqrt{n}+1) \cdot c(\sigma_q) \\
&\leq \sum_{q \in Q} (\sqrt{n}+1) \cdot c(\sigma_q) && Q_1 \subseteq Q \\
&\leq (\sqrt{n}+1) \cdot OPT_1 && \text{(Lemma 3.6)}
\end{aligned}
$$

□

Proof of Theorem 3.2. Clearly ALG is equal to the cost of all transmissions. Combining Lemma 3.9 and Lemma 3.10 we get that $c(\Lambda) = c(\Lambda_0) + c(\Lambda_1) \leq (\sqrt{n}+1) \cdot OPT_0 + (\sqrt{n}+1) \cdot OPT_1 = (\sqrt{n}+1) \cdot OPT$. □

Improved $\sqrt{s}+1$ competitive ratio

Recall that $s = \max_t A(t)$, where $A(t)$ is the number of items with requests on them at time t.

Observe that Algorithm 1 might transmit items without requests on them. In instances where $s \ll n$, this will result in unnecessary spending. We claim that by only transmitting items with pending requests on them and replacing n by s in Algorithm 1, we can obtain a $\sqrt{s}+1$ competitive ratio. Furthermore, the algorithm does not need to know s as a priori, it can update s iteratively.

Define $M(t) = \max_{t' \leq t} A(t')$. Clearly $M(t) \leq s$ for all t.

DEFINITION 3.11. *(Cheap/ expensive). At time t, we say that*

- *$x \in \mathcal{X}$ is* cheap *if $c(x) \leq \frac{K}{\sqrt{M(t)}}$. Any request on x is a* cheap *request.*
- *$x \in \mathcal{X}$ is* expensive *if $c(x) > \frac{K}{\sqrt{M(t)}}$. Any request on x is an* expensive *request.*

Unlike in the original algorithm, cheap items and requests can become expensive, but not vice versa.

Upon the deadline of a request q, the algorithm starts a transmission. If σ_q is cheap, the algorithm transmits all cheap items with requests on them. Otherwise, the algorithm only transmits $\{\sigma_q\}$.

Algorithm 2 Optimized non-clairvoyant algorithm for JRP with deadlines

```
1: procedure UPONDEADLINE(q)
2:     Let t be the current time
3:     if c(σ_q) ≤ K/√M(t) then
4:         S ← {x ∈ 𝒳 : there's a pending request on x and c(x) ≤ K/√M(t)}
5:         Transmit S
6:     else
7:         Transmit {σ_q}
```

We prove that Algorithm 2 gives the following result.

THEOREM 1.2. *There exists a $(\sqrt{s}+1)$-competitive deterministic non-clairvoyant algorithm for JRP with deadlines.*

As before, denote Q the set of triggering requests. Let Q_0 be the set of triggering requests that were cheap at their deadlines, that is, $\{q \in Q : c(\sigma_q) \leq \frac{K}{\sqrt{M(d_q)}}\}$. Similarly Q_1 is the set of triggering requests that were expensive at their deadlines, that is, $\{q \in Q : c(\sigma_q) > \frac{K}{\sqrt{M(d_q)}}\}$. Naturally, Λ_0 is the set of transmissions triggered by Q_0 and Λ_1 is the set of transmissions triggered by Q_1.

As in the previous section, split OPT into ordering fees OPT_0 and item costs OPT_1. The lower bounds on OPT_0 and OPT_1 in Lemmas 3.4 and 3.6 still hold.

The rest of the analysis is similar to Algorithm 1's analysis where we bound $c(\Lambda_0)$ in Lemma 3.14 and $c(\Lambda_1)$ in Lemma 3.16.

The following proposition is the counterpart to Proposition 3.7.

PROPOSITION 3.12. *Q_0 is disjoint.*

Proof. Assume for contradiction that there exists two requests $q_1, q_2 \in Q_0$ such that q_2 was pending at q_1's deadline, that is, $d_{q_1} \leq d_{q_2}$ and $d_{q_1} \geq r_{q_2}$. Since $M(t)$ is non-decreasing over

time by definition, $M(d_{q_1}) \leq M(d_{q_2})$ and thus $c(\sigma_{q_2}) \leq \frac{K}{\sqrt{M(d_{q_2})}} \leq \frac{K}{\sqrt{M(d_{q_1})}}$, that is, q_2 was cheap at d_{q_1}. Observe that when q_1 reached its deadline, the algorithm transmitted all cheap items with requests on them. Hence q_2 should have been served with q_1, contradicting the assumption that q_2 later triggered another transmission. □

The following proposition is the counterpart to Proposition 3.8.

PROPOSITION 3.13. $c(\Lambda_0) \leq (\sqrt{s}+1) \cdot K \cdot |\Lambda_0|$.

Proof. Consider any transmission $\lambda \in \Lambda_0$ and let S_λ be the set of items transmitted. As we only transmit items with requests on them, $|S_\lambda| \leq M(t_\lambda)$. Since S_λ only contains cheap items, $c(S_\lambda) \leq |S_\lambda| \frac{K}{\sqrt{M(t_\lambda)}} \leq K\sqrt{M(t_\lambda)} \leq K\sqrt{s}$. Thus every transmission in Λ_0 has cost at most $K + K\sqrt{s} = (\sqrt{s}+1) \cdot K$. Summing over every transmission in Λ_0 yields $c(\Lambda_0) \leq |\Lambda_0| \cdot (\sqrt{s}+1) \cdot K$. □

LEMMA 3.14. $c(\Lambda_0) \leq (\sqrt{s}+1) \cdot OPT_0$.

Proof. $|Q_0| = |\Lambda_0|$ as every transmission in Λ_0 was triggered by a distinct request. By Proposition 3.12, Q_0 is disjoint and thus we can apply Lemma 3.4 to get $K \cdot |\Lambda_0| \leq K \cdot |Q_0| \leq OPT_0$.

Combining this inequality with Proposition 3.13 we obtain $c(\Lambda_0) \leq (\sqrt{s}+1) \cdot K \cdot |\Lambda_0| \leq (\sqrt{s}+1) \cdot OPT_0$. □

We now proceed to bound $c(\Lambda_1)$.

PROPOSITION 3.15. $c(\sigma_q) > \frac{K}{\sqrt{s}}$ *for all* $q \in Q_1$.

Proof. Since $M(t)$ is non-decreasing, $\frac{K}{\sqrt{M(t)}}$ is non-increasing and thus any expensive item remains expensive until the end of the algorithm. Hence $c(\sigma_q) > \frac{K}{\sqrt{s}}$ for all $q \in Q_1$. □

Finally, we prove the counterpart to Lemma 3.10.

LEMMA 3.16. $c(\Lambda_1) \leq (\sqrt{s}+1)\, OPT_1$.

Proof. The proof is almost identical to the proof of Lemma 3.10, except n is replaced by s.

Recall that when an expensive request expires, the algorithm only transmits the single item the request is on. Thus each $q \in Q_1$ triggers a transmission of cost $K + c(\sigma_q)$. We can rewrite $c(\Lambda_1)$ as:

$$\begin{aligned}
c(\Lambda_1) &= \sum_{q \in Q_1} [K + c(\sigma_q)] \\
&\leq \sum_{q \in Q_1} [\sqrt{s}\, c(\sigma_q) + c(\sigma_q)] && \text{(Proposition 3.15)} \\
&= \sum_{q \in Q_1} (\sqrt{n} + 1)\, c(\sigma_q) \\
&\leq \sum_{q \in Q} (\sqrt{n} + 1)\, c(\sigma_q) && Q_1 \subseteq Q \\
&\leq (\sqrt{s} + 1)\, OPT_1 && \text{(Lemma 3.6)}
\end{aligned}$$

□

Proof of Theorem 1.2. Combining Lemma 3.14 and Lemma 3.16 yields $c(\Lambda) = c(\Lambda_0) + c(\Lambda_1) \leq (\sqrt{s}+1) \cdot OPT_0 + (\sqrt{s}+1) \cdot OPT_1 = (\sqrt{s}+1) \cdot OPT$. □

3.1.4 Delay variant

In this variant, each request q is associated with a continuous non-decreasing delay function $d_q(t)$ of time. Assume without loss of generality that $d_q(r_q) = 0$.

We provide an algorithm that gives the following theorem:

THEOREM 1.3. *There exists a $O(\sqrt{s})$-competitive deterministic non-clairvoyant algorithm for JRP with delay.*

We reuse the same idea as in the deadline case of splitting items/requests into cheap and expensive ones. As requests don't have deadlines, we decide when to serve them based on their accumulated delay so far.

Algorithm description

$M(t)$ is defined identically as in the deadline case, where $M(t) = \max_{t' \leq t} A(t')$ and $A(t)$ is the number of items with requests on them at time t.

Recall the definition of cheap and expensive requests in the previous section.

DEFINITION 3.11. *(Cheap/ expensive). At time t, we say that*

- $x \in \mathcal{X}$ *is* cheap *if* $c(x) \leq \frac{K}{\sqrt{M(t)}}$. *Any request on x is a* cheap *request.*
- $x \in \mathcal{X}$ *is* expensive *if* $c(x) > \frac{K}{\sqrt{M(t)}}$. *Any request on x is an* expensive *request.*

At any time t, the algorithm does the following:

(1) Let $Q_C(t)$ be the set of cheap pending requests.
(2) If total delay of $Q_C(t)$ is at least K, transmit all cheap items with requests on them to serve $Q_C(t)$.
(3) For any expensive item x, if the total delay of all requests on x is at least $c(x)$, transmit x.

Analysis

We define some notations before delving into the proof.

Denote $\lim_{t\to\infty} d_q(t) = d_q(\infty)$ for ease. Note that if $d_q(\infty) = \infty$, the algorithm must serve q eventually. Otherwise the algorithm can choose to serve q or not.

Let Λ_0 denote the set of transmissions triggered by the delay of all cheap pending requests being at least K and Λ_1 the set of transmissions triggered by the delay of all pending requests on any expensive item x being at least $c(x)$, such that $\Lambda = \Lambda_0 \cup \Lambda_1$. Denote $R(\mathcal{X}')$ the set of *unserved* requests on any subset of items $\mathcal{X}' \subseteq \mathcal{X}$.

For each transmission $\lambda \in \Lambda$, let t_λ be when λ occurred and Q_λ the set of requests served by λ. Additionally, if $\lambda \in \Lambda_1$, let x_λ denote the item whose requests triggered λ.

The competitive analysis is more involved than the deadline case as we also have to handle delay costs incurred. However the intuitive idea is similar, whereby each transmission in Λ_0

charges K to the optimal solution and each transmission in Λ_1 charges its item cost to the optimal solution.

The algorithm needs to pay transmission costs and well as delay costs of requests. A request is either served by some transmission λ or is not served by the algorithm. For any request q served by λ, the algorithm incurs $d_q(t_\lambda)$ in delay cost. If q is not served, the algorithm must pay $d_q(\infty)$.

Overall, the total cost of the algorithm is the sum of transmission costs, delay costs of requests up until the time they are served and delay costs of unserved requests, that is,

$$ALG = \sum_{\lambda \in \Lambda} \left[c(\lambda) + \sum_{q \in Q_\lambda} d_q(t_\lambda) \right] + \sum_{q \in R(\mathcal{X})} d_q(\infty) \tag{3.1}$$

For each transmission λ, we bound the transmission cost $c(\lambda)$ and delay cost of Q_λ together. This is done for transmissions in Λ_0 in Proposition 3.18 and for transmissions in Λ_1 in Proposition 3.20.

PROPOSITION 3.17. *For every $\lambda \in \Lambda_0$, $\sum_{q \in Q_\lambda} d_q(t_\lambda) = K$.*

Proof. Fix $\lambda \in \Lambda_0$. As λ transmitted all cheap items with requests on them, λ served exactly all requests in $Q_C(t_\lambda)$, that is, the set of all cheap pending requests at time t_λ. Hence $\sum_{q \in Q_C(t_\lambda)} d_q(t_\lambda) = \sum_{q \in Q_\lambda} d_q(t_\lambda)$. Denote $d_C(t) = \sum_{q \in Q_C(t_\lambda)} d_q(t)$ as a shorthand.

Recall that λ was triggered by $d_C(t_\lambda) \geq K$. We need to prove that $d_C(t_\lambda) = K$.

Since $M(t)$ is non-decreasing, requests cannot go from being cheap to expensive. Hence at any time t, if there's a new cheap pending request q in $Q_C(t)$, q must be an arriving request with $d_q(t) = 0$. Using this fact and the continuity of delay functions, λ must have been triggered by $d_C(t)$ being exactly K, that is, $d_C(t_\lambda) = K$. In other words, the delay of cheap pending requests cannot suddenly jump in value. □

PROPOSITION 3.18. *For every $\lambda \in \Lambda_0$, $c(\lambda) + \sum_{q \in Q_\lambda} d_q(t_\lambda) \leq O(\sqrt{s}) \cdot K\cdot$.*

Proof. Fix $\lambda \in \Lambda_0$. λ transmitted all cheap items at the time with requests on them. By definition of cheap, each transmitted item has cost at most $\frac{K}{\sqrt{M(t_\lambda)}}$. As there at most $A(t_\lambda)$ items with requests on them at t_λ and $A(t_\lambda) \leq M(t_\lambda) \leq s$, the cost of all transmitted items is at

most $A(t_\lambda) \cdot \frac{K}{\sqrt{M(t_\lambda)}} \leq K\sqrt{M(t_\lambda)} \leq K\sqrt{s}$. Hence we can bound the transmission cost $c(\lambda) \leq (\sqrt{s}+1) \cdot K$. Combining this inequality with Proposition 3.17 gives $c(\lambda) + \sum_{q \in Q_\lambda} d_q(t_\lambda) \leq (\sqrt{s}+2) \cdot K$. □

PROPOSITION 3.19. *For every $\lambda \in \Lambda_1$, $\sum_{q \in Q_\lambda} d_q(t_\lambda) \leq K + c(x_\lambda)$.*

Proof. Fix $\lambda \in \Lambda_1$ and consider when item x_λ became expensive. Before this point in time, all requests on x_λ were cheap. Therefore the total cumulative delay of these requests at any point in time before x_λ became expensive did not exceed K, otherwise they would have triggered a transmission in Λ_0. On the other hand, λ was triggered by the delay of all requests on x_λ being at least $c(x_\lambda)$. Hence Q_λ's delay was at most $K + c(x_\lambda)$ at time t_λ. □

PROPOSITION 3.20. *For every $\lambda \in \Lambda_1$, $c(\lambda) + \sum_{q \in Q_\lambda} d_q(t_\lambda) \leq O(\sqrt{s}) \cdot c(x_\lambda)$.*

Proof. Fix $\lambda \in \Lambda_1$. The transmission cost is $c(\lambda) = K + c(x_\lambda)$ as the algorithm only transmitted x_λ. With Proposition 3.19, we get that $c(\lambda) + \sum_{q \in Q_\lambda} d_q(t_\lambda) \leq K + c(x_\lambda) + K + c(x_\lambda) = 2K + 2c(x_\lambda)$.

As x_λ was an expensive item when it was served, $c(x_\lambda) > \frac{K}{\sqrt{m_\lambda}} \geq \frac{K}{\sqrt{s}}$ and thus $K < \sqrt{s} \cdot c(x_\lambda)$. Hence we can bound $c(\lambda) + \sum_{q \in Q_\lambda} d_q(t_\lambda) \leq 2K + 2c(x_\lambda) \leq (2\sqrt{s}+2) \cdot c(x_\lambda)$. □

LEMMA 3.21. $ALG \leq O(\sqrt{s}) \left[K \cdot |\Lambda_0| + \sum_{\lambda \in \Lambda_1} c(x_\lambda) + \sum_{q \in R(\mathcal{X})} d_q(\infty)\right]$

Proof. The inequality can be obtained from Proposition 3.18 and Proposition 3.20.

$$
\begin{aligned}
ALG &= \sum_{\lambda \in \Lambda} \left[c(\lambda) + \sum_{q \in Q_\lambda} d_q(t_\lambda)\right] + \sum_{q \in R(\mathcal{X})} d_q(\infty) && \text{(Equation 3.1)} \\
&= \sum_{\lambda \in \Lambda_0} \left[c(\lambda) + \sum_{q \in Q_\lambda} d_q(t_\lambda)\right] + \sum_{\lambda \in \Lambda_1} \left[c(\lambda) + \sum_{q \in Q_\lambda} d_q(t_\lambda)\right] + \sum_{q \in R(\mathcal{X})} d_q(\infty) && \Lambda = \Lambda_0 \cup \Lambda_1 \\
&\leq O(\sqrt{s}) \cdot K \cdot |\Lambda_0| + \sum_{\lambda \in \Lambda_1} \left[c(\lambda) + \sum_{q \in Q_\lambda} d_q(t_\lambda)\right] + \sum_{q \in R(\mathcal{X})} d_q(\infty) && \text{(Prop. 3.18)} \\
&\leq O(\sqrt{s}) \cdot K \cdot |\Lambda_0| + O(\sqrt{s}) \sum_{\lambda \in \Lambda_1} c(x_\lambda) + \sum_{q \in R(\mathcal{X})} d_q(\infty) && \text{(Prop. 3.20)} \\
&\leq O(\sqrt{s}) \left[K \cdot |\Lambda_0| + \sum_{\lambda \in \Lambda_1} c(x_\lambda) + \sum_{q \in R(\mathcal{X})} d_q(\infty)\right]
\end{aligned}
$$

□

We now present a charging scheme to show that $ALG \leq O(\sqrt{n}) \cdot OPT$. Fix an optimal solution and let Λ^* be the set of all transmission in this solution. In general λ^* is always used to refer to a transmission in Λ^*. Let d_q^* denote the delay cost of request q paid by the optimal.

Let $\mathcal{X}_0 = \{x \in \mathcal{X} : c(x) \leq \frac{K}{\sqrt{s}}\}$ and $\mathcal{X}_1 = \{x \in \mathcal{X} : c(x) > \frac{K}{\sqrt{s}}\}$. In other words, $\mathcal{X}_0$ is the final set of cheap items and $\mathcal{X}_1$ the final set of expensive items.

We charge transmissions and delay costs in the optimal as follows:

(1) *For every $\lambda \in \Lambda_0$, charge K to OPT.* Charge the ordering fee K of λ to the ordering fee of some transmission λ^* before or at time t_λ that served any request in Q_λ. If no such transmission exists, the optimal solution must have paid the delay cost of Q_λ until time t_λ. As λ was triggered by the total delay of cheap requests being at least K, $\sum_{q \in Q_\lambda} d_q(t_\lambda) \geq K$. Hence we instead charge K to Q_λ's delay cost in the optimal.

(2) *For every $\lambda \in \Lambda_1$, charge $c(x_\lambda)$ to OPT.* Charge the item cost $c(x_\lambda)$ to item x_λ in some transmission λ^* before or at time t_λ that served any request in Q_λ. If no such transmission exists, the optimal solution must have paid the delay cost of Q_λ until time t_λ. As λ was triggered by the total delay of all requests on x_λ being at least $c(x_\lambda)$, $\sum_{q \in Q_\lambda} d_q(t_\lambda) \geq c(x)$. Hence we instead charge $c(x_\lambda)$ to Q_λ's delay cost in the optimal.

(3) *Charge $\sum_{q \in R(\mathcal{X}_0)} d_q(\infty)$ to OPT.* By definition $\mathcal{R}(\mathcal{X}_0)$ contains all cheap requests when $M(t) = s$. Since the online algorithm did not serve $R(\mathcal{X}_0)$, the total delay of $R(\mathcal{X}_0)$ must not have exceeded K. We thus charge the total delay cost of $R(\mathcal{X}_0)$ to the ordering fee of some transmission λ^* that served any request in $R(\mathcal{X}_0)$. If no such transmission exists, we charge $R(\mathcal{X}_0)$'s delay cost in the algorithm to their delay cost in the optimal.

(4) *Charge $\sum_{q \in R(\mathcal{X}_1)} d_q(\infty)$ to OPT.* For $x \in \mathcal{X}_1$, the total delay of all unserved requests on x must not have exceeded $c(x)$, otherwise the algorithm would have served some request in $\mathcal{R}(\{x\})$. We thus charge the total delay cost of $R(\{x\})$ to item x in some transmission λ^* that served any request in $R(\{x\})$. If no such transmission exists, we charge $R(\{x\})$'s delay cost to their delay cost in the optimal.

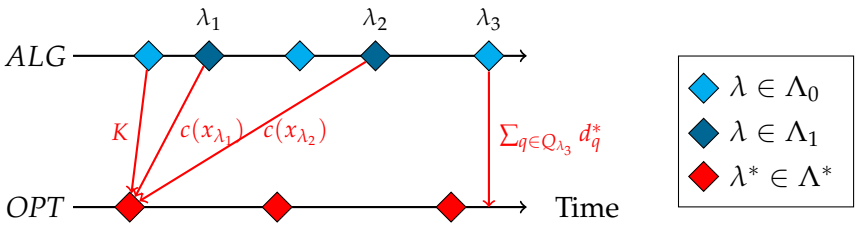


FIGURE 3.2. Charging transmissions in the optimal solution

We will prove that the above charging scheme charges to disjoint portions of the optimal cost, which implies the following lemma::

LEMMA 3.22.

$$OPT \geq K \cdot |\Lambda_0| + \sum_{\lambda \in \Lambda_1} c(x_\lambda) + \sum_{q \in R(\mathcal{X})} d_q(\infty)$$

Before proving Lemma 3.22, we show how it implies the main result of this section.

Proof of Theorem 1.3.

$$\begin{aligned} ALG &\leq O(\sqrt{s}) \left[K \cdot |\Lambda_0| + \sum_{\lambda \in \Lambda_1} c(x_\lambda) + \sum_{q \in R(\mathcal{X})} d_q(\infty)\right] && \text{(Lemma 3.21)} \\ &\leq O(\sqrt{s})\, OPT && \text{(Lemma 3.22)} \end{aligned}$$

Therefore our algorithm is $O(\sqrt{s})$-competitive, as required. □

To prove Lemma 3.22, we show that we never charge a transmission's ordering fee or item cost twice and likewise we never charge a request's delay cost twice.

The charging scheme ensures the following observation.

OBSERVATION 3.23. *If $\lambda \in \Lambda$ charges a transmission $\lambda^* \in \Lambda^*$, then λ^* occurred before λ or equivalently, before time t_λ.*

PROPOSITION 3.24. *We never charge a transmission's ordering fee twice.*

Proof. Assume for contradiction that some $\lambda \in \Lambda_0$ charges its ordering fee to λ^*'s ordering fee and either:

- A different $\lambda' \in \Lambda_0$ charges its ordering fee to λ^*. Without loss of generality we can assume that $t_\lambda \leq t_{\lambda'}$. This means that there is some request $q \in Q_{\lambda'}$ that was served

by λ^* (in the optimal) and thus was pending at t_λ by Observation 3.23. By definition q was a cheap request when it was served at $t_{\lambda'}$ by the online algorithm. As $M(t)$ is non-decreasing, q was also a cheap request at time t_λ. However λ served all cheap requests pending at t_λ, contradicting $q \in Q_{\lambda'}$.

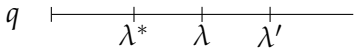


FIGURE 3.3. When λ^*, λ, λ' occur relative to a request's lifetime

- $R(\mathcal{X}_0)$ charges its delay cost to λ^*. This means there is some request $q \in R(\mathcal{X}_0)$ that was served by λ^* and thus was pending at t_λ by Observation 3.23. Since q was cheap at the end of the algorithm, it was cheap at t_λ and should have been served by λ, contradicting $q \in R(\mathcal{X}_0)$.

□

PROPOSITION 3.25. *We never charge a transmission's item cost twice.*

Proof. Recall that a transmission $\lambda \in \Lambda_1$ transmits the single item x_λ. Assume for contradiction that some $\lambda \in \Lambda_1$ charges its item cost to x_λ in λ^* and either:

- A different $\lambda' \in \Lambda_1$ charges its item cost to x_λ in λ^*. Without loss of generality we can assume that $t_\lambda \leq t_{\lambda'}$. This means x_λ and $x_{\lambda'}$ must be the same item. Furthermore, there is some request $q \in Q_{\lambda'}$ that was served by λ^* and thus was pending at t_λ by Observation 3.23. Since q is on $x_{\lambda'} = x_\lambda$, q should have been served by λ, contradicting q being served by λ'.
- $R(\{x_\lambda\})$ charges its delay cost to x_λ in λ^*. This means there is some request $q \in R(\{x_\lambda\})$ that was served by λ^* and thus was pending at t_λ by Observation 3.23. Since q is on x_λ, q should have been served by λ, contradicting $q \in R(\{x_\lambda\})$.

□

PROPOSITION 3.26. *We never charge a request's delay cost twice.*

Proof. For every request q served by some $\lambda \in \Lambda$, only λ can charge q's delay cost in the optimal solution, thus double charging cannot occur. For every request q not served by the

algorithm, we charge its delay cost in the algorithm to its delay cost in the optimal at most once. □

Proof of Lemma 3.22. We introduce some notations for the proof. For $\lambda^* \in \Lambda^*$, let $\omega(\lambda^*)$ be the cost of items included in λ^*. We can see that $c(\lambda^*) = K + \omega(\lambda^*)$.

Denote $\mathcal{Q}_0$ the set of all requests served by transmission in Λ_0, that is, $\bigcup_{\lambda \in \Lambda_0} Q_\lambda$. Similarly, denote $\mathcal{Q}_1$ the set of requests served by transmission in Λ_1.

From steps 1 and 2 of the charging scheme and Proposition 3.24 and 3.26, we get that:

$$K \cdot |\Lambda_0| + \sum_{q \in R(\mathcal{X}_0)} d_q(\infty) \leq K \cdot |\Lambda^*| + \sum_{q \in R(\mathcal{X}_0) \cup \mathcal{Q}_0} d_q^* \tag{3.2}$$

From steps 3 and 4 of the charging scheme and Proposition 3.25 and 3.26, we get that:

$$\sum_{\lambda \in \Lambda_1} c(x_\lambda) + \sum_{q \in R(\mathcal{X}_1)} d_q(\infty) \leq \sum_{\lambda^* \in \Lambda^*} \omega(\lambda^*) + \sum_{q \in R(\mathcal{X}_1) \cup \mathcal{Q}_1} d_q^* \tag{3.3}$$

We can now lower bound OPT:

$$\begin{aligned}
OPT &= \sum_{\lambda^* \in \Lambda^*} [K + \omega(\lambda^*)] + \sum_q d_q^* && \text{(by definition)} \\
&= K \cdot |\Lambda^*| + \sum_{q \in R(\mathcal{X}_0) \cup \mathcal{Q}_0} d_q^* + \sum_{\lambda^* \in \Lambda^*} \omega(\lambda^*) + \sum_{q \in R(\mathcal{X}_1) \cup \mathcal{Q}_1} d_q^* \\
&\geq K \cdot |\Lambda_0| + \sum_{q \in R(\mathcal{X}_0)} d_q(\infty) + \sum_{\lambda \in \Lambda_1} c(x_\lambda) + \sum_{q \in R(\mathcal{X}_1)} d_q(\infty) && \text{(Eq. 3.2 and Eq. 3.3)} \\
&= K \cdot |\Lambda_0| + \sum_{\lambda \in \Lambda_1} c(x_\lambda) + \sum_{q \in R(\mathcal{X})} d_q(\infty) && R(\mathcal{X}) = R(\mathcal{X}_0) \cup R(\mathcal{X}_1)
\end{aligned}$$

□

3.2 Multi-level aggregation problem

3.2.1 Problem definition

In this problem, we are given a tree $\mathcal{T} = (V, E)$ with non-negative node costs. Requests arrive on nodes in V and must be served by a deadline, which is not known to the online algorithm until the deadline is reached. A request q on a node σ_q is served by transmitting a sub-tree of $\mathcal{T}$ such that σ_q and the root node r are connected in this sub-tree. In this section, we will provide an algorithm to prove this theorem.

THEOREM 1.4. *There exists a $O(D + \sqrt{n})$-competitive deterministic non-clairvoyant algorithm for MLAP with deadlines.*

JRP is a special case of this problem whereby the tree has depth 1.

3.2.2 Algorithm description

We introduce some notations that are needed to describe the algorithm.

Let $c : V \to \mathbb{R}^+$ be the cost function over V.

Let $d : V \to \mathbb{N}$ be the *depth* function, where $d(r) = 0$. The depth of u is defined as the number of edges in the simple path from r to u or equivalently, the number of nodes in this path minus one. Use D to denote the depth of $\mathcal{T}$, which is defined as the maximum depth over all nodes in V.

For any node $v \in V$, let A_v be the set of its ancestors. We assume that v is not an ancestor of itself.

The algorithm works as follows. Upon the arrival of a request q, the algorithm introduces a variable h_q, which is initially the root node r, and maintains this variable until q is served. h_q is called the *head of request q*.

Upon the deadline of a request q, the algorithm starts the service λ_q. We say that q is a *triggering* request as it triggers a service.

We divide the path from r to σ_q into two sub-paths, $P_1(q)$ is the path from r to h_q (excluding h_q) and $P_2(q)$ is the path from h_q to σ_q. The algorithm initializes the forest F and adds all nodes in $P_1(q)$ and $P_2(q)$ to it.

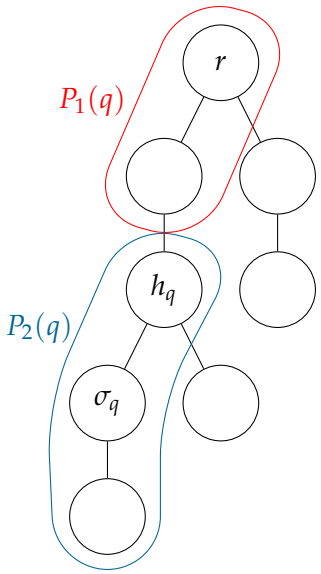


FIGURE 3.4. An example illustrating $P_1(q)$ and $P_2(q)$ for some service λ_q

1. Investment phase: Let S be the set of nodes in the sub-tree rooted at h_q. The algorithm uses nodes in $P_2(q)$ to invest in nodes in S. Each node in $P_2(q)$ can only invest $\frac{1}{\sqrt{n}}$ of its cost to the budget of any of its descendant node. To keep track of this, the algorithm maintains the set of investors $INV(v)$ for every node $v \in V$.

For every node $v \in S$, the algorithm adds every ancestor a of v that is also in $P_2(q)$ to $INV(v)$, paying $\frac{c(a)}{\sqrt{n}}$ for this investment. The algorithm doesn't pay anything if a was already an investor of v.

DEFINITION 3.27. *(Free). A node v is said to be* free *if its budget is at least its cost. Specifically, u is free when $\sum_{a \in INV(u)} \frac{c(a)}{\sqrt{n}} \geq c(u)$.*

If after the service makes investments on v, v becomes free, it is added to F.

2. Transmission phase: The transmission tree T is set to be the sub-tree in F rooted at r. The algorithm transmits T and resets all investments to nodes in T. All nodes in $P_1(q)$ and $P_2(q)$

are in T as they are connected to the root. A free node is in T if and only if all its ancestors are free or part of $P_1(q) \cup P_2(q)$.

3. Downgrade phase:

DEFINITION 3.28. *(Downgrade). During a service λ_q, if the head of some request q' was set to be a new node a, we say λ_q downgraded q' to a or q downgraded q' to a.*

For any node $v \in S \setminus T$, let a be the highest ancestor of v not included in T. The algorithm downgrades the head of every request q' on v to a if $d(h_{q'}) < d(a)$.

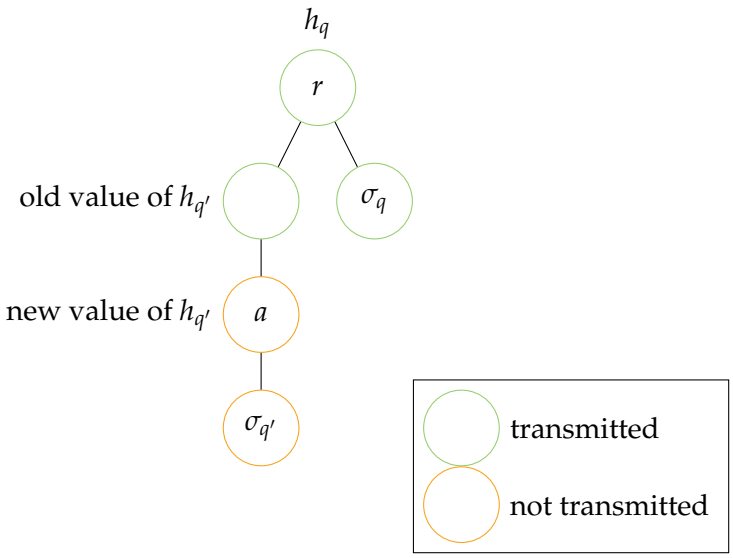


FIGURE 3.5. An example of λ_q downgrading q'

Algorithm 3 Non-clairvoyant algorithm for MLAP with deadlines

procedure UPONARRIVAL(q)
    Set $h_q \leftarrow r$
procedure UPONDEADLINE(q)
    Let $P_1(q)$ be the path from r to h_q (excluding h_q)
    Let $P_2(q)$ be the path of $P(q)$ from h_q to σ_q
    $F \leftarrow P_1(q) \cup P_2(q)$
    Let S be the set of nodes in the sub-tree rooted at h_q
    ▷ **1. Investment phase**
    for each $v \in S$ **do**
        ▷ invest into the budget of v by updating v's investors
        for each $a \in A_v \cap P_2(q)$ **do**
            $INV(v) \leftarrow INV(v) + a$
        ▷ add v to forest if it's free
        if $c(v) \leq \sum_{a \in INV(v)} \frac{c(a)}{\sqrt{n}}$ **then**
            $F \leftarrow F + v$
    ▷ **2. Transmission phase**
    ▷ we don't transmit any nodes disconnected from the root
    Let T be the tree in F rooted at r
    Transmit all nodes in T
    ▷ reset investments to nodes in T
    for each $v \in T$ **do**
        $INV(v) \leftarrow \emptyset$
    ▷ **3. Downgrade phase**
    for each $v \in S$ **do**
        if there's a pending request on v and $v \notin T$ **then**
            Let a be the highest ancestor of v not in T
            for each request q' on v **do**
                if $d(h_{q'}) < d(a)$ **then**
                    Set $h_{q'} \leftarrow a$

3.2.3 Analysis

We first prove that the algorithm serves all requests by their deadlines.

LEMMA 3.29. *Algorithm 3 is feasible for MLAP with deadlines.*

Proof. We show that every request that reaches its deadline is served. The algorithm starts a service whenever a request q reaches its deadline. In the service, all nodes in the path from r to h_q and in the path from h_q to σ_q are added into F. Hence the path from r to σ_q is in the transmission tree, ensuring that the expiring request q is served. The algorithm is therefore feasible. □

Upper bounding *ALG*

Let $c(\lambda_q)$ be the cost of service λ_q. $c(\lambda_q)$ includes the costs of $P_1(q)$, $P_2(q)$ and investment cost, denoted $c_{inv}(\lambda_q)$. Let ALG be the cost of the algorithm, which is the total cost of all transmission trees. Clearly ALG is at most the cost of all services as every service λ_q pays for $P_1(q)$, $P_2(q)$ and any other transmitted node is paid for via investment costs of past or current services.

In this section we bound the cost of each service λ_q to be at most $O(\sqrt{n} + D) \sum_{u \in P_2(q)} c(u)$. The main result is stated in Lemma 3.32. The most difficult part in the analysis is bounding the cost of $P_1(q)$ (Lemma 3.31).

First, we show that λ_q's investment cost can be bounded by the cost of $P_2(q)$.

LEMMA 3.30. *For each service λ_q, its investment cost $c_{inv}(\lambda_q)$ is at most $\sqrt{n} \cdot \sum_{u \in P_2(q)} c(u)$.*

Proof. λ_q invests in at most n nodes. Only nodes in $P_2(q)$ are used to invest and each node can only invest $\frac{1}{\sqrt{n}}$ of its cost. Therefore the total investment cost cannot exceed $\frac{n}{\sqrt{n}} \cdot \sum_{u \in P_2(q)} c(u) = \sqrt{n} \cdot \sum_{u \in P_2(q)} c(u)$. □

Next we consider the cost of $P_1(q)$.

LEMMA 3.31. *For each service λ_q, $\sum_{u \in P_1(q)} c(u) \leq [d(h_q) + \sqrt{n}] \cdot \sum_{u \in P_2(q)} c(u)$.*

We state the main result of this section before proving Lemma 3.31.

LEMMA 3.32. *For each service λ_q, $c(\lambda_q) \leq (2\sqrt{n} + D + 1) \sum_{u \in P_2(q)} c(u)$.*

Proof. From the definition of service cost, we have that:

$$\begin{aligned}
c(\lambda_q) &= c_{inv}(\lambda_q) + \sum_{u \in P_1(q)} c(u) + \sum_{u \in P_2(q)} c(u) \\
&\leq \sqrt{n} \sum_{u \in P_2(q)} c(u) + \sum_{u \in P_1(q)} c(u) + \sum_{u \in P_2(q)} c(u) && \text{(Lemma 3.30)} \\
&\leq \sqrt{n} \sum_{u \in P_2(q)} c(u) + [d(h_q) + \sqrt{n}] \sum_{u \in P_2(q)} c(u) + \sum_{u \in P_2(q)} c(u) && \text{(Lemma 3.31)} \\
&= [2\sqrt{n} + d(h_q) + 1] \sum_{u \in P_2(q)} c(u) \\
&\leq (2\sqrt{n} + D + 1) \sum_{u \in P_2(q)} c(u) && d(h_q) \leq D
\end{aligned}$$

where the last inequality is due to the fact that the depth of any node is at most the depth of the tree. □

COROLLARY 3.33. *$ALG \leq O(\sqrt{n} + D) \sum_{q \text{ is triggering}} \left[\sum_{u \in P_2(q)} c(u) \right]$.*

Proof. As every service serves the request that triggered it, we obtain that all services are triggered by distinct requests. Hence summing the inequality in Lemma 3.32 over every triggering request gives the desired result. □

The rest of this section is dedicated to proving Lemma 3.31, which bounds the cost of $P_1(q)$ within a $O(\sqrt{n} + D)$ factor of the cost of $P_2(q)$. We restate the lemma and introduce some necessary notations.

LEMMA 3.31. *For each service λ_q, $\sum_{u \in P_1(q)} c(u) \leq [d(h_q) + \sqrt{n}] \cdot \sum_{u \in P_2(q)} c(u)$.*

Fix a triggering request q^* and let h^* denote the last value of h_{q^*}. We will show that Lemma 3.31 is true for q^*. If $h_{q^*} = r$, $P_1(q^*) = \varnothing$ which immediately gives the inequality since the left hand side is 0. We assume otherwise, that is, q^* was downgraded at least once.

Let $(q_0, q_1, ..., q_m)$ be the chain of triggering requests such that:

- q_i is the last request that downgraded q_{i+1} for all $0 \leq i < m$.
- q_m is the last request that downgraded q^*.
- q_0 was never downgraded.

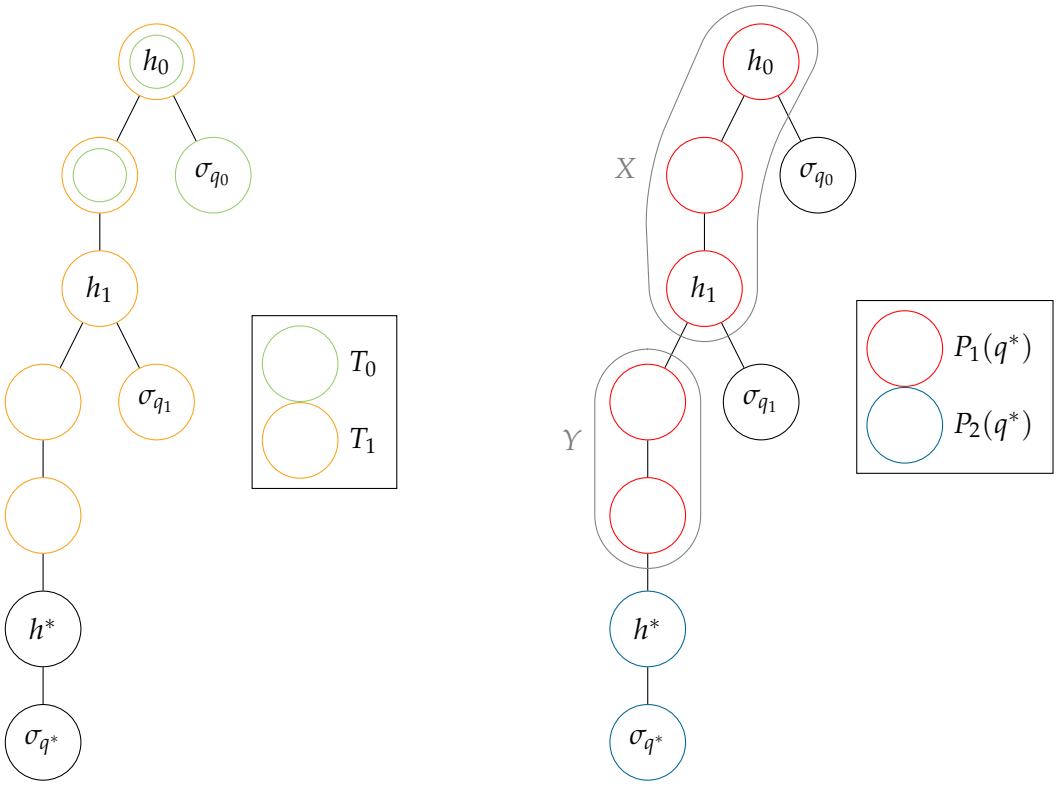
The following notations are introduced for ease.

- λ_i: the service triggered by q_i expiring.
- T_i: the transmission tree transmitted by λ_i.
- h_i: the head of request q_i at its deadline or equivalently, the node that q_{i-1} downgraded q_i to (only if $1 \leq i \leq m$).

Let t_s be the time when q_0 was released and t_f the time when q_m reached its deadline. Define the interval $I = [t_s, t_f)$.

To bound the cost of $P_1(q^*)$, further classify nodes in $P_1(q^*)$ into two sets X and Y:

- X includes every node that was in $INV(h^*)$ at some point during the interval I.
- $Y := P_1(q^*) \setminus X$.



(A) T_0, T_1 and h^* (B) How $P_1(q^*)$ could be divided into X and Y

FIGURE 3.6. An example illustrating the chain (q_0, q_1)

We bound the cost of X by $\sqrt{n} \cdot c(h^*)$ in Lemma 3.34. This is done by showing that at time t_f, h^* was not free and $INV(h^*) = X$. Observe that the definition of X does not guarantee that every node in X remained in $INV(h^*)$ until time t_f.

Then we bound the cost of Y by $\frac{1}{\sqrt{n}} \cdot \sum_{x \in X} c(x)$ in Lemma 3.40.

LEMMA 3.34. $\sum_{x \in X} \frac{c(x)}{\sqrt{n}} < c(h^*)$.

We state and prove the following statements in order to prove Lemma 3.34.

OBSERVATION 3.35. *At any time $t \in I$, there's at least one request in the chain pending, that is, it's served strictly after time t.*

Proof. q_0 was pending from its release time at t_s until its deadline. For all $0 \leq i < m$, since q_i downgraded q_{i+1}, q_{i+1} was pending at q_i's deadline until its deadline for all $1 \leq i < m$. Hence at any point in I, there's always at least 1 request in the chain pending. □

OBSERVATION 3.36. *For every $1 \leq i \leq m$, λ_{i-1} transmitted all ancestors of h_i.*

Proof. In the downgrade phase of λ_{i-1}, the algorithm would have downgraded q_i to the highest ancestor of σ_{q_i} not in the transmission tree T_{i-1}. Since this node is h_i, it follows that all of its ancestors were transmitted. □

PROPOSITION 3.37. *After the downgrade phase of a service λ_q, the head of any unserved request on $u \in S$ is another node in S, where S is the sub-tree rooted at h_q.*

Proof. Let $u \in S$. We know that all ancestors of h_q (excluding itself) are in $P_1(q)$ and therefore are included in the transmission tree. Since u is in the sub-tree rooted at h_q and all of h_q's ancestors (excluding itself) are transmitted, the highest ancestor of u not in the transmission tree must have been a descendant of h_q and thus be in S. □

PROPOSITION 3.38. *For $1 \leq i \leq m$, q_i was always downgraded to a node in $P_1(q^*)$.*

Proof. We prove this is true for q_m and then for all requests in the chain by induction on i starting from m.

Since q_m downgraded q to h^*, h^* must have been a descendant node of the sub-tree rooted at h_m by Proposition 3.37. Thus h_m is an ancestor of h^*. Now consider any previous downgrade of q_m. Since h_m is the last node q_m was downgraded to, any previous downgrade must have

been to an ancestor of h_m, which in turn is an ancestor of h^*. $P_1(q^*)$ being the path from the root to h^* (excluding itself) contains all ancestors of h^*. Thus the statement is true for $i = m$.

Now assume that the statement is true for $i = k$, that is, q_{k-1} downgraded q_k to $h_k \in P_1(q^*)$. By a similar argument as in the base case we get that h_{k-1} is an ancestor of h_k and is part of $P_1(q^*)$. Any previous downgrade of q_{k-1} was therefore also to a node in $P_1(q^*)$. The statement holds for $i = k - 1$.

By induction, the proposition holds for all $1 \leq i < m$. □

PROPOSITION 3.39. *h^* was never transmitted in I. Hence $X = INV(h^*)$ at time t_f.*

Proof. Assume for the sake of contradiction that at time $t \in I$, h^* was transmitted as part of some transmission tree T. By Observation 3.35, there exists a request q in the chain pending at time t. As T included h^*, T must have also included all of its ancestors, that is, all nodes in $P_1(q^*)$. Hence the algorithm would have downgraded q to a node outside $P_1(q^*)$. However, Proposition 3.38 guarantees that every request in the chain can only be downgraded to a node in $P_1(q^*)$. This is a contradiction and thus h^* was never transmitted during I.

By definition, X contains every node that was in $INV(h^*)$ at some point during I. Since h^* was never transmitted during I, every node in X remained in $INV(h^*)$. □

We can finally prove Lemma 3.34.

LEMMA 3.34. $\sum_{x \in X} \frac{c(x)}{\sqrt{n}} < c(h^*)$.

Proof. We know that all of h^*'s ancestors were in the transmission tree T_m by Observation 3.36. If h^* was free, the algorithm would have added it to T_m. Thus h^* was not free at t_f, which ensures that its budget at t_f, that is, $\sum_{a \in INV(h^*)} \frac{c(a)}{\sqrt{n}}$, did not exceed $c(h^*)$. Using that $X = INV(h^*)$ from Proposition 3.39 yields the lemma. □

We proceed to bound the cost of Y.

LEMMA 3.40. *For every $y \in Y$, $c(y) \leq \sum_{x \in X} \frac{c(x)}{\sqrt{n}}$.*

In order to prove this lemma, we prove the following two propositions.

PROPOSITION 3.41. *Let u be an ancestor of v. After the investment phase of every service, $INV(u) \subseteq INV(v)$.*

Proof. At the beginning of the algorithm, this proposition holds as $INV(u) = \emptyset = INV(v)$. The algorithm does not invest in $UponArrival$ so we only need to consider $UponDeadline$. Specifically, the algorithm only makes changes to $INV(u)$ and $INV(v)$ in the investment phase and the transmission phase.

Consider an iteration of $UponDeadline(q)$ and assume that the proposition holds at the start of this iteration. We prove that after the investment phase, $INV(u) \subseteq INV(v)$ always holds. This is clearly true if the algorithm does not invest in u so assume otherwise. Since the algorithm only invests in nodes in the sub-tree rooted at h_q, u is in this sub-tree. As v is a descendant of u, it's also in the same sub-tree and the algorithm does invest in it. If a node a is used to invest into u, $a \in P_2(q) \cup A_u$. Since u is an ancestor of v, we have that $A_u \subseteq A_v$. Hence if the algorithm adds a to $INV(u)$, it also adds a to $INV(v)$, maintaining $INV(u) \subseteq INV(v)$.

It remains to prove that the inequality is maintained after resetting investments in the transmission phase. If the algorithm does not reset investments to v then clearly the inequality holds so we assume otherwise. Investments are only reset for transmitted nodes. If v is transmitted, any ancestor of v is also transmitted as every transmission tree is necessarily rooted at r and connected. Hence u is also transmitted and we get that $INV(u) = \emptyset = INV(v)$ at the end.

By applying the above argument to each successive sub-routine and using that the inequality holds at the beginning of the algorithm, the proposition is obtained. □

PROPOSITION 3.42. *For every $y \in Y$, y became free at some point during I.*

Proof. If $y \in P_2(q_i)$ for any $0 \leq i \leq m$, y would have invested into h^* during the service λ_i as it's an ancestor of h^*. Since y was never an investor of h^* in I, $y \notin P_2(q_i)$ for all $0 \leq i \leq m$. Note that the converse does not necessarily hold.

Let $y \in Y$. Assume for contradiction that y never became free. We find the transmission in the chain for which y was first transmitted and define $\alpha := \arg\min_{y \in T_i} i$. This is well defined as y being an ancestor of h^* means it was transmitted as part of T_m. By assumption y was not free in λ_α and we showed previously that y was not in $P_2(q_\alpha)$ either. Hence y must have been

part of $P_1(q_\alpha)$, and thus an ancestor of the node h_α. But this implies that y was transmitted by $\lambda_{\alpha-1}$ through Observation 3.36, contradicting the assumption of λ_α as the first transmission that contained y. □

Recall Lemma 3.40.

LEMMA 3.40. *For every $y \in Y$, $c(y) \leq \sum_{x \in X} \frac{c(x)}{\sqrt{n}}$.*

Proof. Proposition 3.42 ensures that y became free in I. We consider the first service in I such that y was free after the investment phase.

$$\begin{aligned} c(y) &\leq \sum_{a \in INV(y)} \frac{c(a)}{\sqrt{n}} && (y \text{ was free}) \\ &\leq \sum_{a \in INV(h^*)} \frac{c(a)}{\sqrt{n}} && INV(y) \subseteq INV(h^*) \\ &\leq \sum_{a \in X} \frac{c(a)}{\sqrt{n}} \end{aligned}$$

Using that y is an ancestor of h^* and Proposition 3.41, we get the second inequality. The last inequality follows from the definition of X being the set that contains any investor of h^* at some point during I. □

We can finally prove Lemma 3.31 using Lemma 3.34 and Lemma 3.40.

LEMMA 3.31. *For each service λ_q, $\sum_{u \in P_1(q)} c(u) \leq [d(h_q) + \sqrt{n}] \cdot \sum_{u \in P_2(q)} c(u)$.*

Proof. We prove the desired inequality for the service λ_{q^*}. Recall that the sets X and Y make up $P_1(q^*)$.

$$\begin{aligned} \sum_{u \in P_1(q^*)} c(u) &= \sum_{x \in X} c(x) + \sum_{y \in Y} c(y) \\ &\leq \sum_{x \in X} c(x) + \sum_{y \in Y} \left[\sum_{x \in X} \frac{c(x)}{\sqrt{n}} \right] && \text{(Lemma 3.40)} \\ &\leq \sqrt{n} \cdot c(h^*) + \sum_{y \in Y} c(h^*) && \text{(Lemma 3.34)} \\ &\leq \sqrt{n} \cdot c(h^*) + d(h^*) \cdot c(h^*) && |Y| \leq |P_1(q^*)| = d(h^*) \\ &\leq [\sqrt{n} + d(h^*)] \cdot \sum_{u \in P_2(q^*)} c(u) \end{aligned}$$

The last inequality is clear as $h^* \in P_2(q^*)$. □

Lower bounding *OPT*

In this section, we show that the sum over every triggering request q of the cost of $P_2(q)$ is a lower bound on *OPT*. This can be done via a simple charging procedure.

The algorithm only starts a service when a request q reaches its deadline. Therefore for each triggering request q, the optimal solution must contain a transmission tree T^* that serves this request and thus contains the simple path from r to σ_q. We charge the cost of every node $u \in P_2(q)$ to $u \in T^*$.

LEMMA 3.43. *For each transmission tree of the optimal solution, we charge every node at most once.*

Assume for contradiction that for some optimal transmission tree T^*, we charge a node v twice. Let T_1 and T_2 be the two transmission trees that charged $v \in T^*$. Let q_1 and q_2 be the two requests that triggered the two services respectively. Without loss of generality we assume T_1 was transmitted before T_2 and thus q_1's deadline was before q_2's deadline. As the optimal solution served both requests together, q_2 was pending at q_1's deadline, which was also when T_1 was transmitted.

As $v \in T^*$ was charged twice, $v \in P_2(q_1)$ and $v \in P_2(q_2)$. Since q_2 was pending when T_1 was transmitted but not served then, $\sigma_{q_2} \notin T_1$. The downgrade phase of λ_{q_1} guarantees that $h_{q_2} \notin T_1$, where h_{q_2} is the head of q_2 after λ_{q_1}. But $v \in P_2(q_2)$ implies that v is a descendant of h_{q_2}, and thus not in T_1. This contradicts $v \in P_2(q_1) \subseteq T_1$.

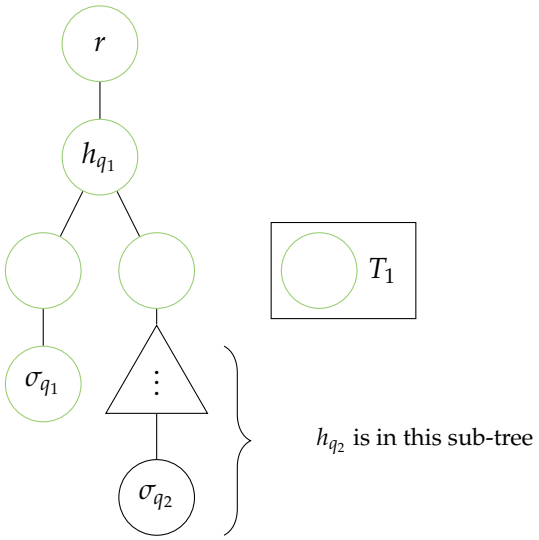


FIGURE 3.7. An example showing that T_1 and $P_2(q_2)$ are disjoint

This corollary follows from Lemma 3.43.

COROLLARY 3.44. $OPT \geq \sum_{q \text{ is triggering}} \left[\sum_{u \in P_2(q)} c(u)\right]$.

Corollary 3.33 and Corollary 3.44 are used to prove the desired competitive ratio.

Proof of Theorem 1.4.

$$\begin{aligned} ALG &\leq O(\sqrt{n} + D) \cdot \sum_{q \text{ is triggering}} \Big[\sum_{u \in P_2(q)} c(u) \Big] && \text{(Corollary 3.33)} \\ &\leq O(\sqrt{n} + D) \cdot OPT && \text{(Corollary 3.44)} \end{aligned}$$

□

CHAPTER 4

Set cover with delay problem

4.1 Problem definition

We are given a universe of elements $\mathcal{E}$ and a weighted family of subsets over $\mathcal{E}$ denoted by $\mathcal{S}$. Let $c : \mathcal{S} \to \mathbb{R}^+$ be the cost function over $\mathcal{S}$. Each request q arrives on an element $e \in \mathcal{E}$ and is served by buying any set $S \in \mathcal{S}$ such that $e \in S$. Every request q is associated with a continuous non-decreasing delay function $d_q(t)$ of time. An online algorithm has to pay $d_q(t)$ if it serves q at time t. If the algorithm does not serve q it has to pay $\lim_{t\to\infty} d_q(t)$.

Let n be the number of elements in $\mathcal{E}$ and m the number of sets in $\mathcal{S}$. We also define the frequency $k := \max_{e\in\mathcal{E}} |\{S \in \mathcal{S} : e \in S\}|$, that is, the maximum number of sets any element is part of. Clearly $k \leq m$.

4.2 Non-clairvoyant setting

In the non-clairvoyant setting, the algorithm does not know $d_q(t)$ for any future time t.

A natural fractional relaxation of this problem is to allow sets to be fractionally bought. For example, if an element e is in both S_1 and S_2, the algorithm can transmit half of S_1 and half of S_2 to serve a request on e.

Azar et al. (2020) prove the following theorem in their paper.

THEOREM 4.1. *(Azar et al., 2020) [Theorem 2]. There exists a $O(\log k)$-competitive non-clairvoyant deterministic algorithm for fractional SCD.*

We refer to the algorithm that gives Theorem 4.1 as *ONF* (short for online fractional). We can run *ONF* on any SCD instance to obtain a momentary non-negative buying function $x_S(t)$ for

each set $S \in \mathcal{S}$. $x_S(t)$ is defined for every time t. The total buying cost of ONF is $\sum_{S \in \mathcal{S}} c(S) \cdot \int_0^\infty x_S(t)\, dt$.

To simplify the bounding of delay costs, we restate the following remark in the paper.

REMARK 4.2. *(Azar et al., 2020) [Remark 6]. For the delay model under which a partially served request incurs the same amount of delay penalty as a fully unserved request, ONF is $O(\log k)$-competitive.*

4.2.1 Intuition

Azar et al. (2020) present a randomized rounding scheme using the fractional algorithm that loses a $\log n$ factor in the competitive ratio. The randomized algorithm consists of two steps:

- Type A buying: For each set S, buy S with some probability dependent on $x_S(t)$.
- Type B buying: For an element e, if ONF has bought $\frac{1}{2}$ of e since the last time the integral algorithm bought e, buy the cheapest set containing e.

The analysis consists of bounding the expected value of Type A buying and showing that Type B buying happens with low probability, thus also bounding its expected cost.

We will derandomize this $O(\log k \log n)$-competitive algorithm into a deterministic algorithm with no degradation in the competitive ratio.

To achieve this, we come up with a potential function Φ that guides the online algorithm in rounding the fractional solution. Our algorithm is inspired by the derandomization technique used for the online set cover problem (Buchbinder and Naor, 2009). Also see (Alon et al., 2003) for the original algorithm for this problem.

We construct a deterministic algorithm following a similar pattern to the randomized one. For type A buying, rather than buying sets with some probability, we decide to buy a set or not depending on which choice minimizes Φ. Type B buying is identical to the randomized case.

Intuitively, Φ will represent the expected cost of the integral algorithm. Thus making choices that minimize Φ is akin to calculating conditional expectations and picking the option that yields the lowest expectation. This approach is called the method of conditional expectations. A more detailed exposition on this method can be found in (Alon and Spencer, 2008).

4.2.2 Preliminary notations

Let $c(e)$ be the cost of the cheapest set containing e.

For ease, introduce $x_e(t) = \sum_{S:e\in S} x_S(t)$, which can be viewed as a momentary buying function for element e.

We define the threshold time t_l^e for $l \in \mathbb{N}$ (excluding 0) to be the first time for which $\int_0^{t_l^e} x_e(t)\,dt = \int_0^{t_l^e} \left[\sum_{S:e\in S} x_S(t)\right] dt = \frac{l}{2}$. In other words, t_l^e is the first time for which ONF has fractionally bought $\frac{l}{2}$ of sets containing e. Denote s_e the index of the last threshold time for e.

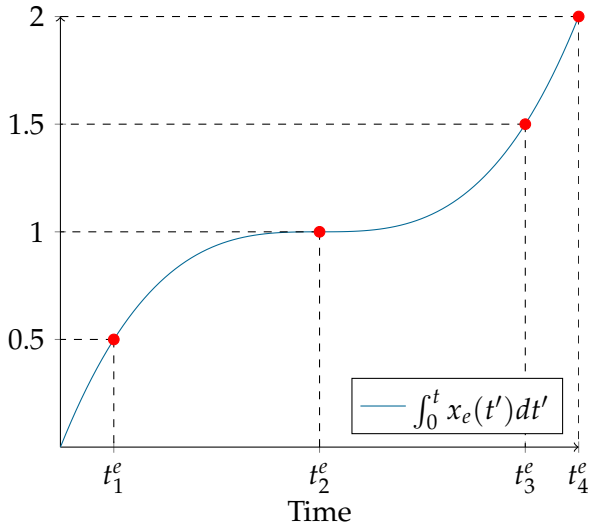


FIGURE 4.1. Example of threshold times

4.2.3 Algorithm description

We refer to the deterministic algorithm described here as OND (short for online deterministic).

For every $e \in \mathcal{E}$, the algorithm keeps track of a family of disjoint intervals $\mathcal{I}(e)$, which is initially $\emptyset$.

Define the potential function Φ:

$$\Phi = \sum_{S\in\mathcal{S}} n_S \cdot c(S) + \sum_{e\in\mathcal{E}} \sum_{\substack{I\in\mathcal{I}(e) \\ e \text{ has not been bought in } I}} n^{-1+2w_e(I)} \cdot c(e)$$

- n_S is the number of times S has been bought by the algorithm, initially 0.

- $c(S)$ is the cost of the set S.
- $w_e(I)$ is the fractional amount that e has been bought in the interval I so far (by ONF).

At the beginning, $\mathcal{I}(e) = \emptyset$. We want $\mathcal{I}(e)$ to consist of disjoint intervals as follows:

INVARIANT 4.3. *At any time t,*

- *If $t \leq t_1^e$, $\mathcal{I}(e) = \emptyset$ for all $e \in \mathcal{E}$.*
- *If $t \in (t_l^e, t_{l+1}^e]$, $\mathcal{I}(e) = \{(t_1^e, t_2^e],\ (t_2^e, t_3^e],\ ..., (t_{l-1}^e, t_l^e],\ (t_l^e, t]\}$.*

Define $I_e(t) = I$ if there exists an interval $I \in \mathcal{I}(e)$ such that $t \in I$. Otherwise, $I_e(t) = \text{NULL}$. In other words, $I_e(t)$ is the interval in $\mathcal{I}(e)$ that t is in. When we use this term, t will always refer to the current time, so we drop the t for convenience and use I_e.

The algorithm consists of a single routine, *UponThreshold*, which is called for e at every threshold time t_l^e. We can divide this function into three phases.

1. Buying phase A: If this is the first service, the algorithm skips this phase. Otherwise, denote t_0 the last time *UponThreshold* was called, that is, the last service time. Note that t_0 is the last threshold time of any element before the current service, which is not necessarily a threshold time of e.

For each set S, the algorithm performs the following steps:

(1) Let δ_S be the amount of S that ONF has fractionally bought from t_0 until now.
(2) For every element e' in S such that $I_{e'} \neq \text{NULL}$, update $w_{e'}(I_{e'})$ to be $w_{e'}(I_{e'}) + \delta_S$.
(3) Buy S if buying it decreases Φ.

We call buying in this phase *type A* buying.

2. Buying phase B: If $I_e \neq \text{NULL}$ and e has not been bought in the interval I_e, buy e by buying the cheapest set containing it. This is called *type B* buying.

3. Closing phase: Finally, OND closes the interval I_e (if one exists) and creates a new interval to be the next interval in $\mathcal{I}(e)$. Though not stated explicitly in the algorithm, we assume that each interval is left-open and right-closed to satisfy Invariant 4.3. Therefore immediately after

this service, I_e will point to the new interval I. Naturally $w_e(I)$ is set to be 0 as we're at the start of the interval.

By closing and starting intervals at threshold times, this phase ensures that OND correctly constructs $\mathcal{I}(e)$ as per Invariant 4.3.

Now we can present some intuition behind the definition of Φ. The first sum is simply the total cost of the algorithm so far, which we want to minimize. Choosing to buy a set increases the first sum but could decrease the second sum for each element in the set. Choosing to not buy a set on the other hand keeps the first sum unchanged but could increase the second sum. Hence OND simply makes the locally optimal choice when performing type A buying, that is, whatever choice minimises OND at the time. On the other hand, type B buying is necessary to ensure that before each interval in $\mathcal{I}(e)$ finishes, e is bought at least once.

Algorithm 4 Non-clairvoyant algorithm for SCD

procedure UPONTHRESHOLD(e)
  Let t be the current time
  ▷ **1. Buying phase A**
  for each $S \in \mathcal{S}$ **do**
    if this is the first service **then**
      break
    Let t_0 be the last service time
    ▷ how much ONF has bought S since the last service
    $\delta_S \leftarrow \int_{t_0}^{t} x_S(t')\, dt'$
    for each $e' \in S$ and $I_{e'} \neq$ NULL **do**
      $w_{e'}(I_{e'}) \leftarrow w_{e'}(I_{e'}) + \delta_S$
    $\Phi' \leftarrow$ value of Φ if we were to buy S
    ▷ buy S if that decreases Φ
    if $\Phi' \leq \Phi$ **then**
      Buy S
  ▷ **2. Buying phase B**
  if $I_e \neq$ NULL and e has not been bought in I_e **then**
    Buy the cheapest set containing e
  ▷ **3. Closing phase**
  if $I_e \neq$ NULL **then**
    Close the current interval I_e
  Start a new interval I in $\mathcal{I}(e)$ and set $w_e(I) \leftarrow 0$

4.2.4 Analysis

In this section, we bound the cost of OND within a $O(\log n)$ factor of the cost of ONF.

Let $ALG^f = ALG^f_d + ALG^f_p$ be the cost of the fractional solution, where ALG^f_d is the delay cost and ALG^f_p is the cost incurred by buying sets.

Similarly let $ALG = ALG_d + ALG_p$ be the total cost of OND, where ALG_d is the delay cost and ALG_p is the buying cost.

We will show that $ALG_d \leq ALG^f_d$ and $ALG_p \leq O(\log n)\ ALG^f_p$.

Upper bounding ALG_d

In this section, we show that for any request, OND serves it before ONF, thus guaranteeing that $ALG_d \leq ALG^f_d$.

This observation follows from Invariant 4.3.

OBSERVATION 4.4. *At the end of the algorithm,* $\mathcal{I}(e) = \{(t^e_1, t^e_2],\ (t^e_2, t^e_3],\ ...,\ (t^e_{s_e}, \infty)\}$

This observation follows from type B buying and Observation 4.4.

OBSERVATION 4.5. *At the end of the algorithm, for every* $e \in \mathcal{E}$ *and every* $I \in \mathcal{I}(e)$, *e was bought at least once in I unless I is the final interval* $(t^e_{s_e}, \infty)$.

LEMMA 4.6. *For every request q, OND serves q no later than ONF (fully) serves q.*

Proof. Let q be a request on an element e and r_q be the time q is released. Let t_q be the at which q is served by ONF, that is, $\int_{r_q}^{t_q} x_e(t)\ dt = 1$. We prove that the lifetime of q, defined by the interval $[r_q, t_q]$, fully covers at least one interval in $\mathcal{I}(e)$. Take the first threshold time after r_q, denoted $t^e_{l'}$ and consider two cases:

- $t^e_{l'} = t^e_1$ is the first threshold time. As t^e_2 is the first time at which e was fully bought by ONF, t^e_2 is also the first time that ONF could have fully served any request on e. Hence $t^e_2 \leq t_q$. This gives that $(t^e_1, t^e_2] \subseteq [r_q, t_q]$. The interval $(t^e_1, t^e_2]$ is in the family of intervals $\mathcal{I}(e)$, as required.

- $t^e_{l'}$ is not the first threshold time. Assume for contradiction that $t_q < t^e_{l'+1}$. From the definition of threshold time, $\int_{t^e_l}^{t^e_{l+1}} x_e(t)\, dt = \frac{1}{2}$ for all $1 \leq l \leq s_e$. We can upper bound $\int_{r_q}^{t_q} x_e(t)$:

$$\begin{aligned}
\int_{r_q}^{t_q} x_e(t)\, dt &= \int_{r_q}^{t^e_{l'}} x_e(t) + \int_{t^e_{l'}}^{t_q} x_e(t) \\
&\leq \int_{t^e_{l'-1}}^{t^e_{l'}} x_e(t) + \int_{t^e_{l'}}^{t_q} x_e(t) && \text{as } t^e_{l'-1} < r_q \\
&= \frac{1}{2} + \int_{t^e_{l'}}^{t_q} x_e(t) \\
&< \frac{1}{2} + \int_{t^e_{l'}}^{t^e_{l'+1}} x_e(t) && \text{as } t_q < t^e_{l'+1} \\
&= \frac{1}{2} + \frac{1}{2} = 1
\end{aligned}$$

  The strict inequality follows from the definition of t^e_{l+1} being the first time for which $\int_0^{t^e_{l+1}} x_e(t)\, dt = \frac{l+1}{2}$. $\int_{r_q}^{t_q} x_e(t) < 1$ is a contradiction as ONF fully bought e in the interval $[r_q, t_q]$. Hence $t^e_{l'+1} \leq t_q$ and we get that $(t^e_{l'}, t^e_{l'+1}] \subseteq [r_q, t_q]$. Clearly $(t^e_{l'}, t^e_{l'+1}]$ is in $\mathcal{I}(e)$.

Let I be the interval in $\mathcal{I}(e)$ such that $I \subseteq [r_q, t_q]$. I cannot be $(t^e_{s_e}, \infty)$ as $t_q < \infty$. By Observation 4.5, OND bought e at least once in I. Hence OND served q during I, before ONF fully served the request at time t_q. □

COROLLARY 4.7. *The total delay cost of OND is less than the delay cost of ONF.*

Proof. Using Remark 4.2 and Lemma 4.6 yields that OND always incurs less delay costs than ONF. □

Upper bounding ALG_p

In this section, we prove an upper bound on the value of Φ at the end of the algorithm (Lemma 4.11). Using this lemma, we can show that $\Phi \leq O(\log n)\ ALG^f_p$ (Lemma 4.13). Finally, we observe that ALG_p can be bounded by Φ.

To upper bound Φ, we consider buying phase A and B separately and show that the increase to Φ is bounded in each phase (Proposition 4.8 and Proposition 4.10).

PROPOSITION 4.8. *Buying phase A increases Φ by at most $\sum_{S\in\mathcal{S}} 2\delta_S c(S)\log n$, where δ_S is how much ONF has fractionally bought S from the last service time until the current service.*

Proof. The proof for this is probabilistic. We show that for each set $S \in \mathcal{S}$, either buying S or not buying S will increase Φ by at most $2\delta_S c(S)\log n$.

Fix a set S. Say we buy S with probability $1 - n^{-2\delta_S}$. Then the expected change to the first sum of Φ is $(1 - n^{-2\delta_S}) \cdot c(S)$. $1 - n^{-2\delta_S}$ can be bounded by $2\delta_S \log n$ using the Taylor series $e^{-2\delta_S \log n} = 1 - 2\delta_S \log n + \epsilon$, where $\epsilon > 0$. Hence the expected increase to the first sum of Φ is at most $2\delta_S c(S)\log n$.

Observe that the algorithm increases $w_e(I_e)$ by δ_S for every $e \in S$ if $I_e \neq$ NULL. Let the value of $w_e(I_e)$ before and after this step be w_e and $w_e + \delta_S$, respectively.

Fix any element $e \in S$ such that $I_e \neq$ NULL. If e has already been bought in I_e then the increase to $w_e(I_e)$ does not change Φ. Otherwise, buying S causes the term $n^{-1+2w_e(I_e)}$ to vanish. Using that $n^{-2\delta_S}$ is the probability of not buying S, the expected value of the this term in the second sum is $n^{-2\delta_S} \cdot n^{-1+2w_e+2\delta_S} = n^{-1+2w_e}$, which is equal to its old value n^{-1+2w_e}. Thus the expected increase to the second sum of Φ is 0.

Since *OND* always picks the choice that yields the lower value of Φ, the actual increase to Φ will be at most the expected increase. Summing over all sets gives the proposition. □

The following proposition is a precursor to Proposition 4.10.

PROPOSITION 4.9. *Consider a service for e at time t_l^e such that $2 \leq l \leq s_e$. After buying phase A, $w_e(I_e) = \frac{1}{2}$.*

Proof. Fix a service λ for e at time t_l^e. At this time, $I_e = (t_{l-1}^e, t_l^e]$ by definition.

As the algorithm starts a service at every threshold time, there was a service at time t_{l-1}^e. For every service after t_{l-1}^e and for every set S such that $e \in S$, *OND* adds δ_S to $w_e(I_e)$, where δ_S is the amount *ONF* has fractionally bought S since the previous service. This process continues until the current service λ.

Hence after λ's buying phase A, $w_e(I)$ is equal to how much *ONF* has bought any set containing e from t_{l-1}^e until t_l^e, which is $\frac{1}{2}$ by definition of threshold times. □

PROPOSITION 4.10. *Φ is not increased during buying phase B.*

Proof. Consider any service for some element e. By Proposition 4.9, $w_e(I_e) = \frac{1}{2}$ after buying phase A.

In buying phase B, if e has been bought in I_e then OND does not do anything so we assume otherwise.

By the definition of Φ, there is a term $n^{-1+2w_e(I_e)}c(e)$ in the second sum of Φ as long as e has not been bought in I_e. As $w_e(I_e) = \frac{1}{2}$ after buying phase A, this term is exactly $c(e)$. When OND buys the cheapest set containing e, this term vanishes, decreasing Φ by $c(e)$. At the same time, buying a set S increases the first term of Φ by $c(S)$. In this case, the increase is $c(e)$ as we buy the cheapest set containing e. Overall, Φ remains the same. □

LEMMA 4.11. *At the end of the algorithm, $\Phi \leq 2\log n \sum_{S\in\mathcal{S}} c(S) \int_0^\infty x_S(t)\, dt + \sum_{e\in\mathcal{E}} \sum_{I\in\mathcal{I}(e)} \frac{c(e)}{n}$.*

By Proposition 4.10, Φ can only be increased during buying phase A or in the closing phase. Let Φ_1 be how much Φ is increased during buying phase A and Φ_2 be how much Φ is increased during the closing phase over the entire algorithm. Clearly $\Phi = \Phi_1 + \Phi_2$ as Φ is 0 at the beginning of the algorithm. We bound Φ_1 by $2\log n \sum_{S\in\mathcal{S}} c(S) \int_0^\infty x_S(t)\, dt$ and Φ_2 by $\sum_{e\in\mathcal{E}} \sum_{I\in\mathcal{I}(e)} \frac{c(e)}{n}$ to obtain the lemma.

From Proposition 4.8, each service's buying phase A increases Φ by at most $\sum_{S\in\mathcal{S}} 2\delta_S c(S)\log n$, where δ_S is how much ONF has fractionally bought S since the last service. Summing over all services, we obtain $\Phi_1 \leq 2\log n \sum_{S\in\mathcal{S}} c(S) \int_0^\infty x_S(t)\, dt$ as $\int_0^\infty x_S(t)\, dt$ is how much ONF has bought S in total.

During the closing phase, Φ is only increased when a new interval is created. This step causes a new term $n^{-1+2w_e(I)}c(e)$ to be added to the second sum of Φ. As $w_e(I)$ is initially 0, this increases Φ by $n^{-1}c(e) = \frac{c(e)}{n}$. Summing over all elements in $\mathcal{E}$ and all intervals in $\mathcal{I}(e)$ gives $\Phi_2 \leq \sum_{e\in\mathcal{E}} \sum_{I\in\mathcal{I}(e)} \frac{c(e)}{n}$

PROPOSITION 4.12. $\sum_{I\in\mathcal{I}(e)} c(e) \leq 2 \cdot ALG_p^f$.

Proof. Recall that s_e is the index of e's last threshold time. By Observation 4.4, there are s_e intervals in $\mathcal{I}(e)$ and therefore $\sum_{I\in\mathcal{I}(e)} c(e) = s_e \cdot c(e)$ at the end of the algorithm. By the

definition of threshold times, $\int_0^{t^e_{s_e}} x_e(t)\,dt = \frac{s_e}{2}$. Thus we can bound s_e by $2\int_0^\infty x_e(t)\,dt$, giving that $\sum_{I\in\mathcal{I}(e)} c(e) \le 2\int_0^\infty x_e(t)c(e)\,dt$.

$\int_0^\infty x_e(t)\,dt$ is how much ONF spent buying sets containing e. Since $c(e)$ is the price of the cheapest set containing e, ONF's buying cost is at least $\int_0^\infty x_e(t)c(e)\,dt$, giving the desired inequality. □

We now prove that Φ is within $O(\log n)$ of the fractional buying cost.

LEMMA 4.13. $\Phi \le O(\log n)\ ALG_p^f$.

Proof. As ALG_p^f is how much ONF spent on buying sets, $ALG_p^f = \sum_{S\in\mathcal{S}} c(S)\int_0^\infty x_S(t)\,dt$. Hence we can upper-bound the left hand side as follows:

$$\begin{aligned}
\Phi &\le 2\log n \sum_{S\in\mathcal{S}} c(S)\int_0^\infty x_S(t)\,dt + \sum_{e\in\mathcal{E}}\sum_{I\in\mathcal{I}(e)} \frac{c(e)}{n} && \text{(Lemma 4.11)}\\
&= 2\log n\cdot ALG_p^f + \sum_{e\in\mathcal{E}}\sum_{I\in\mathcal{I}(e)} \frac{c(e)}{n}\\
&\le 2\log n\cdot ALG_p^f + n\cdot \max_{e\in\mathcal{E}} \sum_{I\in\mathcal{I}(e)} \frac{c(e)}{n}\\
&= 2\log n\cdot ALG_p^f + \max_{e\in\mathcal{E}} \sum_{I\in\mathcal{I}(e)} c(e)\\
&\le 2\log n\cdot ALG_p^f + 2\cdot ALG_p^f && \text{(Proposition 4.12)}\\
&= O(\log n)\ ALG_p^f
\end{aligned}$$

□

Finally, Corollary 4.7 and Lemma 4.13 can be used to prove the main theorem.

THEOREM 1.6. *There exists a $O(\log n \log k)$-competitive deterministic non-clairvoyant algorithm for set cover with delay.*

Proof. Recall the definition of Φ:

$$\Phi = \sum_{S\in\mathcal{S}} n_S c(S) + \sum_{e\in\mathcal{E}} \sum_{\substack{I\in\mathcal{I}(e)\\ e \text{ has not been bought in } I}} n^{-1+2w_e(I)} c(e)$$

Since n_S is the number of times OND has bought S, clearly the buying cost of OND is $\sum_{S \in \mathcal{S}} n_S c(S) \leq \Phi$. Thus we can bound $ALG_p \leq \Phi \leq O(\log n)\ ALG_p^f$ by Lemma 4.13.

Corollary 4.7 guarantees that OND pays less delay than ONF, ensuring $ALG_d \leq ALG_d^f$. Hence we get that $ALG = ALG_p + ALG_d \leq O(\log n)\ ALG^f$. Using that ONF satisfies Theorem 4.1, $ALG^f \leq O(\log k)\ OPT$, where OPT is the optimal cost.

Thus $ALG \leq O(\log n \log k)\ OPT$, giving the desired competitive ratio. □

4.3 Clairvoyant setting

The set-up for clairvoyant set cover with delay is identical to the non-clairvoyant setting. The only difference is that an online clairvoyant algorithm has immediate access to the entirety of each request's delay function when it arrives.

In this section, we present a $O(\log m)$-competitive algorithm and a $O(\log n)$-competitive algorithm for SCD by applying the delay framework in (Azar and Touitou, 2020). Both algorithms are clairvoyant and run in exponential time.

Since n and m are given at the beginning, an online algorithm could pick between the two aforementioned algorithms depending on which value is smaller, satisfying this theorem.

THEOREM 1.7. *There exists a $O(\min\{\log n, \log m\})$-competitive deterministic clairvoyant algorithm for SCD which runs in exponential time.*

4.3.1 Intuition

The network design problem defined in (Azar and Touitou, 2020) is identical to the general problem given in section 2.1. In this problem, we are given a set of weighted items $\mathcal{U}$ and a cost function on these items. As we're using the delay variant, each request q comes with a continuous non-decreasing delay function $d_q(t)$, which is fully revealed to the algorithm when q arrives.

The framework presented in (Azar and Touitou, 2020) makes calls to an approximation algorithm for the prize-collecting network design problem (PCND). In the prize-collecting problem, the input is a set of requests Q and a penalty function $\pi : Q \to \mathbb{R}^+$. A solution is

a subset of items $X \subseteq \mathcal{U}$ which serves some requests $Q' \subseteq Q$. The cost of the solution is $\sum_{x \in X} c(x) + \sum_{q \in Q \setminus Q'} \pi(q)$, that is, the cost of items bought plus the penalties for unserved requests. (Azar and Touitou, 2020) proves the following theorem:

THEOREM 4.14. *(Azar and Touitou, 2020) [Theorem 4.1]. If there exists a deterministic (randomized) γ-approximation algorithm for PCND which runs in polynomial time, then there exists a $O(\gamma \log |\mathcal{U}|)$-competitive deterministic (randomized) algorithm for network design with delay, which runs in polynomial time.*

However, if we don't restrict ourselves to polynomial running time algorithms, the framework can instead make calls to an optimal algorithm for PCND, giving the following theorem:

THEOREM 4.15. *(Azar and Touitou, 2020) [Theorem 5.11]. Given an optimal algorithm for PCND, there exists an $O(\log |\mathcal{U}|)$-competitive algorithm for ND with delay (with no guarantees on running time).*

4.3.2 Prize-collecting set cover

Recall that for the set cover with delay problem, $\mathcal{U} = \mathcal{S}$. The appropriate prize-collecting problem for SCD is the offline prize-collecting set cover problem (PCSC), where the input is a set of requests Q and a penalty function $\pi : Q \to \mathbb{R}^+$. We slightly abuse notation and write $q \in S$ if the element q occurs on is in the set S. The solution is a collection of subsets $\mathcal{S}'$ of cost $\sum_{q \in Q \setminus \cup_{S \in \mathcal{S}'} S} \pi(q) + \sum_{S \in \mathcal{S}'} c(S)$. Essentially, for every request q, the algorithm either buys a set S such that $q \in S$ or pays the penalty $\pi(q)$.

We show that this problem can be solved optimally offline if we allow for exponential running time.

THEOREM 4.16. *There exists an optimal algorithm for the prize-collecting set cover problem that runs in exponential time.*

Proof. Every collection of subsets is a potential solution to PCSD. Our algorithm thus considers every potential solution and picks one with the lowest cost. This algorithm is clearly optimal. There are $2^{|\mathcal{S}|}$ solutions to consider as the algorithm either buys a set or doesn't, making the running time exponential. □

Denote a call to this algorithm $PCSC(Q, \pi)$.

4.3.3 $O(\log m)$-competitive algorithm

We directly apply (Azar and Touitou, 2020)'s framework to the set cover with delay problem. Using that the optimal algorithm for prize-collecting set cover has exponential running time, Theorem 4.15 gives that there exists a $O(\log m)$-competitive clairvoyant algorithm for SCD.

THEOREM 4.17. *There exists a $O(\log m)$-competitive deterministic clairvoyant algorithm for SCD which runs in exponential time.*

4.3.4 $O(\log n)$-competitive algorithm

In this section, we make some changes to (Azar and Touitou, 2020)'s framework to obtain a $O(\log n)$ competitive ratio.

In the original framework, each request q is associated with a level ℓ_q. A service λ of level $\ell_\lambda = j + 1$ is started when the total residual delay of requests of level at most j reach 2^{j+1}. Here residual delay refers to part of the accumulated delay that the online algorithm has not paid for.

For a service λ, *cheap items* are those whose cost is at most $\frac{2^{\ell_\lambda}}{|\mathcal{U}|}$ or $\frac{2^{\ell_\lambda}}{m}$. Denote E_0^λ the set of cheap items. During every service, the framework makes calls to the algorithm for the prize-collecting problem in which the cost of cheap items is set to 0. Denote such a call with the subscript $E_0^\lambda \to 0$, for example, $PCSC_{E_0^\lambda \to 0}(Q, \pi)$.

We modify the definition of *cheap items* as follows. For a service λ, $E_0^\lambda = \{S_e \mid e \in \mathcal{S} \text{ and } c(S_e) \leq \frac{2^{\ell_\lambda}}{n}\}$, where S_e is the cheapest set containing the element e. Other than changing the definition of cheap items, the rest of the framework remains the same.

This change does not affect most of the original analysis. We omit proofs that remain identical to the original analysis and only restate statements that need to be adapted for our modification.

For readers referring to (Azar and Touitou, 2020)' paper, n is used there to denote the number of items in $\mathcal{U}$ but for the set cover with delay problem, we use it to denote the number of elements. Therefore n in the original paper is equal to m in our application.

PROPOSITION 4.18. *(Azar and Touitou, 2020) [Proposition 4.16]. The total cost of a service λ is at most $O(\lambda) \cdot 2^{\ell_\lambda}$.*

Proof. Each service cost consists of multiple components, one of which is the cost of cheap items. The remaining components are identical to the original framework so we do not restate the proof.

We show that the cost of E_0^λ is at most 2^{ℓ_λ}. By definition, there are at most n cheap sets in E_0^λ. Hence the cost of buying all sets in E_0^λ is $\sum_{S \in E_0^\lambda} c(S) \leq \sum_{S \in E_0^\lambda} \frac{2^{l_\lambda}}{n} \leq n \cdot \frac{2^{l_\lambda}}{n} = 2^{l_\lambda}$. □

The following is the counterpart to Proposition 4.20 in (Azar and Touitou, 2020), except the $\log m$ factor is replaced by $\log n$.

PROPOSITION 4.19. *(Azar and Touitou, 2020) [Proposition 4.20]. There exists a constant β such that for every optimal service λ^*, we have*

$$\sum_{\lambda \in \Lambda^0} \min\{2^{\ell_\lambda}, PCSC_{E_0^\lambda \leftarrow 0}(Q_{\lambda \cap \lambda^*}, \pi_{t_\lambda \to \tau_\lambda})\} \leq \beta \log n \cdot c(\lambda^*). \tag{4.1}$$

Proof. Fix any service λ^* in the optimal solution. Let t^* be $\max_{q \in Q_{\lambda^*}} r_q$, that is, the latest release time of all requests served by λ^*. $Q_{\lambda \cap \lambda^*}$ is the set of requests served by λ in our solution and served by λ^* in the optimal solution. Observe that we only need to consider services in Λ_0 such that $Q_{\lambda \cap \lambda^*} \neq \emptyset$, otherwise λ does not contribute anything to the left hand side of Equation 4.1 as $PCSC(\emptyset, \pi) = 0$.

τ_λ refers to the forwarding time of a service λ. We refer readers to the original algorithm for how this is determined for each service. $\pi_{t_\lambda \to \tau_\lambda}(q)$ gives the delay of request q from time t_λ to τ_λ, that is, $d_q(\tau_\lambda) - d_q(t_\lambda)$.

Most of the original proof remains unchanged. We're only concerned with case 2 of the proof, where $\tau_\lambda > t^*$. Define $\ell = \lfloor \log(c(\lambda^*)) \rfloor$, where $c(\lambda^*)$ is the cost of the service λ^*. As per the original proof, there is at most one service $\lambda \in \Lambda_0$ of level j such that $Q_{\lambda \cap \lambda^*} \neq \emptyset$ and $\tau_\lambda > t^*$. There are three sub-cases to consider for each service λ:

(1) $\ell_\lambda \leq \ell$. There are no changes to the original proof for this sub-case. The contribution to the left hand side of Equation 4.1 is $2 \cdot c(\lambda^*)$.

(2) $\ell < \ell_\lambda < \ell + \lceil \log(n) \rceil + 1$. Denote $SC(Q)$ the call to an optimal algorithm for regular set cover for a set of requests Q. Specifically, $SC(Q)$ returns a collection of subsets that can serve every request in Q. Calls to SC with the subscript $E_0 \leftarrow 0$ means that the cost of all sets in E_0 are set to 0. We have that:

$$\begin{aligned} \min\{2^{\ell_\lambda}, PCSC_{E_0^\lambda \leftarrow 0}(Q_{\lambda \cap \lambda^*}, \pi_{t_\lambda \to \tau_\lambda})\} &\leq PCSC_{E_0^\lambda \leftarrow 0}(Q_{\lambda \cap \lambda^*}, \pi_{t_\lambda \to \tau_\lambda}) \\ &\leq SC_{E_0^\lambda \leftarrow 0}(Q_{\lambda \cap \lambda^*}) \\ &\leq c(\lambda^*) \end{aligned} \tag{4.2}$$

The second last inequality holds as the set cover problem does not allow for a penalty function. The last inequality holds as λ^* served all requests in $Q_{\lambda \cap \lambda^*}$.

Summing Equation 4.2 over at most one λ from each level, the total contribution to the left hand side of Equation 4.1 from these levels is at most $\lceil \log n \rceil \cdot c(\lambda^*)$.

(3) $\ell_\lambda \geq \ell + \lceil \log n \rceil + 1$. We show that $PCSC_{E_0^\lambda \leftarrow 0}(Q_{\lambda \cap \lambda^*}, \pi_{t_\lambda \leftarrow \tau_\lambda}) = 0$. Clearly $PCSC_{E_0^\lambda \leftarrow 0}(Q_{\lambda \cap \lambda^*}, \pi_{t_\lambda \leftarrow \tau_\lambda}) \leq SC_{E_0^\lambda \leftarrow 0}(Q_{\lambda^*})$.

Let $\mathcal{E}_{\lambda^*}$ be the set of elements bought by λ^*. We prove that $\forall e \in \mathcal{E}_{\lambda^*},\ c(S_e) \leq \frac{2^{\ell_\lambda}}{n}$, implying that all elements in $\mathcal{E}_{\lambda^*}$ can be covered by subsets in E_0^λ with cost 0, giving $SC_{E_0^\lambda \leftarrow 0}(Q_{\lambda^*}) = 0$.

Assume for the sake of contradiction that there exists $e \in \mathcal{E}_{\lambda^*}$ such that $S_e \notin E_0^\lambda$ and therefore $c(S_e) > \frac{2^{\ell_\lambda}}{n}$. Then $c(\lambda^*) \geq c(S_e) > \frac{2^{\ell_\lambda}}{n}$ since $e \in \mathcal{E}_{\lambda^*}$ and by definition of S_e being the cheapest set containing e. By the constraint on ℓ, $c(\lambda^*) \leq 2^{\ell+1} \leq 2^{\ell_\lambda - \lceil \log n \rceil} \leq \frac{2^{\ell_\lambda}}{n}$, which is a contradiction.

□

LEMMA 4.20. *(Azar and Touitou, 2020) [Proposition 4.19].* $\sum_{\lambda \in \Lambda_0} 2^{\ell_\lambda} \leq O(\log n) \cdot OPT$.

Proof. The proof is identical to the original paper, except that $O(\log m)$ is replaced by $O(\log n)$ in the inequality due to Proposition 4.19. □

Overall, we obtain the counterpart to Theorem 4.15 for the SCD problem.

THEOREM 4.21. *Given an optimal deterministic algorithm for PCSC, there exists a $O(\log n)$-competitive deterministic algorithm for SCD.*

Using that there exists an optimal algorithm for PCSC that runs in exponential time (Theorem 4.16), the following theorem holds.

THEOREM 4.22. *There exists a $O(\log n)$-competitive deterministic clairvoyant algorithm for SCD runs in exponential time.*

CHAPTER 5

Conclusion

In this thesis, we study non-clairvoyant problems with deadlines or delay and present competitive algorithms for these problems.

We provide $O(\sqrt{n})$-competitive non-clairvoyant algorithms for joint replenishment with deadlines/delay and show that they are optimal up to a constant factor by providing a matching lower bound. For the more general multi-level aggregation problem, we give a $O(D + \sqrt{n})$-competitive non-clairvoyant algorithm for the deadline variant. An open problem is to close this gap or provide a similar upper bound for the delay variant.

For set cover with delay in the non-clairvoyant setting, we settle the problem completely by providing a deterministic $O(\log k \log n)$-competitive algorithm which runs in polynomial time, matching the previous upper bound by a randomized algorithm. We also show that there's a competitive gap of $\log n$ between clairvoyant SCD and non-clairvoyant SCD by providing a $O(\min\{\log n, \log m\})$-competitive clairvoyant algorithm for SCD. This is also closer to the known lower bound of $O(\sqrt{\log n})$ and $O(\sqrt{\log m})$ for clairvoyant SCD, though an open problem is to bridge this gap.

There exist other problems that can be studied in the non-clairvoyant setting, such as Steiner tree problems. A $O(\sqrt{n} \log n)$-competitive ratio exists for edge-weighted Steiner tree with deadlines by using the well known randomized metric embedding technique and our non-clairvoyant algorithm for MLAP with deadlines. An open problem is to obtain better deterministic algorithms for this class of problems.

Bibliography

Noga Alon, Baruch Awerbuch, Yossi Azar, Niv Buchbinder, and Joseph Naor. 2003. The online set cover problem. In Lawrence L. Larmore and Michel X. Goemans, editors, *Proceedings of the 35th Annual ACM Symposium on Theory of Computing, June 9-11, 2003, San Diego, CA, USA*, pages 100–105. ACM.

Noga Alon and Joel H. Spencer. 2008. *The Probabilistic Method, Third Edition*. Wiley-Interscience series in discrete mathematics and optimization. Wiley.

Yossi Azar, Ashish Chiplunkar, Shay Kutten, and Noam Touitou. 2020. Set cover with delay - clairvoyance is not required. In Fabrizio Grandoni, Grzegorz Herman, and Peter Sanders, editors, *28th Annual European Symposium on Algorithms, ESA 2020, September 7-9, 2020, Pisa, Italy (Virtual Conference)*, volume 173 of *LIPIcs*, pages 8:1–8:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

Yossi Azar and Noam Touitou. 2019. General framework for metric optimization problems with delay or with deadlines. In David Zuckerman, editor, *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 60–71. IEEE Computer Society.

Yossi Azar and Noam Touitou. 2020. Beyond tree embeddings - a deterministic framework for network design with deadlines or delay. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 1368–1379. IEEE.

Shai Ben-David, Allan Borodin, Richard M. Karp, Gábor Tardos, and Avi Wigderson. 1990. On the power of randomization in online algorithms (extended abstract). In Harriet Ortiz, editor, *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, May 13-17, 1990, Baltimore, Maryland, USA*, pages 379–386. ACM.

Shai Ben-David, Allan Borodin, Richard M. Karp, Gábor Tardos, and Avi Wigderson. 1994. On the power of randomization in on-line algorithms. *Algorithmica*, 11(1):2–14.

Marcin Bienkowski, Martin Böhm, Jaroslaw Byrka, Marek Chrobak, Christoph Dürr, Lukáš Folwarczný, Lukasz Jez, Jiri Sgall, Nguyen Kim Thang, and Pavel Veselý. 2016. Online algorithms for multi-level aggregation. In Piotr Sankowski and Christos D. Zaroliagis, editors, *24th Annual European Symposium on Algorithms, ESA 2016, August 22-24, 2016, Aarhus, Denmark*, volume 57 of *LIPIcs*, pages 12:1–12:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

Marcin Bienkowski, Jaroslaw Byrka, Marek Chrobak, Lukasz Jez, Dorian Nogneng, and Jirí Sgall. 2014. Better approximation bounds for the joint replenishment problem. In Chandra Chekuri, editor, *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 42–54. SIAM.

Niv Buchbinder, Moran Feldman, Joseph (Seffi) Naor, and Ohad Talmon. 2017. *O*(depth)-competitive algorithm for online multi-level aggregation. In Philip N. Klein, editor, *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 1235–1244. SIAM.

Niv Buchbinder, Tracy Kimbrel, Retsef Levi, Konstantin Makarychev, and Maxim Sviridenko. 2008. Online make-to-order joint replenishment model: primal dual competitive algorithms. In Shang-Hua Teng, editor, *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2008, San Francisco, California, USA, January 20-22, 2008*, pages 952–961. SIAM.

Niv Buchbinder and Joseph Naor. 2009. The design of competitive online algorithms via a primal-dual approach. *Found. Trends Theor. Comput. Sci.*, 3(2-3):93–263.

Rodrigo A. Carrasco, Kirk Pruhs, Cliff Stein, and José Verschae. 2018. The online set aggregation problem. In Michael A. Bender, Martin Farach-Colton, and Miguel A. Mosteiro, editors, *LATIN 2018: Theoretical Informatics - 13th Latin American Symposium, Buenos Aires, Argentina, April 16-19, 2018, Proceedings*, volume 10807 of *Lecture Notes in Computer Science*, pages 245–259. Springer.

Marek Chrobak, Mordecai J. Golin, Tak Wah Lam, and Dorian Nogneng. 2015. Scheduling with gaps: New models and algorithms. In Vangelis Th. Paschos and Peter Widmayer, editors, *Algorithms and Complexity - 9th International Conference, CIAC 2015, Paris, France, May 20-22, 2015. Proceedings*, volume 9079 of *Lecture Notes in Computer Science*, pages 114–126. Springer.

Daniel R. Dooly, Sally A. Goldman, and Stephen D. Scott. 2001. On-line analysis of the TCP acknowledgment delay problem. *J. ACM*, 48(2):243–273.

Bala Kalyanasundaram and Kirk Pruhs. 1993. Online weighted matching. *J. Algorithms*, 14(3):478–488.

Anna R. Karlin, Claire Kenyon, and Dana Randall. 2001. Dynamic TCP acknowledgement and other stories about e/(e-1). In Jeffrey Scott Vitter, Paul G. Spirakis, and Mihalis Yannakakis, editors, *Proceedings on 33rd Annual ACM Symposium on Theory of Computing, July 6-8, 2001, Heraklion, Crete, Greece*, pages 502–509. ACM.

Steven S. Seiden. 2000. A guessing game and randomized online algorithms. In F. Frances Yao and Eugene M. Luks, editors, *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing, May 21-23, 2000, Portland, OR, USA*, pages 592–601. ACM.