I got 99 problems but they're actually 1

A reduction hierarchy for online problems with metric spaces and delay

THOMAS SCHWARZ
SID: 490387586

Supervisor: Dr. Seeun William Umboh

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

School of Computer Science
The University of Sydney
Australia

6 November 2022

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: Thomas Schwarz

Signature: [illegible] **Date**: November 6, 2022

Abstract

Many online problems involve a metric space and requests with delay, where the requests arrive on the points of the metric space, the cost of fulfilling a request is related to the metric space, and the time at which we fulfil them can be delayed by paying the delay cost. This includes the joint replenishment problem, the multi-level aggregation with delay problem, the online facility location problem, and the online service with delay problem. While these problems all have a metric space and receives requests with delay, they greatly differ in how those requests are fulfilled and the costs you pay to fulfil them. We investigate how different these problems really are, and find that they exist in a reduction hierarchy - which we present and give the online reductions for - and that they all can be reduced to a new problem we introduce, group online service with delay.

All these reductions are able to induce a corresponding online algorithm with a preserved competitive ratio. In group online service with delay, requests arrive on a group of points, and can be fulfilled by having the server visiting any one point in the group.

We also show it is impossible to reduce any of these problems, under certain assumptions that apply to most reasonable online reductions, to the metrical task system without a superpolynomial growth in the size of the metric space - thus requiring a superpolynomial competitive ratio for the online reduction. This is a novel and surprising result, as the metrical task system is a very flexible problem and thus considered easy to make reductions to. To show this we also introduce a formalised definition of an online reduction, and several categories of 'reasonable' reductions that one must meet to meaningfully relate problems.

Acknowledgements

I would like to thank my friends, my family, and my supervisor Dr Seeun William Umboh for all of their help, guidance and support.

CONTENTS

List of Figures

CHAPTER 1

Introduction

Many naturally arising problems involve point-like concepts with a notion of a distance between them and requests that arrive over time but whose fulfilment can be delayed to a moment of our choosing. These problems with metric spaces and delay constitute a broad category of online problems and includes the joint replenishment problem, the multi-level aggregation with delay problem, the online facility location problem, online service with delay and the metrical task system problem. While these problems all feature a metric space and requests with delays, they also differ substantially both in their nature and their best known results. We ask how different are these problems really, and how are they still related to each other?

We formally show all these problems are related to each other and exist in a reduction hierarchy shown in fig 1.1. While some of these relations were already known as special cases of each other or as a loose relation in a non-formal sense within the folklore, other relations such as that between online facility location and online service with delay were not known.

With online problems, where information is slowly revealed to our algorithm, and our algorithm must continually make decisions that can't be taken back, outputting the optimal result is generally impossible. As a result we instead care about the ratio between what our algorithm paid, and what the optimal algorithm that knows everything in advance would pay, and we call an algorithm with such a 'competitive ratio' of α, α-competitive.

Thus, in order to formally relate these problems to each other we adapt the notion of a reduction between problems to the online setting, accounting for the change in focus of traditional reductions on preserving time-complexity to preserving the competitive-ratio while respecting the restrictions on information reveal and decision lock-in inherent to the online setting. This formalised notion of an online reduction continues to capture the previously known reductions in the folklore but now allows for novel more complicated reductions to be made, revealing strong relations between problems whose surface differences require the more complicated framework to analyse. Formalising online reductions also allows us to show restrictions on all reductions between certain problems.

While understanding the relations between problems is interesting in its own right, it more importantly helps reveal what about a problem is challenging. For example, knowing joint replenishment problem cannot be reduced to metrical task system without creating an exponentially larger metric space, as we show, indicates that JRP's memory of which requests have been fulfilled is a significant distinction from MTS's memory-less nature.

The area of online algorithms is full of mysteries. For example, the work function algorithm (WFA) was shown optimal on metrical task system in Chrobak and Larmore (1998), and yet while that same algorithm adapted for the k-Server problem is the best known for that problem, we still don't know if it's optimal and there have been no major improvements since Koutsoupias and Papadimitriou (1995), leaving the k-server conjecture one of the biggest open questions in online algorithms. Given the WFA is $2n-1$-competitive for MTS, which is optimal, and the WFA is also $2k-1$-competitive for k-Server - but after nearly three decades we still don't know if that's optimal - having a way to interrogate the differences between different online problems seems useful. Thus, the use of online reductions as a new technique to relate online problems to each other is probably the most important part of this thesis, although the various results themselves are also novel and interesting.

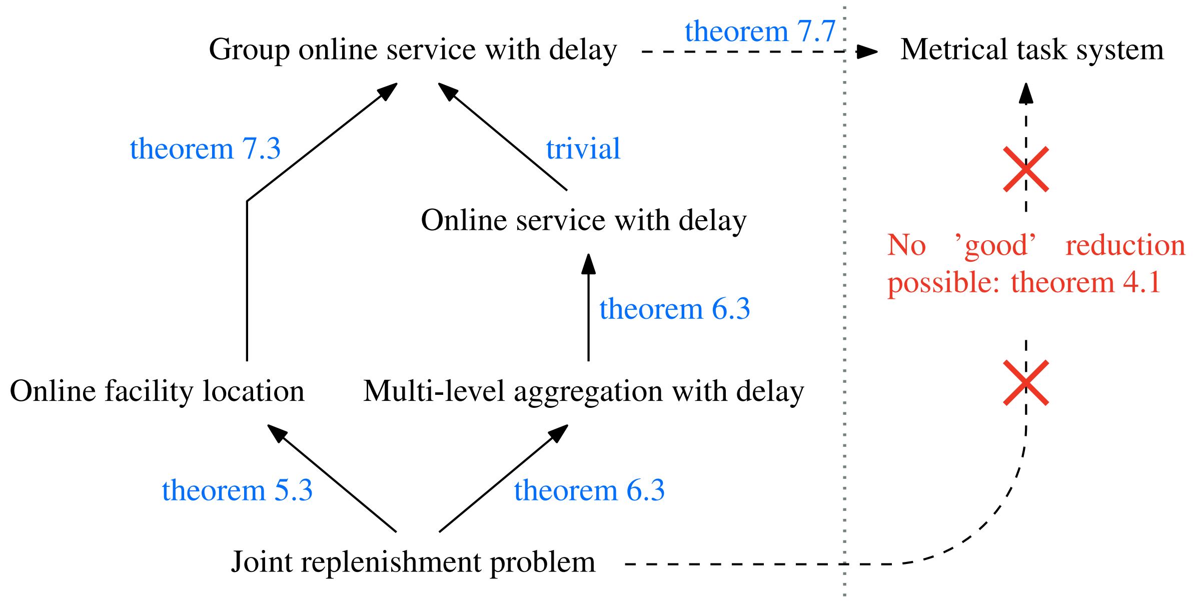

Group online service with delay
theorem 7.7
Metrical task system
theorem 7.3
trivial
Online service with delay
No 'good' reduction possible: theorem 4.1
theorem 6.3
Online facility location
Multi-level aggregation with delay
theorem 5.3
theorem 6.3
Joint replenishment problem

FIGURE 1.1. A reduction hierarchy relating the problems we investigate

1.1 Problems and Background

We first give a brief recap of some relevant concepts, and then introduce briefly the problems that we investigate in this thesis. We also give a detailed formal definition of each problem in the relevant sections.

In a 'normal', non-online, or offline problem, an algorithm is given the entire problem instance at once, and then - after some calculations - returns an answer for that instance. However an online problem takes place over time, with information slowly revealed, and decisions made, as time progresses.

When we evaluate an online algorithm we care about how well that algorithm does compared to the optimal choice when given all the information in advance. We use ALG to denote the cost of the algorithms' decisions on some input, and OPT the cost of the optimal decisions on that same input. We say an algorithm is k-competitive if, for any possible input, $ALG \leq k \cdot OPT + b$ for some constant b.

Many of our problems involve a metric space. A metric space has a set of point P, and a function $d : P \times P \to \mathbb{R}_{\geq 0}$ which is distance-like. In particular:

- The distance from a point to itself is always 0: $d(x, x) = 0$ for all x in P.
- The distances between any two different points is always larger than 0: $d(x, y) > 0$ for any x, y in P with $x \neq y$.
- The distance from point x to y is the same as from y to x: $d(x, y) = d(y, x)$, for all pairs of points x and y
- The distances follow the triangle inequality. In particular, you can't get from x to y for a shorter distance by detouring through some third point z: $d(x, y) \leq d(x, z) + d(z, y)$.

In this thesis we only deal with metric spaces that have finitely many points. Most metric spaces we encounter are represented as graphs with positive edge weights, where the distance from one point to another is the length of the shortest path between them in the graph.

In this thesis we investigate the following problems:

1.1.1 Joint replenishment problem (JRP)

In the variation of joint replenishment problem we study we are given a fixed ordering cost, a set of n items, and an order cost for each item. Requests for items arrive over time, with each request including a positive non-decreasing function representing the delay cost. We serve a request by including the item its for in an order, and paying the delay cost for how long that request waited to be served. When we serve an order we pay the order cost, which is the sum of the fixed ordering cost and the item order cost for each type of item we serve. An order that fulfils multiple requests for the same item still only pays for that item once.

1.1.2 Multi-level aggregation with delay (MLAP)

In multi-level aggregation with delay we are given a tree with n leaves. Requests for leaf nodes in the tree arrive over time, again with each request including a delay cost. We serve requests by including the node that request is for in an order and pay the delay cost for how long that request waited. When we serve an order, we pay the weight of the edges needed to connect each node in our order to the root of the tree.

1.1.3 Online facility location (facility)

In the variation of online facility location we study, we are given a set of points our customers are at, a set of potential facility locations, and for each potential facility location a facility rental cost. We have a metrical distance function defined over all pairs of points, regardless of if a point is a customer or a potential facility location. Requests arrive at our customer demand points, with delay costs that grow based off how long that request waits to be fulfilled. At any time we can rent a facility for that facilities' rental cost and can then connect it to customers - paying the distance cost between the facility and the customer to connect them and fulfilling that customers' requests.

1.1.4 Online service with delay (OSD)

In online service with delay we are given a metric space with n points. Requests arrive at points in the metric space over time, each including a non-decreasing delay cost function. We have a server which sits at some point in our metric space and can move at any time to another point, but has to pay the distance cost between where it is and where it wants to go when it moves. When our server visits a point it fulfils all requests at that point, but we pay the delay cost based off how long we made each request wait.

1.1.5 Group online service with delay (gOSD)

In group online-service with delay requests - instead of arriving at a single point in our metric space - arrive at a non-empty set of points and are fulfilled when our server moves to any point the request arrived at. All other details are the same as online service with delay.

1.1.6 Metrical task system (MTS)

In metrical task system we are given a metric space with n points. Our server again sits at some point in the metric space and can move between points by paying the distance cost between them. We receive a series of cost vectors where each cost vector specifies, for every state in the metric space, how much it costs to serve that cost vector from that state. We can then move to whatever state we wish for the distance cost or choose to stay where we are, but must then serve the cost vector, after which we receive the next cost vector and repeat.

1.2 Our contributions

Our most interesting contribution is theorem 4.1 which - with corollary 4.6 and the reductions referenced in fig 1.1 - show every online reduction, that meets certain broad assumptions, from any of JRP, MLAP, online facility location, OSD or gOSD to MTS, must create an MTS instance with a metric space that has superpolynomially many points with regards to the size of the metric space of the original problem. As the competitive ratio of MTS is linear with the size of its metric space, this means MTS cannot directly help us with solving any of those problems. Not only is this an interesting and surprising result on its own, but the technique is entirely novel and could be used to formally reveal the relationship between other online problems.

Another major contribution is the discovery of a formal reduction hierarchy, shown in fig 1.1, between many online problems involving metric spaces and requests with delays, which we formally show to be strongly related to each other with an online reduction. While some of these reductions - such as from JRP to MLAP - were already known, other problems - such as MLAP and OSD - for which it was only known that they were similar, now have formal reductions to each other thus showing they are highly related. Finally, some relations - such as the relationship between facility and OSD - were shown for the first time, and led to the introduction of a new problem, group online service with delay - a generalisation of OSD - which both OSD and facility can be reduced to.

Both of these contributions relied on our formalisation of the concept of an online reduction, our third major contribution. While reductions have occasionally appeared in the online algorithms literature, it is generally either to trivially relate problems as generalisations or simplifications of one another, or as a once-off that merely uses it for a very specific purpose that doesn't require properly engaging with adapting reductions to an online setting. As I needed to provide online reductions for many problems - several of which substantially differ from each other - and theorem 4.1 needed to deal with all possible online reductions between JRP and MTS - it was necessary to formalise online reductions, including deciding what restrictions should be placed on them. Formalising online reductions allowed us to make the two major contributions listed earlier, and can be applied as a tool to relate many more online problems than what I had the time to investigate, thus making it a major contribution in its own right.

We also make several contributions outside of the core reduction hierarchy focus of this thesis, including the introduction of the group online service with delay problem and the first deterministic algorithm for it, as well as a proof that the offline version of the online service with delay problem is NP-hard.

1.3 Related work

In this section I first investigate existing uses of online reductions in the literature, and secondly the current progress on each of our investigated problems.

1.3.1 Online reductions

Reductions between online problems are not commonly used in the literature, and generally appear in isolation as a tool in some specific proof. For example, a reduction from metrical task system to competitive vertex recoloring is used in Azar et al. (2022) to show competitive vertex recoloring has a $\Omega(n)$-competitive lowerbound. They show every MTS instance can be reduced to a vertex recoloring instance, with an $O(f(n))$-competitive algorithm in vertex recoloring implying an $O(f(n))$-competitive algorithm for MTS. As Chrobak and Larmore (1998) show a $(2n-1)$-competitive lowerbound for MTS, any online algorithm for vertex recoloring with a sub-linear competitive ratio would imply an online sublinear competitive algorithm for MTS, a contradiction, thus through the reduction proving the lowerbound. This reduction respects online problems' slow reveal of information over time: given the initial MTS input it constructs some initial vertex recoloring input, and then when each MTS input is received, immediately sends a corresponding input in the vertex recoloring. The translation of an algorithms' actions in vertex recoloring back to an action in the MTS instance is also done in an online fashion, and is done without memory, depending only on the latest recoloring.

Aside from this, most online reduction usage is trivial and merely to show a problem can be solved by focusing on a special case, or that some special case can be solved by a more general problem that is trivially related. For example, Chen et al. (2022) reduce cardinality joint replenishment problem to a special case, piecewise cardinality JRP, where the delay cost functions are piecewise linear functions. The reduction takes instances of cardinality JRP, and approximates their delay cost functions by piecewise linear functions, producing a piecewise cardinality JRP instance, with a loss of at most an $O(1)$ amount in the competitive ratio.

1.3.2 Problems

Due to the large number of problems investigated, and our focus on investigating the relations between these problems rather than improving their competitive ratio, we focus on the current best known results for each problem instead of the history of progress in each problem.

Metrical task system was introduced by Chrobak and Larmore (1998) and shown in that paper to have a tight competitive ratio of $(2n-1)$-competitive, with this ratio shown to be achieved by the work function algorithm. Subsequent progress has focused on randomised algorithms for MTS, for example with Bubeck et al. (2021) presenting an $O(\log^2 n)$-competitive randomised algorithm by embedding the metric space in a hierarchically separated tree.

We study the online make-to-order with delay version of classical joint replenishment problem. Buchbinder et al. (2013) use a linear programming approach to achieve a 3-competitive algorithm, and also present a lower bound of approximately 2.64 for any deterministic algorithm.

Azar and Touitou (2019) use a similar framework applied to each of multilevel aggregation with delay, online facility location and online service with delay to achieve new results for all three problems. These include an $O(D^2)$-competitive algorithm for MLAP with D the depth of the tree, an $O(\log^2 n)$-competitive algorithm for online facility location with delay and an $O(\log^2 n)$-competitive algorithm for online service with delay. This paper indicates that all these problems are somewhat related to each other, with the same general framework working for all of them, however it is notable that this is only a framework, with a separate algorithm still made for each problem, which is then analysed separately each time, and that Azar and Touitou (2019) still treat these as three separate problems and don't investigate the relationship between them. The MLAP and facility results are then improved in Azar and Touitou (2020) to an $O(\log n)$-competitive ratio for both of them. Note a slightly more specific case of facility location than what we investigate is studied in Azar and Touitou (2019) and Azar and Touitou (2020).

Bienkowski et al. (2016) show a 4-competitive lowerbound for MLAP, for the special case where the tree is a path, which is also the best known lowerbound for general MLAP. No known lower bounds currently exist for facility location and OSD. While we study the clairvoyant case of multi level aggregation with delay - where we are given the full delay cost function of a request as soon as it arrives - the non-clairvoyant case is also interesting and has recently been shown by Le et al. (2023) to be $O(\sqrt{n} + D)$-competitive, with D the depth of the tree, with a $\Omega(\sqrt{n})$-competitive lowerbound for non-clairvoyant JRP with deadlines also applying to MLAP.

1.4 Thesis outline

In chapter 2 we introduce a formal definition of an online reduction. This requires us to first introduce a formal definition of an online problem, and some related concepts, so we can talk about turning an instance of one problem into another problem. We also introduce an 'order preserving online reduction', a stronger notion of an online reduction where the relative costs of the different options an algorithm can choose are preserved between the original problem and the problem instance we've reduced it to. Chapter 2 does not present any new results, but is important as it sets up the online reduction framework that we use in the rest of the thesis.

All subsequent chapters, aside from chapter 8, focus on using these online reductions to show relations between problems and thus build up the results that make up fig 1.1.

We start in chapter 3 by defining the joint replenishment problem (JRP) and the multi level aggregation with delay problem (MLAP), and then relating them with an exactly cost preserving online reduction between them. Chapter 3 is intended to be a gentle introduction to using the online reduction framework just given in chapter 2, with the actual reduction from JRP to MLAP not technically challenging and not a novel result that would surprise anyone familiar with the area. Nevertheless it is a nice introductory application of our online reduction framework and also shows our online reduction framework captures existing reductions known in folklore and the literature.

In chapter 4 we present the most interesting result of this thesis, theorem 4.1. We first define the metrical task system (MTS) problem. We then prove theorem 4.1, showing there are no ‘good’ reductions - for a specified definition of good - between JRP and MTS. By corollary 4.6 this ‘there are no good reductions to MTS’ property spreads from JRP to any problem we can reduce JRP to without growing the metric space, and thus applies to MLAP and the reductions we show in subsequent chapters.

In chapter 5 we define online facility location (facility) and provide a reduction to it from JRP. This chapter is not technically challenging, but the relationship shown between JRP and facility is novel.

In chapter 6 we define online service with delay (OSD) and provide a reduction from multi level aggregation with delay (MLAP) to OSD.

In chapter 7 we introduce the group online service with delay (gOSD) problem, a novel problem that applies the common group generalisation of a problem to OSD. We then present a reduction from online facility location to gOSD, and a ‘bad’ reduction that per theorem 4.1 must, and does, grow the metric space superpolynomially. This completes the reduction hierarchy given in fig 1.1.

In chapter 8 we give a proof that finding the offline optimal solution for OSD is NP-hard. We then state some implications about reductions from OSD to MTS, which while separate from the fig 1.1 reduction hierarchy focus of the rest of this thesis, this result is what directly inspired theorem 4.1 and the subsequent investigation into and discovery of that reduction hierarchy. Thus despite it’s change in focus from the rest of this thesis, chapter 8 gives useful context and is a nice ending result for the thesis.

In table A.1 we give a summary table of most of the common notation used in this thesis.

CHAPTER 2

Online Reductions

In this chapter we motivate and provide a definition for a reduction between two online problems. To do this we must first provide a definition of an online problem and notation to talk about a problem, so we can then talk about how to relate these problems to each other.

We then provide several stronger definitions with the goal of ensuring an online reduction from problem P to R causes a competitive algorithm for problem R to induce a competitive algorithm on problem P, and further stronger definitions to ensure an online reduction shows problem P and R are meaningfully related to each other.

When you formally define something you have the freedom to make whatever choices you want, but if you want your definition to be useful, you need to try to make those choices so as to capture some existing informal idea, and hopefully to do so in a way that can be used as a useful tool to show something. Thus we firstly give, in section 2.1, an informal walk-through of some of the key existing ideas of online reductions we're trying to capture with our definition, and key restrictions we need that will make our definition useful. Subsequent sections then give those definitions, generally by giving the informal intuition followed by a formal definition. In order to keep our definition of an online reduction broad and to be clear about what conditions we achieve or require with our theorems, many of those key restrictions that make a definition useful are given as progressively stronger classes of an online reduction, such as a competitive-ratio preserving online reduction, or an exactly cost-preserving online reduction.

2.1 Motivation

Online problems inherently differ from offline problems in their goal, their nature, and the challenges they face, which requires introducing a definition of a reduction specific to online problems.

When designing online problems we are interested in achieving a good competitive ratio: thus as the definition of a reduction for offline problems in complexity theory is guided by the goal of inducing a polynomial time algorithm, our online reduction definition is guided by the goal of inducing a competitive algorithm. In particular: If you can reduce one online problem P to another online problem R, the existence of an α-competitive online algorithm for R should induce an online algorithm for P, with a competitive ratio related to α.

In particular, we have the following algorithm for problem P that is given the algorithm for problem R as a black-box:

(1) Given the starting input for some instance of problem P, construct some starting input for a problem R instance
(2) Whenever we're given part of the input sequence in problem P, construct some corresponding part of the input sequence for problem R and gave it to the algorithm for R
(3) Whenever the algorithm for problem R takes some action, translate this into some action to take in problem P.

Some further restrictions on the reduction, described below, then gives an α-competitive algorithm.

While this is sufficient for a reduction to create an online algorithm for problem P, it is not sufficient to show that problem P and problem R are meaningfully related. For example given any two problems P and R, if we have an algorithm for problem P, we can reduce P to R by:

(1) Solving problem P using our algorithm for problem P
(2) Creating a trivial instance of problem R
(3) Mapping the solution to that trivial instance back to the solution for problem P we found in step 1.

This is the online equivalent of reducing any decision problem to, for example, subset-cover by

(1) Solving that problem
(2) Reducing any 'yes' instance to the subset-cover instance 'Can $\{1, 2\}$ be covered by $\{1, 2\}$' and any 'no' instance to the subset-cover instance 'Can $\{1, 2\}$ be covered by $\{1\}$'

which - while it does technically induce an algorithm for any decision problem - does not meaningfully relate all decision problems to subset-cover. Complexity theory solves this problem by providing restrictions on reductions: namely requiring the reduction to complete in polynomial time.

Thus we need to provide some further restrictions on our reduction to ensure that the existence of an online reduction from P to R shows that problems P and R are actually related to each other. These restrictions are motivated by two goals:

- A reduction from problem P to problem R, where R has an α-competitive algorithm, should provide an algorithm for problem P, with a corresponding upper bound on the competitive ratio for problem P
- When solving an instance of problem P by reducing it to problem R, the algorithm for problem R should perform the hard work of actually solving the problem, with the reduction merely rephrasing the problem.

We provide the motivation of these restrictions when we provide the relevant definitions.

2.2 Defining online problems

2.2.1 Intuition

In any online problem instance we are first given some starting input, such as a set of items we can sell or a metric space. Time then proceeds, with information revealed to our algorithm over time and our algorithm making decisions over time. The online problem gives us a cost we are trying to minimise, based off the decisions we have made.

We view time as a series of m discrete time-steps, with in each time-step the instance revealing some piece of information to the algorithm, and the algorithm then making some decision.

An online problem is thus a set of valid starting input $\bar{S}$, a function mapping a particular starting input to a set of valid time-step inputs, a function mapping some starting input and time-step input to a set of valid decisions an algorithm can make, and a cost function.

An instance of an online problem is then a particular starting input and time-step input.

An algorithm makes decisions over time, with a sequence of decisions made so far called a strategy.

Our notion of a time-step input is similar to the literature's input sequence and captures the same idea, but differs in that we require some kind of input - potentially an 'empty' input signifying no new information has arrived - at every discrete moment of time, while an input sequence generally just specifies when new interesting information arrives. We make this change as it simplifies the relationship of time between our reduced and primal problems, a concern important to us but not normally present in the literature.

2.2.2 Formal definition

DEFINITION 2.1. *An online problem $(\bar{S}, \bar{T}, \bar{D}, c)$ consists of:*

- *A set $\bar{S}$ of all valid starting input instances.*
- *A function $\bar{T}(S) : \bar{S} \rightarrow 2^{\{(T_1, T_2, \ldots, T_m) | m \in \mathbb{N}\}}$ that, for a given starting input S, produces the set of all valid time-step input sequences.*
- *A function $\bar{D}(S, T)$ that for a given starting input $S \in \bar{S}$ and a given time-step input sequence $T \in \bar{T}(S)$, produces the set of all valid decision sequences $\bar{D}$.*
- *A cost function $c(S, T, \sigma)$ where $S \in \bar{S}, T \in \bar{T}(S)$ and σ is a sequence of decisions produced by an algorithm which we call a strategy and is defined below in definition 2.3. c then maps a strategy on some starting input and timestep input to some cost in $\mathbb{R}_{\geq 0}$.*

For most problems the validity of a single time-step input is independent of the time-step sequence preceding it. In that case for notational simplicity we write $\bar{T}(S)$ as the set of all possible single time-step inputs that can be given, with a valid time-step input sequence any combination of those time-step inputs.

For most problems the validity of a single decision is independent of the decisions that proceeded it. In these cases for notational simplicity we write $\bar{D}$ as the set of valid decisions, with the valid decision sequences any combination of those valid decisions.

Given an online problem, an instance of an online problem picks out specific values from the valid possible values defined by the online problem:

DEFINITION 2.2. *An instance (S, T) of an online problem $(\bar{S}, \bar{T}, \bar{D}, c)$ from 2.1 consists of:*

- *A starting input $S \in \bar{S}$ from the set of possible inputs $\bar{S}$*
- *A time-step sequence $T \in \bar{T}(S)$*

An algorithm for an online problem produces a strategy by repeatedly making decisions, while receiving new time-steps from an instance of the problem:

DEFINITION 2.3. *A strategy $\sigma = (\sigma_1, \sigma_2, ..., \sigma_m)$ for a given online problem instance is a sequence of decisions.*

A completed strategy has a decision for every time-step input, while a partial strategy contains all the decisions made 'so far' when the algorithm has only received some of the time-step input. Both completed and partial strategies must be mapped to some cost by c.

As it is an online problem, the algorithm makes decision σ_i only knowing the contents of $T_1, T_2, ..., T_i$ and being unaware of $T_{i+1}, T_{i+2}, ..., T_m$.

We let Σ_A refer to the set of all possible strategies for a given instance of the problem A.

We use 1-indexed python list-slicing notation to refer to portions of a strategy. In particular if we have the strategy $\sigma = (\sigma_1, \sigma_2, ..., \sigma_m)$ then:

$$\sigma_{[a:b]} = (\sigma_a, \sigma_{a+1}, ..., \sigma_{b-2}, \sigma_{b-1})$$

$$\sigma_{[a:]} = \sigma_{[a:|\sigma|+1]} = (\sigma_a, \sigma_{a+1}, ..., \sigma_{m-1}, \sigma_m)$$

$$\sigma_{[:b]} = \sigma_{[1:b]} = (\sigma_1, \sigma_2, ..., \sigma_{b-2}, \sigma_{b-1})$$

If this definition of an online problem is unclear, we recommend reading it in conjunction with section 3.1, which gives a simple definition of JRP in this framework. Definitions of an online problem in this framework are generally simple in practice - the overall online problem definition only becomes more complicated to be fully general and handle all the definitions we need.

2.3 Defining online reductions

2.3.1 Intuition of definition

Given an instance of the primal problem P, something that meets our definition of an online reduction to the reduced problem R should do the following:

- Map the primal starting input into a starting input for the reduced problem.
- For each time-step in the primal input, produce some number of reduced input time-steps. These time-steps can be functions of earlier time-step input, but not later time-step input as those haven't been revealed to us yet.
- A way to map a decision made in the reduced problem back to a decision in the primal problem.

An online reduction is thus made up of three functions: a function ρ_S to map the **s**tarting input, a function ρ_T to map the **t**ime-step input, and a function ρ_Σ to map each strategy σ. We give a diagram of these various functions that together make a reduction r in fig 2.1.

Only 'reasonable' decisions an algorithm could make need to be mapped. In particular, we assume an algorithm is trying to achieve a good competitive ratio.Note that defining what is 'reasonable' can depend on the nature of a particular problem, however great care should be taken when judging a decision to be unreasonable: reasonable strategies can be bad or sub-optimal, just not maliciously or pointlessly bad.

Examples of unreasonable decisions would include any decision that costs an infinite amount, or in the joint replenishment problem ordering an item when there is no request for that item, or in online service with delay moving to a point when that point, and all points along the way to that point, don't have any unfulfilled requests.

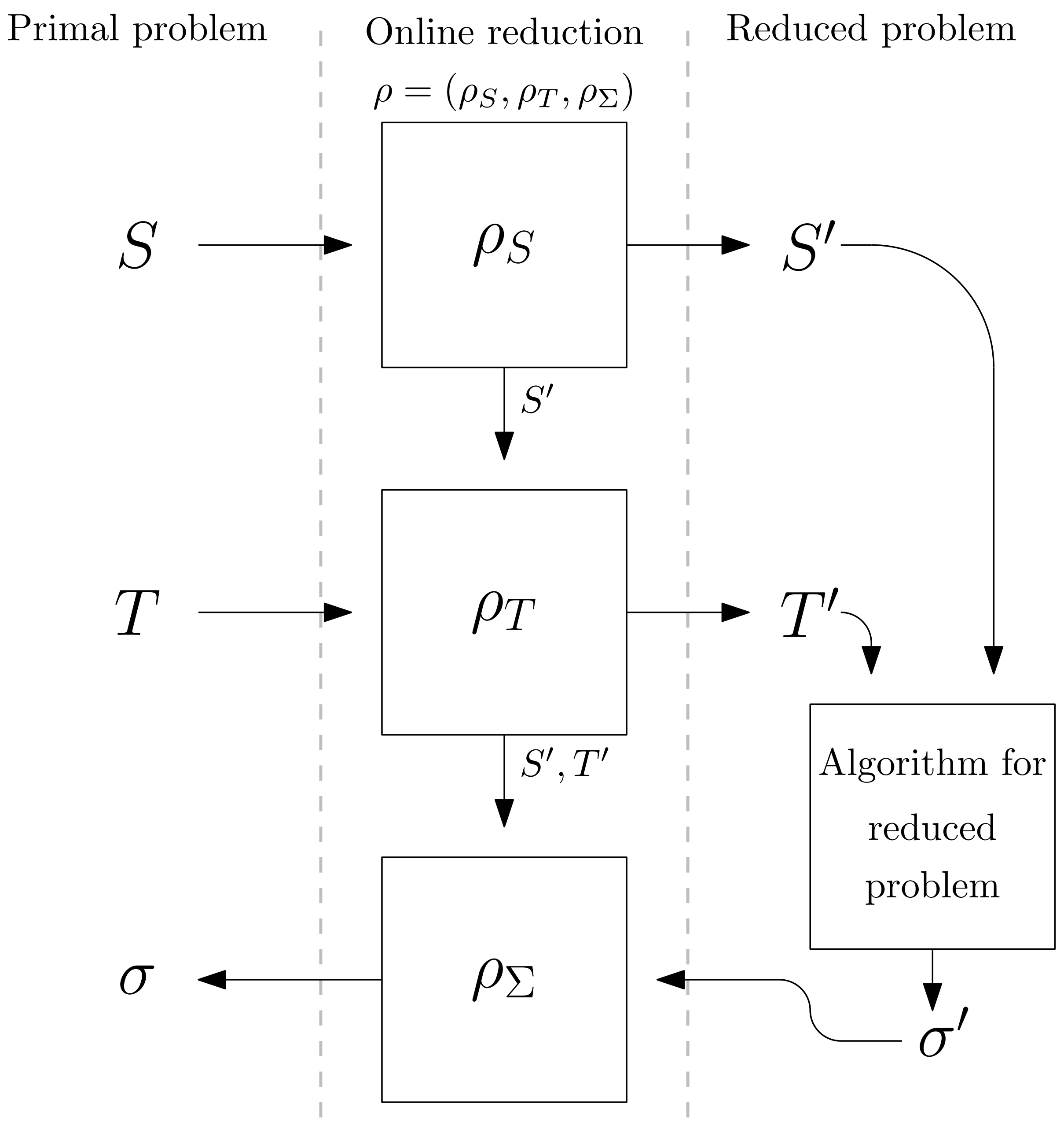


FIGURE 2.1. Diagram showing the various reduction functions $\rho_S, \rho_T, \rho_\Sigma$ which together make the reduction ρ, and how these are used to turn an online algorithm for the reduced problem into an online algorithm for the primal problem.
ρ_T is also given access to S and so can be assumed to know S', and ρ_Σ is also given access to S and T and so can be assumed to know S' and T'.

2.3.2 Formal definition

DEFINITION 2.4. *An online reduction $\rho = (\rho_S, \rho_T, \rho_\Sigma)$ from the primal problem $P = (\bar{S}, \bar{T}, \bar{D}, c)$ to the reduced problem $R = (\bar{S}', \bar{T}', \bar{D}', c')$ consists of:*

- *A function $\rho_S : \bar{S} \to \bar{S}'$ from the primal starting input to a starting input in the reduced problem*
- *A function ρ_T which looks like $\rho_T(S, (T_1, T_2, ..., T_m)) = (T_1', T_2', ..., T_m')$, and maps the time-step sequence input T to a valid time-step sequence T' for the starting input S'. T_i' is a function of $T_1', T_2', ..., T_i'$ only.*
- *A function ρ_Σ which looks like $\rho_\Sigma(S, (T_1, T_2, ..., T_k), \sigma') = \sigma$ and maps a reasonable strategy σ' in the reduced problem instance to a strategy σ in the primal problem instance. We require σ and σ' have length $k \leq m$.*

This definition assumes for every timestep T_τ in the primal problem our reduction creates exactly one timestep T_τ' in the reduced problem. We presented it like this for notational simplicity, however with some reductions, for every timestep T_τ we wish to create l timesteps in the reduced problem. In these cases we denote our reduced time-steps as $T_{\tau,\phi}'$, the ϕ-th timestep arriving after the first timestep $T_{\tau,1}'$ corresponding to timestep T_τ. This helps us keep track of how time relates between our primal and reduced problems: timesteps are still given to the algorithm one-by-one like normal, and the algorithm must still make a decision after every time-step.

Our function ρ_Σ responsible for mapping our strategies then only needs to worry about mapping strategies where the next timestep to be received corresponds to a new primal timestep - i.e. the next timestep to be received will be denoted $T_{\tau,1}'$. We also now require, for $\rho_\Sigma(S, (T_1, T_2, ..., T_k), \sigma') = \sigma$, that σ have length k and σ' have the same length as the number of timestep sequences produced by $\rho_T(S, (T_1, T_2, ..., T_k))$.

Where unambiguous we sometimes use ρ to refer to ρ_Σ, the function that maps strategies.

2.3.3 Why isn't ρ_T allowed to know the algorithms' decisions σ'?

In our online reduction definition, ρ_Σ is the only part of the reduction allowed to know what decisions the algorithm for the reduced problem is making: ρ_T is a function only of the starting input and the timestep input. When developing an online algorithm one often views the algorithm as being in a discussion or a game with some adversary: our algorithm receives a timestep input, makes a decision, and then the adversary gives some new timestep designed to trip up our algorithm. From this perspective, not extending the discussion between algorithm and adversary to a discussion between the reduction and the algorithm seems like a mistake. This is not the case - blinding ρ_T, the timestep portion of the reduction, to the reduced algorithms' decisions is a deliberate choice, and a necessary choice to allow for the preservation of the competitive ratio, which is needed to make our online reduction tool useful.

The competitive ratio α is defined to be between the algorithm's strategy ALG and the optimal strategy OPT on the same time-step sequence. Just as how reductions in complexity theory are designed to allow a polynomial time algorithm for one problem to induce a polynomial time algorithm in another problem for which a reduction between them exists, if you have an online reduction from the primal problem to the reduced problem we want an algorithm with a competitive ratio for the reduced problem to induce an algorithm with a related competitive ratio in the primal problem. Imagine we had an online reduction that produced different time-step sequences if the algorithm changes its earlier decisions. A k-competitive algorithm on our reduced problem will produce some strategy of cost less than or equal to $k \cdot OPT$, where OPT is the cost of the optimal strategy *on that same timestep sequence*. However there could exist some other strategy, that would cause the reduction to send different timestep sequences, achieving some lower cost than OPT. Thus, while the k-competitive ratio may achieve a cost less than or equal to k times the optimal strategy on the timestep input it was given, this may have no relation to the cost of the overall optimal strategy. Our competitive ratio will thus have lost all meaning and we are unable to induce an algorithm in the primal problem with a related competitive ratio. While we may be able to analyse the produced primal strategies directly, we are gaining nothing from our online reduction and so it no longer functions as a useful tool to relate problems.

This restriction - the time-step sequence is not a function of the algorithm's decisions - is an existing restriction in the literature. Sometimes when analysing the worst case input for an algorithm, the literature uses terminology that suggests we are modifying the input based off the algorithms actions in order to make the worst case input easier to describe - for example making statements like "we serve a request with costs A at all points except the algorithms location, where we serve cost B". This however is merely an abuse of terminology: what is actually happening is out of all possible input sequences, we are picking the worst-case one that happens to, at that relevant time-step, give some input which aligns in some way with how the algorithm has acted - and so is most easily described as if it were a function of the algorithms previous action - but is in fact just a particularly unlucky input for our algorithm.

2.4 Limited memory online reductions

In definition 2.4 we defined ρ_Σ as a function that maps an entire strategy: $\rho_\Sigma(\sigma') = \sigma$. With a k-limited memory online reduction, ρ_Σ must map the last decision of any two reduced strategies with identical last k decisions, to the same primal decision. Formally ρ is a k-limited memory online reduction if:

For all strategies σ', ω' in Σ_R

If $\sigma'_{[:-k]} = \omega'_{[:-k]}$

Then $\rho_\Sigma(\sigma')_{[:-1]} = \rho_\Sigma(\sigma')_{[:-1]}$

If k=1 than we call ρ a memoryless online reduction.

Nearly all reductions presented in this thesis are memoryless, except for one which is 2-limited. Theorem 4.1 is only shown for memoryless reductions. It is our view that online reductions that are not limited memory online reductions with a small k do not meaningfully relate two online problems, as such reductions remove the linkage between the algorithms choices in the reduced problem and the corresponding induced primal strategy. Furthermore, unlimited memory online reductions allow complexity from the primal problem to be hidden in the reduction, further removing their ability to meaningfully relate online problems.

When unambiguous we may use ρ_Σ, or ρ to refer to either mapping an entire strategy, or just a single decision - as in a memoryless reduction it makes sense to map a single decision.

2.5 Competitive ratio preserving online reductions

A competitive ratio preserving online reduction is one where the costs between the strategies in the reduced problem, and the strategies they induce in the primal problem, are related enough such that a competitive algorithm in the reduced problem induces a competitive algorithm in the primal problem.

DEFINITION 2.5. *Assume we have an online reduction ρ from problem P to problem R. ρ is a competitive ratio preserving online reduction if for any instance of P:*

$$\text{For every strategy in the reduced problem } \sigma',\ \theta c'(\sigma') \geq c(\rho(\sigma'))$$
$$\text{and } \gamma c'(OPT') \leq c(OPT)$$

where θ, γ are constants with regards to σ', OPT' is the optimal strategy in the reduced problem for the instance produced by the reduction, and OPT is the optimal strategy in the primal problem.

We let $\beta = \frac{\theta}{\gamma}$ and if $\beta > 1$ call the reduction a β-competitive online reduction.

THEOREM 2.6. *A competitive ratio preserving online reduction ρ from problem P to problem R, with an $alpha$-competitive algorithm for problem R, induces an $\alpha\beta$-competitive algorithm for problem P.*

PROOF. Given an instance (S, T) of P we let $S' = \rho_S(S), T' = (\rho_T)$ our reduced instance of R and consider $\sigma = \rho(\sigma')$ the strategy our α-competitive algorithm produced on S', T'.

We have the following:

$$\text{From our definition, } c(\sigma) \leq \theta c'(\sigma')$$
$$\text{and } c(OPT) \geq \gamma c'(OPT')$$
$$\text{Dividing the first inequality by the second gives } \frac{c(\sigma)}{c(OPT)} \leq \frac{\theta}{\gamma} \frac{c'(\sigma')}{c'(OPT')}$$
$$\text{By the definition of an } \alpha\text{-competitive algorithm, } \frac{c(\sigma)}{c(OPT)} \leq \frac{\theta}{\gamma}\alpha$$

and hence our reduction induces an $\alpha\beta$-competitive algorithm for problem P. □

2.6 Order preserving online reductions

We introduce the concept of an order preserving online reduction to capture a class of sensible assumptions about an online reduction.

In particular, an order preserving online reduction preserves the ordering between strategies: if one strategy is cheaper than another in the reduced problem, they should map to a strategy that is cheaper than the other in the original problem.

We also require that every strategy in the primal problem has some strategy in the reduced problem that maps to it.

These requirements are however slightly stronger than necessary and so we weaken them to:

- If multiple reduced strategies map to the same primal strategy, we only consider the cheapest reduced strategy that creates a unique primal strategy, and thus only care about the relative ordering of those cheapest strategies
- If multiple primal strategies have the same cost c, only one of these needs to have some strategy in the reduced problem that maps to it.

Formally:

DEFINITION 2.7. *Let $\rho = (\rho_S, \rho_T, \rho_\Sigma)$ be an online reduction from $P = (\bar{S}, \bar{T}, \bar{D}, c)$ to $R = (\bar{S}', \bar{T}', \bar{D}', c')$. ρ is an order preserving online reduction if*

$$\textit{For all primal strategies } \mu, \textit{ there exists a reduced strategy } \mu' \textit{ s.t. } c(\mu) = c(\rho_\Sigma(\mu')) \tag{2.1}$$

$$\textit{and}$$

$$\textit{For all pairs of reduced strategies } \bar{\sigma}', \bar{\omega}', \tag{2.2}$$

$$\textit{where } \sigma' \textit{is the cheapest reduced strategy that maps to the same primal strategy as } \bar{\sigma}'$$

$$\textit{and } \omega' \textit{is the cheapest reduced strategy that maps to the same primal strategy as } \bar{\omega}',$$

$$\textit{we have:}$$

$$\begin{aligned} c'(\sigma') < c'(\omega') &\quad\Longrightarrow\quad c(\rho_\Sigma(\sigma')) < c(\rho_\Sigma(\omega')) \\ c'(\sigma') = c'(\omega') &\quad\Longrightarrow\quad c(\rho_\Sigma(\sigma')) = c(\rho_\Sigma(\omega')) \\ c'(\sigma') > c'(\omega') &\quad\Longrightarrow\quad c(\rho_\Sigma(\sigma')) > c(\rho_\Sigma(\omega')) \end{aligned}$$

Formally we can specify σ', ω' as:

$$\sigma' = \min_{\sigma''}\{\sigma'' \in \Sigma_R \mid \rho(\sigma'') = \rho(\bar{\sigma'})\}$$

$$\omega' = \min_{\omega''}\{\omega'' \in \Sigma_R \mid \rho(\omega'') = \rho(\bar{\omega'})\}$$

2.7 Exactly cost preserving online reductions

An exactly cost preserving online reduction is one where every primal strategy has a reduced strategy that maps back to it, and every reduced strategy costs the same amount as the primal strategy it maps to:

DEFINITION 2.8. *A reduction from problem P to problem R is an exactly cost preserving online reduction if*

For every primal strategy σ there exists a reduced strategy σ' s.t. $\rho(\sigma') = \sigma$ (2.3)

and, for every reduced strategy ω', $c'(\omega') = c(\rho(\omega'))$ (2.4)

Note that an exactly cost preserving online reduction is trivially a competitive ratio preserving online reduction with $\beta = 1$ and is also an order preserving online reduction.

CHAPTER 3

Relating the Joint Replenishment Problem to Multi Level Aggregation with Delay

In this chapter we introduce the online joint replenishment problem (JRP) and the multi level aggregation with delay problem (MLAP). We then give an exactly cost preserving online reduction from JRP to MLAP.

This is done to give a gentle introduction to the use of our online reduction framework with a simple example, and as a sanity check to show our online reduction framework does in fact capture existing folklore reductions. Correspondingly it is not a technically challenging chapter, and the resulting theorem 3.4 we prove is already known, in an informal sense, in the folklore - this chapter is merely to ease into the chapter 2 online reduction framework.

The joint replenishment problem has direct applications to problems arising in logistics, and is designed to model manufacturing items on-demand with no inventory. Multi level aggregation with delay has similar applications to logistics and manufacturing, but can model more complicated relationships between items while still capturing the advantages of economies of scale.

3.1 Joint Replenishment Problem definition

In the Joint replenishment problem we are given a set of items $P = \{1, ..., n\}$, costs to order each item $d_1, d_2, ..., d_n$ and a joint ordering cost d_0.

Requests for items arrive over time, with request r for item p_r arriving at time t_r with a non-decreasing delay function $c_r : \mathbb{N} \to \mathbb{R}_{\geq 0}$. We view delay functions as mapping from $\mathbb{N}$ rather than $\mathbb{R}_{\geq 0}$ as we discretised time in chapter 2. If request r is fulfilled at time $t' \geq t_r$, we pay $c_r(t' - t_r)$ in delay costs for that request.

We fulfil requests by including the item that request is for in an order, served after the request has arrived. Serving an item in an order fulfils all requests for that item, but in that order we only pay for the item once, irrespective of how many requests are currently open for it.

To serve an order $\sigma \subseteq P$ we pay the order fulfilment cost $d_0 + \sum_{i \in \sigma} d_i$.

Note we have modified the notation from the literature, such as in Buchbinder et al. (2013), in order to use consistent notation between related problems in this paper. In particular we use P instead of $\mathcal{H}$ to refer to the items as these will generalise in other problems to points in a metric space, d_i instead of K_i as the item ordering costs generalise to distances in that metric space, and σ for orders which generalise into decisions in a strategy. We have also discretised time into arbitrarily small time-steps, for example viewing delay functions as mapping from $\mathbb{N}$ instead of $\mathbb{R}_{\geq 0}$. In the standard literature delay functions are continuous and so we can approximate any delay function arbitrarily well by arbitrarily many discrete timesteps.

Within the online problem framework of 2.1:

DEFINITION 3.1. *Joint Replenishment problem:*

- $\bar{S} = (P, d_0, d_1, d_2, ..., d_n)$ *where* $P = \{1, 2, ..., n\}$ *are the item types and the* $d_i \in \mathbb{R}_{\geq 0}$ *are the order costs.*
- $\bar{T}(S) = 2^{\{r \mid r \text{ a valid request}\}}$*, with a time-step a set of requests we receive in that time-step, where a valid request is* $r = (p_r, c_r)$ *with* $p_r \in P$*,* $c_r : \mathbb{N} \to \mathbb{R}_{\geq 0}$*. If request* r *arrived in time-step* τ *we use the notation* $t_r = \tau$ *to record request* r *arrived at time* τ*.*
- $\bar{D}(S, T) \subseteq P$*, a set of items to order.*
- $c(S, T, \sigma)$ *: with* $\sigma = (\sigma_1, \sigma_2, ..., \sigma_k), \sigma_\tau \in D$ *a strategy. We define* c *as the sum of the ordering costs and the delay costs:*

$$c = \sum_{\substack{\tau=1,...,k \\ \sigma_\tau \neq \varnothing}} \left(d_0 + \sum_{i \in \sigma_\tau} d_i \right) + \sum_{r \in T_1 \cup T_2 \cup ... \cup T_k} c_r(t'_r - t_r) \tag{3.1}$$

where t'_r *is the time we serve request* r*, or the delay cost incurred so far if we are yet to serve* r*:*

$$t'_r = \min(\{k\} \cup \{\tau \in \{1, ..., k\} \mid p_r \in \sigma_\tau, \tau \geq t_r\})$$

3.2 Multi Level Aggregation with Delay definition

Multi-level aggregation with delay (MLAP) generalises the online joint replenishment problem to allow for more general tree-like ordering cost dependencies, rather than the simple d_0 fixed joint ordering cost.

In MLAP we are given a rooted tree P with positive edge weight values d_e.

Requests arrive over time at leaves in the tree, with request r at node p_r arriving at time t_r with a non-decreasing delay function $c_r : \mathbb{N} \to \mathbb{R}_{\geq 0}$. If request r is fulfilled at time $t' \geq t_r$, we pay a delay cost of $c_r(t' - t_r)$ for that request.

We fulfil request r by including p_r in an order served at some time $t' \geq t_r$. In an order we serve a subtree $\sigma \subseteq P$ that includes the root: $p_{root} \in \sigma$. To serve order σ we pay the weight of all the edges in the subtree, as shown in fig 3.1.

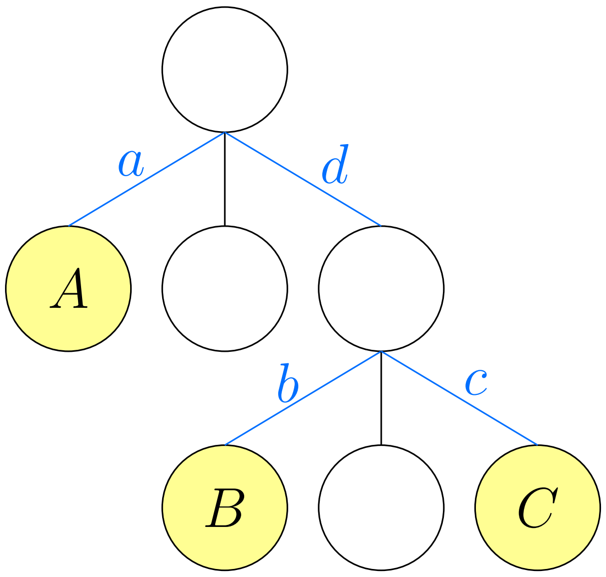


FIGURE 3.1. Diagram of an example MLAP instance, where we place an order for items $\{A, B, C\}$ and pay the cost of edges $\{a, b, c, d\}$, the edges in the subtree with just the items in our order.

Intuitively, the cost of serving a request p_r is the cost to connect p_r to the root, but we can achieve some savings through utilising economies of scale, by connecting multiple nodes to the root at the same time, and only paying for the common edges once.

Note again we modify the notation from literature in order to use consistent notation for related concepts between related problems in this paper. In particular we use P to refer to our tree, as this concept is similar to the JRP items P and generalises to points in a metric space, and we use d_e for our tree's edge weights as this again relates to the JRP item ordering costs and generalises to distances in a metric space.

Within the online problem definition of 2.1:

DEFINITION 3.2. *Joint Replenishment problem:*

- $\bar{S} = (P, d)$ *where P is a tree with n leaf nodes labelled* $1, 2, ..., n$ *and edge weights d_e. We let $d_{u,v}$ correspond to the sum of the edge weights in the path from u to v:* $\sum_{e \in path(u,v)} d_e$
- $\bar{T}(S) = 2^{\{r | r \text{ a valid request}\}}$*, where a valid request is a tuple (p_r, c_r) with p_r a leaf node of P and $c_r : \mathbb{N} \to \mathbb{R}_{\geq 0}$. If request r arrived in time-step τ we use the notation $t_r = \tau$ to record r arrived at time τ.*
- $\bar{D}(S, T) = \{\sigma \mid \sigma$ *a subtree of P containing the root*$\}$.
- $c(S, T, \sigma)$ *with $\sigma = (\sigma_1, \sigma_2, ..., \sigma_k)$ a strategy, is the sum of the ordering costs (the weight of every order's subtree) and the delay costs.*

$$c(S, T, \sigma) = \sum_{\tau=1}^{k} \left(\sum_{e \in \sigma_\tau \text{ an edge}} d_e \right) + \sum_{r \in T_1 \cup T_2 \cup ... \cup T_k} c_r(t'_r - t_r) \tag{3.2}$$

where t'_r is the time request r is fulfilled:

$$t'_r = \min(\{k\} \cup \{\tau' \in \{1, ..., k\} \mid p_r \in \sigma_{\tau'}, \tau' \geq t_r\})$$

3.3 Exactly cost preserving online reduction from JRP to MLAP

We now present an exactly cost preserving online reduction from JRP to MLAP, showing that JRP is in some sense a special case of MLAP. This is not intended to be a surprising result, but rather to give an easy example online reduction to help get us started.

3.3.1 Intuition

JRP problem instances can be viewed as a three layer tree, with items at the bottom connected to an extra node which is then connected to the root. By setting the edge weights of this tree we can make our

MLAP order costs on this tree align exactly with the corresponding JRP order costs. A diagram of this tree is shown in fig 3.2. This is a well known reduction within the folklore.

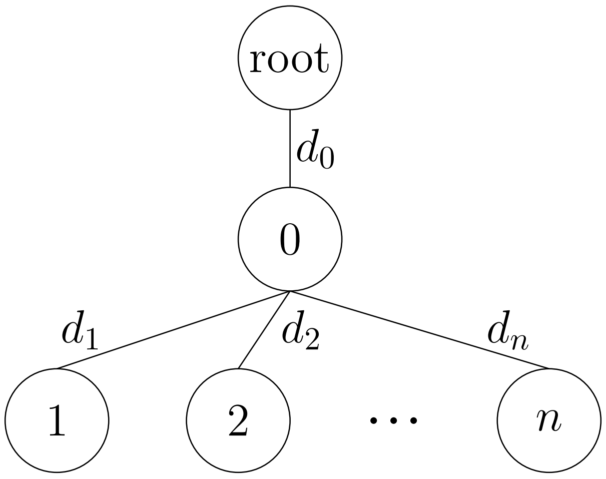


FIGURE 3.2. MLAP metric space produced by our reduction from a JRP instance.

Let our JRP instance have items $1, 2, ..., n$, item ordering costs $d_1, d_2, ..., d_n$ and fixed ordering costs d_0. Our reduction ρ creates an MLAP instance with the starting input $S = (P, D)$ shown in fig 3.2 a tree with leaf nodes labelled $1, 2, ..., n$ connected to a parent node labelled 0. We give the edge $(i, 0)$ weight $w_{(i,0)} = d_i$ the JRP item ordering cost for i. The parent node 0 is connected to the root node by an edge of weight $w_{(0,\text{root})} = d_0$ the JRP fixed ordering cost.

If we receive a request on item i in JRP, in the reduced MLAP instance we send a request on node i at the same time with the same delay cost. If in the MLAP instance an algorithm decides to serve an order σ_i on nodes $\{0, root, \sigma_{i_1}, \sigma_{i_2}, ..., \sigma_{i_l}\}$ we translate that to a JRP order on items $\{\sigma_{i_1}, \sigma_{i_2}, ..., \sigma_{i_l}\}$. Note nodes 0 and $root$ must appear in every non empty order.

Note the cost in MLAP for this order is the same as the cost in the JRP strategy it is translated to, and the requests fulfilled in MLAP and JRP also correspond with each other so that we always have equal delay costs .

3.3.2 Formal definition

REDUCTION 3.3. *Joint Replenishment Problem to Multi level aggregation with delay:*

$\rho_S(S)$: We are given the points $P = \{1, 2, ..., n\}$, fixed ordering cost d_0 and item ordering costs $d_1, d_2, ..., d_n$. We return the MLAP starting input:

$$S = (P', d') \text{ where}$$

$$P' = P \cup \{0, root\}$$

and $d'(i, j)$ is the distance function for paths in the tree metric shown in fig 3.2. Specifically, on a tree with point $root$ the root with a single child 0 with edge weight d_0, and leaf nodes $\{1, 2, ...n\}$ all children of 0 with edge weights from 0 to i of d_i. The distance $d'(i, j)$ between two nodes i and j is the length of the unique simple path in the tree from i to j.

$\rho_T(S, T)$: Given a series of time-steps $T = (T_1, T_2, ..., T_m)$ we give the same sequence to our facility instance: $T' = (T_1, T_2, ..., T_m)$. In particular, if each timestep T_i is a set of requests, we send those same requests to our MLAP instance. No relabelling is necessary as the labels of our nodes correspond to the label of the items.

$\rho_\Sigma(\sigma')$: Given a strategy $\sigma' = (\sigma'_1, \sigma'_2, ..., \sigma'_k)$ we translate it to the JRP strategy that, for each σ'_i in MLAP that makes an order, makes an order in JRP on the corresponding items. This involves removing the added $root$ and 0 points from the decision, but nothing more

Hence,

$$\sigma_i = \rho_\Sigma(\sigma'_i) = \begin{cases} \varnothing & \text{if } \sigma'_i = \varnothing \\ \sigma'_i \setminus \{root, 0\} & \text{if } \sigma'_i \neq \varnothing \end{cases}$$

We return the strategy $\sigma = (\sigma_1, \sigma_2, ..., \sigma_k)$.

3.3.3 Analysis

THEOREM 3.4. *Reduction 3.3 is an exactly cost preserving online reduction from joint replenishment problem to multi level aggregation with delay.*

PROOF. We proceed by showing lemmas 3.5 and 3.6 which together directly prove our theorem.

We assume we are given some JRP instance.

LEMMA 3.5. *Every strategy σ for the JRP instance has a corresponding strategy σ' in the reduced MLAP instance such that $\rho(\sigma') = \sigma$.*

PROOF. Let our JRP strategy be denoted by $\sigma = (\sigma_1, \sigma_2, ..., \sigma_k)$.

Consider the MLAP strategy which, whenever our JRP strategy serves an order on items $\{p_1, p_2, ..., p_i\}$, serves a corresponding order on the subtree $\{root, 0, p_1, p_2, ..., p_i\}$:

$$\sigma'_i = \begin{cases} \varnothing & \text{if } \sigma_i = \varnothing \\ \sigma_i \cup \{root, 0\} & \text{if } \sigma_i \neq \varnothing \end{cases}$$

Note as every non empty order includes $root$ and 0 - the only non leaf nodes in our tree - and as the JRP order is non empty we must have at least one leaf node present - every MLAP order is a valid subtree and hence our constructed σ' a valid MLAP strategy

By the definition of ρ_Σ it is clear $\rho_\Sigma(\sigma'_i) = \sigma_i$ as required. □

LEMMA 3.6. *Every strategy σ' in the reduced MLAP instance has equal cost to its corresponding JRP strategy: i.e. $c'(\sigma') = c(\rho(\sigma'))$*

PROOF. Consider some strategy $\sigma' = (\sigma'_1, \sigma'_2, ..., \sigma'_k) \in \Sigma_{MLAP}$.

We first note that, as this strategy is a strategy for our reduction, we can easily characterise what every decision must look like:

OBSERVATION 3.7. *Each σ'_τ is of the form $\varnothing$ or $\{root, 0\} \cup P_\tau$ where $P_\tau \subseteq \{1, 2, ..., n\}$, by the nature of our reduction.*

The cost of σ' is thus (from equation 3.2):

$$c'(\sigma') = \sum_{\tau=1}^{k} \left(\sum_{e \in \sigma'_\tau \text{ edges}} d_e \right) + \sum_{r \in T_1 \cup T_2 \cup ... \cup T_k} c_r(\min(\{k\} \cup \{\tau \in \{1, ..., k\} \mid p_r \in \sigma'_\tau, \tau' \geq t_r\}) - t_r)$$

which, by the structure of the tree and observation 3.7 can be simplified to:

$$c'(\sigma') = \sum_{\substack{\tau=1,\ldots,k \\ \sigma'_\tau \neq \varnothing}} \left(d_0 + \sum_{i \in P_\tau} d_i \right) + \sum_{r \in T_1 \cup T_2 \cup \ldots \cup T_k} c_r(\min(\{k\} \cup \{\tau \in \{1,...,k\} \mid p_r \in P_\tau, \tau' \geq t_r\}) - t_r)$$

Let $\sigma := \rho(\sigma')$ be our corresponding JRP strategy.

The cost of σ is thus (from equation 3.1, observation 3.7, and the definition of ρ_Σ):

$$c(\sigma) = \sum_{\substack{\tau=1,\ldots,k \\ \sigma_\tau \neq \varnothing}} \left(d_0 + \sum_{i \in P_\tau} d_i \right) + \sum_{r \in T_1 \cup T_2 \cup \ldots \cup T_k} c_r(\min(\{k\} \cup \{\tau' \in \{1,...,k\} \mid p_r \in P_\tau, \tau' \geq t_r\}) - t_r)$$

Hence $c'(\sigma') = c(\rho(\sigma))$ as required. □

Lemmas 6.4 and 6.5 together meet conditions 2.3 and 2.4 for an exactly cost preserving online reduction and thus theorem 3.4 is proven as required. □

CHAPTER 4

Relating the Joint Replenishment Problem to Metrical Task System

We now define the metrical task system problem, a very flexible problem for which it is easy to find reductions for problems to, which also has a known tight competitive ratio for deterministic algorithms of $2n - 1$, where n is the size of the MTS metric space. For the definition of JRP please see section 3.1.

To relate JRP and MTS we give the most significant result of this paper, theorem 4.1, which states it is impossible to have an order preserving memoryless online reduction from the joint replenishment problem to the metrical task system without creating a huge number of states and thus experiencing a bad competitive ratio.

4.1 Metrical Task System Definition

In Metrical Task System we are again given a metric space P with n points and a metrical distance function between the points $d : P \times P \to \mathbb{R}_{\geq 0}$. We also have a server which is located at a point $p \in P$, and can move around the metric space to another point $q \in P$ by paying $d(p, q)$.

At each time-step we receive a cost vector $T \in \mathbb{R}^n$. Each entry of the cost vector corresponds to a point $p \in P$ and specifies how much we have to pay to serve that cost vector with our server at point p. After receiving that cost vector but before paying it, we can choose to move our server to any point $q \in P$ - paying the distance cost as normal - and we then pay the cost vector from where we moved to.

We consider a version of metrical task system where new points can be added to the metric space at any time. We call this modifiable MTS and it is considered within folklore to be the same as traditional MTS. The known optimal algorithm for MTS, the work function algorithm, which was introduced and analysed in Chrobak and Larmore (1998), works with essentially no modification on modifiable MTS and achieves the same competitive ratio of $2n - 1$, where n is the final number of points in the metric space. This can be seen by comparing the work function algorithm on a modifiable MTS instance with the equivalent traditional MTS, where all points exist for all time-steps but have cost vector entries set to ∞ before the point has been ‘added’. The work function algorithm’s chosen strategy , and the optimal strategy, are the same between our modifiable and traditional MTS instances, and so achieve the same competitive ratio.

We use P_τ and d_τ to refer to the metric space and the metrical distance function respectively as it is in time-step τ. We let $n_\tau = |P_\tau|$ be the size of the metric space at time-step τ. We let P and n be the metric space and its size respectively once all points have been revealed - i.e. $P = P_m, n = n_m$.

In the terms of the online problem definition of 2.1:

- $\bar{S} = \{S \mid S \text{ a valid starting input}\}$. A valid starting input is $S = (P_0, d_0, p_0)$ where $P_0 = \{1, 2, ..., n_0\}, n_0 \in \mathbb{N}$ holds the points initially in our metric space, $d_0 : P_0 \times P_0 \to \mathbb{R}_{\geq 0}$ is a metrical distance function on those points and $p_0 \in P_0$ is the starting location of our server.
- $\bar{T}(S)$: a single time-step T_i includes r_i, the cost vector for that time-step, P_i the metric space at that time-step and d_i the metrical distance function for the metric space at that time-step. To specify valid time-step sequences we add constraints to make sure each time-step correctly modifies the metric space - i.e. it only adds points, and the distances for existing points remains unmodified. Our $\bar{T}(S)$ is thus the set of all possible time-step sequences:

$$\begin{aligned}\bar{T}(S) = \{(T_1, T_2, ...T_m) \mid \quad & T_i = (P_i, d_i, r_i) \text{ where} \\ & P_{i-1} \subseteq P_i, \\ & d_i \text{ metrical}, \\ & d_i(p,q) = d_{i-1}(p,q)\ \forall p, q \in P_{i-1} \\ & r_i \in \mathbb{R}^{n_i}_{\geq 0} \qquad\qquad \forall i = 1, ..., m\ \forall m \in \mathbb{N}\}\end{aligned}$$

- $\bar{D}(S, T) = \{(\sigma_1, \sigma_2, ..., \sigma_j) \mid \sigma_i \in P_i\}$ The sequence of points we wish to serve each request in, with request r_i served from point p_i. Note here we are specifying the valid strategies, not the valid individual decisions, as the validity of a decision depends on which time-step it is made in.
- $c(S, T, \sigma)$: with $\sigma \in \bar{D}(S, T)$ a strategy, we define the cost as the sum of the movement costs and the amount we paid to serve each cost vector:

$$c = \sum_{i=1}^{m} (d_i(\sigma_{i-1}, \sigma_i) + r_i(\sigma_i) \tag{4.1}$$

with $\sigma_0 := p_0$ for notational simplicity and $r_i(\sigma_i)$ the entrance in the i-th cost vector for point σ_i.

4.2 All order preserving memoryless online reductions from JRP to MTS create superpolynomial many MTS states

4.2.1 Intuition

MTS, as a problem, only has one part of it that has some kind of state: which point the server is at. This state can only remember one of n options at a time, and the problem cannot remember where the server has been - once the server moves to a state, MTS charges it the same amount from then on as a server that was always at that state, or arrived in some different way.

JRP, however, remembers what requests are fulfilled or unfulfilled. Thus JRP can distinguish between strategies with any differing combination of unfulfilled requests - even if those strategies have no longer differed for arbitrarily long, as long as they still have differing fulfilled/unfulfilled requests, JRP can remember and charge them different costs. If we send a request on each point, and look at the strategies that fulfil half of them, there are $n\ choose\ \frac{n}{2}$ combinations of requests left unfulfilled - a superpolynomial number. If time between our JRP and reduced MTS instances are related, once the JRP strategies start acting identically MTS will struggle to differentiate them, but JRP still can.

Thus if we look at our reduced MTS instance, if it has a polynomial (with regards to n) number of states, by the pigeonhole principle, two MTS strategies that induce two different 'half-fulfilling' JRP strategies, will eventually occupy the same MTS state. If we're careful to have our costs and time sync up between JRP and MTS, we can make a strategy that mimics one of those strategies during the time that determines what JRP strategy each should map to, but mimics the other when MTS has to charge us for the differing costs of the JRP strategies. As MTS has no way to remember what costs to charge beyond what state a strategy is currently in, MTS will have no way to distinguish our mimic strategy and our original one, and will have to charge them the same amount, even though the JRP strategies are charged different amounts. Doing this carefully allows us to have the strategies not just break costs but to also break order-preservation.

A helpful diagram of these strategies is given in fig 4.1.

This also explains why we need a memoryless reduction: without this, the reduction can remember for MTS and distinguish between the strategies even though MTS can't. With further still broad assumptions about how reductions with memory can map strategies between the primal and reduced problem, it should be possible to modify this proof to apply to unlimited memory - but restricted strategy reductions - as well - however if this is actually possible, and what restrictions would be needed, remains an open question. It is also clear that should a remembering reduction no longer has the algorithm for MTS actually 'solve' the problem. Our algorithm for MTS is trying to decide between various actions to make the best choice, however even if it takes the same action and gets charged the same amount in two different strategies, as long as the strategies once differed, the reduction might send them to completely different JRP strategies. How then, is the MTS algorithm actually doing the work to solve the JRP instance and achieve a cheap strategy, when a cheap cost in MTS can cause wildly different unrelated costs in JRP to be charged?

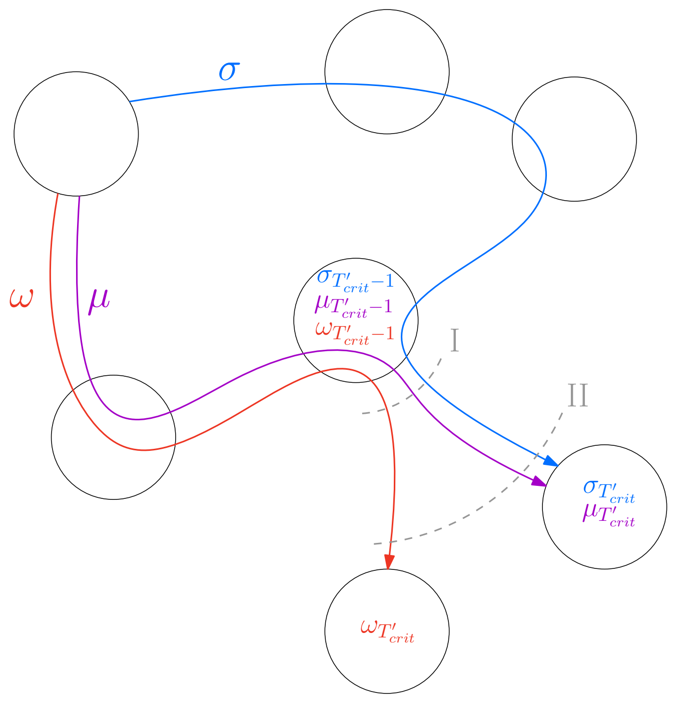


FIGURE 4.1. Diagram of the paths our strategies take through the MTS metric space, showing intuition behind proof.
All our strategies have equal cost at time $T-1$, before crossing I, but have already fixed what JRP order they correspond to. As ω, μ travelled together they correspond to the same JRP strategy, while ω corresponds to some other strategy of some different cost.
At time T, after crossing II, the delay functions in JRP charge their costs, and so our strategies in MTS need to correspondingly cost different amounts. μ needs to cost the same as ω, as they correspond to the same strategy, but μ followed ω and so MTS will charge them the same amount.

4.2.2 Proof

THEOREM 4.1. *Any order preserving memoryless online reductions from JRP to MTS must have a superpolynomial number of MTS states with regards to the number of JRP items n.*

PROOF. Assume, for the sake of contradiction, that there exists ρ an order preserving memoryless online reduction from JRP to MTS that creates polynomially many MTS states with regards to the number of JRP items n.

Consider the JRP instance where we have a setup of n items, with joint ordering cost d_0 and all item order costs $d_i = 1$. In the first time-step we receive one request on every point, with the request on point $i \in P$ having a delay function initially equal to 0 till time $t = T_{crit}$ when it jumps to 2^i.

In our problem notation from 2.2 our JRP instance is:

- $S = (P, d_0, d_1, ..., d_n)$ with $P = \{1, ..., n\}$, $d_0 = d_1 = d_2 = ... = d_n = 1$
- $T = (T_1, \varnothing, \varnothing, ..., \varnothing)$ with $T_1 = \{(i, c_{r_i}) \mid \forall i = 1, ..., n\}$,

$$c_{r_i} = \begin{cases} 0 & t < T_{crit} \\ 2^i & t \geq T_{crit} \end{cases}$$

We consider the set of strategies Λ that immediately serve an order containing exactly half the items, and then do nothing:

$$\Lambda = \left\{(\sigma_1, \varnothing, \varnothing, ..., \varnothing) \mid \sigma_1 \subseteq P, |\sigma_1| = \frac{n}{2}\right\} \tag{4.2}$$

OBSERVATION 4.2. *Every strategy in Λ costs the same before time T_{crit}:*

$$c(\sigma_{[:\tau]}) = \frac{n}{2} + 1 \quad \forall \tau < T_{crit}$$

This is because every strategy pays the order cost for $\frac{n}{2}$ items, and $d_i = 1$ for every item i.

OBSERVATION 4.3. *Every strategy in Λ after $t = T_{crit}$ pays a different cost:*

$$c(\sigma_{[:\tau]}) \neq c(\omega_{[:\tau]}) \quad \forall \tau \geq T_{crit}, \ \forall\, \sigma, \omega \in \Lambda$$

This is because every strategy pays equal order cost, and different delay costs. As the delay cost of each unserved request is 2^i, we can write each strategies' delay cost as a unique binary number, with a 1 in the i-th digit if the request on point i is unserved. As each strategy gets a unique binary number, each strategy pays a different cost.

OBSERVATION 4.4. *The number of strategies in Λ is superpolynomial with regards to n:*

$$|\Lambda| = \binom{n}{\frac{n}{2}} = \frac{n!}{(\frac{n}{2}!)^2} \sim \frac{4^n}{\sqrt{\pi n}}$$

Which follows from the number of ways we can choose half the items to include in the single order, with the asymptotic relation to $O(4^n)$ a well known result (OEIS Foundation Inc., 2022).

From these observations we show theorem 4.1 by showing any online problem where you can construct such an instance - with k strategies all of equal cost before some time T_{crit}, and then all costs differing - needs at least k MTS states in any order preserving memoryless online reduction to MTS.

LEMMA 4.5. *Any order-preserving online reduction ρ from some primal problem P to MTS, where there exists an instance of P with*

- *a set of valid strategies Λ with equal cost before some time-step T_{crit},*
- *an equal decision in all strategies at time-step T_{crit}: $\sigma_{T_{crit}} = \omega_{T_{crit}}\ \forall \sigma, \omega \in \Lambda$, and*
- *different costs in all strategies at T*

must have at least $|\Lambda|$ MTS states.

PROOF. Let our instance of the primal problem be denoted by $P = (S, T)$.

We use ρ to get our corresponding MTS instance:
let $S' = \rho_S(S), T' = \rho_T(T)$ and Σ' the set of all possible MTS strategies for the instance (S', T').

Let Λ' be the set of cheapest strategies that map back to Λ:

$$\Lambda' = \{\lambda' \in \Sigma' \mid \rho(\lambda') \in \Lambda, \quad \lambda' = \min_{\lambda'' \in \Sigma', \rho(\lambda'')=\rho(\lambda')} c'(\lambda'')\}$$

Note as every strategy in Λ has a distinct cost, and we pick the cheapest MTS strategy that maps to an MTS strategy, condition 2.1 implies $|\Lambda'| = |\Lambda|$.

Assume, for the sake of contradiction, that we have fewer than $|\Lambda|$ MTS states: i.e. $n' = |S'| < |\Lambda|$

Let T'_{crit} be the timestep produced by ρ_T that corresponds to T_{crit}. We don't make any assumptions about how many reduced timesteps ρ_T produces for each primal timestep, but continue to number our timesteps $T'_1, T'_2, ...T'_{m'}$ as in this case that is the simplest notation.

By the pigeon hole principle there exist two distinct strategies $\omega', \sigma' \in \Lambda'$ which, in the time-step immediately before T'_{crit}, occupy the same MTS state: $\omega'_{T'_{crit}-1} = \sigma'_{T'_{crit}-1}$.

Assume without loss of generality that σ' is the cheaper strategy: $c'(\omega') \geq c'(\sigma')$. As all our JRP strategies' costs are distinct, and condition 2.2 applies to our JRP strategies, $c'(\omega') > c'(\sigma')$.

We now construct the strategy μ' which mimics ω' for every time-step except for those corresponding to T'_{crit}, when it mimics σ':

$$\mu' = (\omega'_1, \omega'_2, ..., \omega'_{T'_{crit}-1}, \sigma'_{T'_{crit}})$$

An illustrative diagram showing the path these strategies take through the MTS instance's metric space, and the intuition of this proof, is shown in fig 4.1.

Firstly note μ' and ω' are identical till the last timestep: $\mu'_{[:T'_{crit}]} = \omega'_{[:T'_{crit}]}$ and so $\rho(\mu'_{[:T'_{crit}]}) = \rho(\omega'_{[:T'_{crit}]})$.

Secondly note all our strategies in Λ take the same action in the final time-step: $\rho(\sigma'_{T'_{crit}}) = \rho(\omega'_{T'_{crit}})$ and so as ρ is memoryless:

$$\begin{aligned}\rho(\mu') &= (\rho(\omega'_1), \rho(\omega'_2), ..., \rho(\omega'_{T-1}), \rho(\sigma'_T)) \\ &= (\rho(\omega'_1), \rho(\omega'_2), ..., \rho(\omega'_{T-1}), \rho(\omega'_T)) \\ &= \rho(\omega') = \omega\end{aligned}$$

Thus our constructed strategy μ' maps to the same OSD strategy as ω'. As ρ is order-preserving, condition 2.1 $c'(\mu') \geq c'(\omega')$ as ω' was in Λ' and is thus the cheapest way to produce $\rho(\omega'$.

We now analyse directly the costs our MTS strategies must be charged, per the definition of the MTS cost function.

$$c'(\mu') = c'(\mu')$$

$$c'(\mu') = c'(\mu'_{[:T'_{crit}]}) + c'(\mu'_{T'_{crit}}) \qquad \text{as MTS costs are added linearly}$$

$$c'(\mu') = c'(\omega'_{[:T'_{crit}]}) + c'(\mu'_{T'_{crit}}) \qquad \text{as } \mu'_{[:T'_{crit}]} = \omega'_{[:T'_{crit}]}$$

$$c'(\mu') = c'(\sigma'_{[:T'_{crit}]}) + c'(\mu'_{T'_{crit}})$$

as all strategies in Λ' pay the same amount before $t = T_{crit}$

$$c'(\mu') = c'(\sigma'_{[:T'_{crit}]}) + d(\mu'_{T'_{crit}-1}, \sigma'_{T'_{crit}}) + r_{T'_{crit}}(\sigma'_{T'_{crit}}) \text{ by definition of MTS cost function}$$

$$c'(\mu') = c'(\sigma'_{[:T'_{crit}]}) + d(\omega'_{T'_{crit}-1}, \sigma'_{T'_{crit}}) + r_{T'_{crit}}(\sigma'_{T'_{crit}}) \qquad \text{as } \mu'_{T'_{crit}-1} = \omega'_{T'_{crit}-1}$$

$$c'(\mu') = c'(\sigma'_{[:T'_{crit}]}) + d(\sigma'_{T'_{crit}-1}, \sigma'_{T'_{crit}}) + r_{T'_{crit}}(\sigma'_{T'_{crit}})$$

as $\omega'_{T'_{crit}-1} = \sigma'_{T'_{crit}-1}$ per original pigeon hole principle

$$c'(\mu') = c'(\sigma'_{[:T'_{crit}]}) + c'(\sigma'_{T'_{crit}}) \qquad \text{by definition MTS cost function}$$

$$c'(\mu') = c'(\sigma')$$

a contradiction, as $c'(\mu') \geq c'(\omega') > c'(\sigma')$. Hence we must have $n = |S'| \geq |\Lambda|$ proving lemma 4.5 as required. □

As the set of strategies in Λ (defined in 4.2) satisfies the requirements for lemma 4.5, we have proven theorem 4.1 as required. □

As we can chain reductions together, theorem 4.1 also applies to any problem we can reduce JRP to. In particular if we have have a reduction from JRP to some problem R, where R has a metric space whose size is polynomial with regards to the number of JRP items, and we also have a reduction from R to MTS that creates a polynomial number of MTS states with regards to the size of R's metric space, we can reduce JRP to a MTS instance with polynomially many states by first reducing our JRP instance to R, and then reducing R to MTS. We thus get the following:

COROLLARY 4.6. *If there exists an order preserving memoryless online reduction ρ from JRP to a problem R, where R has a metric space whose size grows polynomially with regards to the number of JRP items, no order preserving memoryless online reduction from R to MTS, that has a polynomial number of MTS states with regards to the size of R's metric space, exists.*

As we have reductions that meet these requirements from JRP to multi-level aggregation with delay, online facility location, online service with delay and group service with delay - as shown in the fig 1.1 reduction hierarchy, corollary 4.6 applies to all of those problems and none of them can have an order preserving memoryless online reduction with polynomially many states to metrical task system.

CHAPTER 5

Relating Online Facility Location to Joint Replenishment Problem

In this chapter we introduce another online problem on a metric space, the online facility location problem, and show like with MLAP, the joint replenishment problem can also be reduced to online facility location. While not as technically challenging as chapters 4, 6 or 7, unlike chapter 3 this reduction is novel and not previously known in the literature.

Online facility location has an obvious application to the logistics problem it directly models, but can be applied to other situations such as renting cloud computing facilities.

5.1 Online Facility Location with Delay definition

5.1.1 Intuition

There are many variations of facility location, including many variations of online facility location with delay. We investigate a generalisation of the version introduced in Azar and Touitou (2019), where you can only open facilities at a set of 'facility points', and you can only receive requests at a set of 'customer points', both subsets of the same metric space. While this is sufficient to allow for a reduction from JRP to online facility location, we also allow each facility location to have a distinct 'facility renting' price in order to show that even the most general version of online facility location with delay can still be captured by a reduction to 'group OSD' (a new problem introduced in section 7.2).

In the Azar and Touitou (2019) version of the problem received requests don't need to be fulfilled immediately but instead incur delay costs while they wait to be served. Requests are served by 'renting' a facility and connecting requests to it. After serving an order, the facility is lost and we must pay for the facility again if we wish to use it in another order.

Other variations include connecting incoming requests to the nearest built facility as they arrive, with the algorithm choosing where to build permanent facilities, seen in Anagnostopoulos et al. (2004) with facilities built for uniform costs at potential facility locations and in Fotakis (2008) with varying facility construction costs but potential facility locations and customer locations drawn from the same set.

These restrictions - uniform facility costs or restricted facility construction locations - are generally made to allow for more competitive algorithms or easier analysis, and so it is interesting that in the Azar and Touitou (2019) ‘renting’ variation adding these restrictions seems unlikely to simplify the reduction to group OSD or allow a reduction to OSD. Further - while not formally proven it seems unlikely a reduction from JRP to a version of online facility location with delay that includes both restrictions is possible, which suggests the inclusion of both those restrictions is actually a significant simplification of the problem.

The formal relation of all these variations to each other - renting facilities instead of buying, uniform facility costs, and restricted facility construction locations - showing which can and can’t be reduced to each other, including which variations are equivalent - is an open problem the resolution of which might reveal surprisingly large differences between the various variations.

5.1.2 Our actual problem definition

We are given a metric space with points M and a metrical distance function between the points given by d. From this metric space we are given a set of customer demand points $P \subseteq M$ and potential facility locations $F \subseteq M$.

We can assume without loss of generality that $P \cap F = \varnothing$, by splitting each point in $P \cap F$ into a pair of points ϵ apart with one point in P and one in F.

We label our customer demand points in P as $1, 2, ..., n$. For each point $v \in F$ we are given a facility construction cost $f_v \in \mathbb{R}_{\geq 0}$.

Requests arrive over time at the customer demand points, with requests of the usual form: request r arrives at point $p_r \in P$ at time t_r with a non-decreasing delay function $c_r : \mathbb{N} \rightarrow \mathbb{R}_{\geq 0}$. If request r is fulfilled at time $t' \geq t_r$ we pay a delay cost of $c_r(t' - t_r)$ for that request.

In each time-step an algorithm may choose to rent a set of facilities $v_1, v_2, ..., v_l$, paying the rental cost f_{v_i} for each facility rented. The algorithm can then connect points $p_i \in P$ to a rented facility v_i, paying the distance cost between them $d(p_i, v_i)$ and fulfilling any requests on the customer point p_i.

In our unified problem notation from 2.1 we have:

DEFINITION 5.1. *Online facility location with delay:*

- *$\bar{S}$ is the set of all valid S, where a valid S looks like:*
  $S = (P, F, d, f)$ where $P = \{1, 2, ..., n\}$ is the set of customer demand points, F is the set of potential facility locations $\{v_1, v_2, ..., v_{|l|}\}$, $d : P \sqcup F \to \mathbb{R}_{\geq 0}$ is a metrical distance function between all our points and $f : F \to \mathbb{R}_{\geq 0}$ is our facility construction costs. Note $F \cap P = \varnothing$.
- *$\bar{T}(S) = 2^{\{r | r \text{ a valid request}\}}$, where a valid request is a tuple (p_r, c_r) with $p_r \in P$, c_r a delay function. If request r arrives at time τ, we let $t_r = \tau$.*
- *$\bar{D}(S, T) = \{\{(v_1, L_1), (v_2, L_2), ..., (v_k, L_k)\} \mid v_i \in F,\ v_i \neq v_j \forall i \neq j,\ L_i \subseteq P\}$ with each decision a set containing some number of locations to build facilities at, and the points to connect to each facility.*
- *$c(S, T, \sigma)$ with $\sigma = (\sigma_1, \sigma_2, ..., \sigma_k), \sigma_\tau \in D$ a strategy. We define c as the sum of all the facility construction costs, the customer connection costs and the delay costs:*

$$c(S, T, \sigma) = \sum_{\tau=1,...,k} \left(\sum_{(v,L)\in\sigma_\tau} \left[f_v + \sum_{p\in L} d(v, p) \right] \right) + \sum_{r\in T_1\cup T_2\cup...\cup T_k} c_r\left(t'_r - t_r\right) \tag{5.1}$$

  where $L_{\sigma_\tau} =$ is the set of all customer points served in time-step τ in strategy σ and t'_r is the time we serve request r:

$$t'_r = \min(\{k\} \cup \{\tau' \in \{1, ..., k\} \mid p_r \in L_{\sigma_\tau}, \tau' \geq t_r\})$$

5.2 Reducing Joint Replenishment Problem to Online Facility Location

5.2.1 The reduction

5.2.1.1 Intuition

We are given a JRP instance, with items $1, 2, ..., n$, fixed ordering cost d_0 and item ordering costs $d_1, d_2, ..., d_n$.

We reduce this to an online facility location problem with customer demand points $P = \{1, 2, ..., n\}$ and facilities $F = \{0\}$, a single potential facility point . The cost to rent the facility is the fixed ordering cost d_0 and customer demand point i is located a distance d_i from 0. A diagram of this setup is shown in fig 5.1.

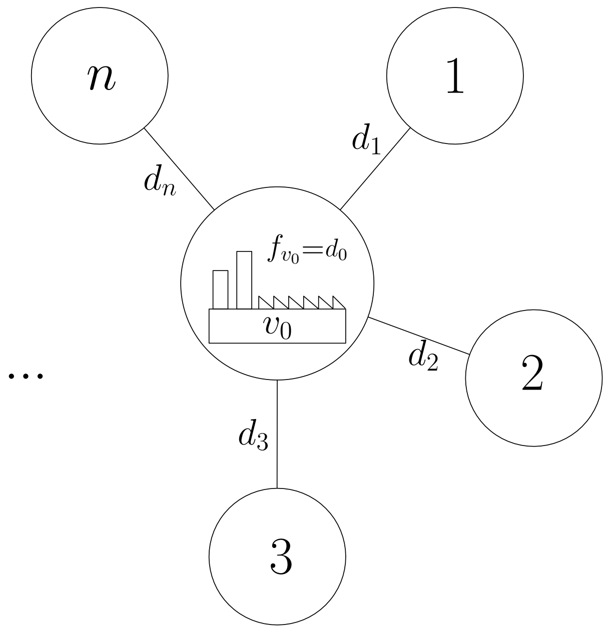


FIGURE 5.1. Diagram of the facility starting input made by our reduction from a JRP instance.

Whenever we receive a request in JRP at a point i, we send a request in facility location at point i and with an equal cost function c_r. If the algorithm in facility decides to rent the factory at 0 and connect it to customers $p_1, p_2, ..., p_i$, we translate that facility decision back to the JRP decision to serve an order with the items $p_1, p_2, ..., p_i$.

Note we have an exact correspondence between the JRP instance and our facility location instance with the regards to the cost to serve each order, the delay costs paid, and the requests that have been fulfilled and not fulfilled so far.

5.2.1.2 Formal definition

REDUCTION 5.2. *Joint Replenishment Problem to Online Facility Location with Delay:*

$\rho_S(S)$*: We are given the points* $P = \{1, 2, ..., n\}$*, fixed ordering cost* d_0 *and item ordering costs* d_i*. We return an online facility location starting input:*

$$S' = (P', F', d', f') \textit{ where}$$
$$P' = P$$
$$F' = \{0\}$$
$$f' = d_0$$

and with $d'(i, j)$ *the distance function for a star metric with our single facility* 0 *at the centre, and a length from the facility at point* 0 *to point* i *of* d_i*.*

$\rho_T(S, T)$*: Given a series of time-steps* $T = (T_1, T_2, ..., T_m)$ *we give the same sequence to our facility instance:* $T' = (T_1, T_2, ..., T_m)$*. Note as we kept the labelling of our points the same, no further translation is necessary.*

$\rho_\Sigma(\sigma')$*: Given a strategy* $\sigma' = (\sigma'_1, \sigma'_2, ..., \sigma'_k)$ *we translate it to a corresponding JRP strategy, with only translation work done to unpack each decision to be in the expected format of a JRP decision - in particular, returning just the customers we connected to* 0*. Note as there is only one facility location point* 0 *in our instance, we have at most one order in each decision. Hence,*

$$\sigma_i = \begin{cases} \varnothing & \textit{if } \sigma'_i = \varnothing \\ L_{\sigma'_i} & \textit{if } \sigma'_i \neq \varnothing \end{cases}$$

recalling $L_{\sigma'_i}$ *is the set of all points connected to a facility in strategy* σ' *at time-step* i*). We return the strategy* $\sigma_i = (\sigma_1, \sigma_2, ..., \sigma_k)$*.*

5.2.2 Analysis

THEOREM 5.3. *Reduction 5.2 is an exactly cost preserving online reduction from joint replenishment problem to online facility location with delay.*

PROOF. We proceed by showing lemmas 5.4 and 5.5 which together directly prove our theorem.

LEMMA 5.4. *Every JRP strategy σ has a corresponding strategy in the reduced facility instance σ' such that $\rho(\sigma') = \sigma$.*

PROOF. Let our JRP strategy be denoted by $\sigma = (\sigma_1, \sigma_2, ..., \sigma_k)$.

Consider the facility strategy which whenever JRP serves an order, builds a facility at 0 and connects it to the points for the items in that order:

$$\sigma'_i = \begin{cases} \varnothing & \text{if } \sigma_i = \varnothing \\ (0, \sigma_i) & \text{if } \sigma_i \neq \varnothing \end{cases}$$

By the definition of ρ_Σ it is clear $\rho_\Sigma(\sigma'_i) = \sigma_i$ as required. □

LEMMA 5.5. *Every strategy in the reduced facility instance σ' has equal cost to its corresponding JRP strategy: $c'(\sigma') = c(\rho(\sigma'))$*

PROOF. Consider some strategy $\sigma' = (\sigma'_1, \sigma'_2, ..., \sigma'_k)$.

We first calculate the cost of this facility strategy.

Note as there is only a single possible facility location, σ'_τ is either $\varnothing$ or of the form $(0, L_\tau)$.

The cost of σ' is thus (from definition 5.1):

$$c'(\sigma') = \sum_{\substack{\tau=1,...,k \\ \sigma'_\tau \neq \varnothing}} \left(d_0 + \sum_{i \in L_\tau} d_i \right) + \sum_{r \in T_1 \cup T_2 \cup ... \cup T_k} c_r(\min(\{k\} \cup \{\tau' \in \{1, ..., k\} \mid p_r \in L'_\tau, \tau' \geq t_r\}) - t_r)$$

We now calculate the cost of the corresponding JRP strategy for σ'.

Let $\sigma := \rho(\sigma') = (\sigma_1, \sigma_2, ..., \sigma_k)$. From definition 3.1 the cost of σ, $c(\sigma)$, is:

$$c(\sigma) = \sum_{\substack{\tau=1,...,k \\ \sigma_\tau \neq \varnothing}} \left(d_0 + \sum_{i \in \sigma_\tau} d_i \right) + \sum_{r \in T_1 \cup T_2 \cup ... \cup T_k} c_r(\min(\{k\} \cup \{\tau' \in \{1, ..., k\} \mid p_r \in L_\tau, \tau' \geq t_r\}) - t_r)$$

As we have either $\sigma_\tau = \varnothing = \sigma'_\tau$ or $\sigma_\tau = L_\tau$, our expressions are equal and hence $c'(\sigma') = c(\rho(\sigma))$ as required. □

Lemmas 5.4 and 5.5 together meet conditions 2.3 and 2.4 for an exactly cost preserving online reduction and thus theorem 3.4 is proven as required. □

CHAPTER 6

Relating Multi-Level aggregation with Delay to Online Service with Delay

In this chapter we define online service with delay (OSD), and then show multi-level aggregation with delay (MLAP) can be reduced to OSD. While it was previously known in the folklore that one can think of OSD as a generalisation of MLAP, this was only in an informal and vague sense; we however with this reduction show that OSD is not just a vague intuitive generalisation but they can in fact be formally related, and an $O(f(n))$-competitive algorithm for OSD induces an $O(f(n))$-competitive algorithm for MLAP, where n is the number of points in MLAP's metric space.

The online service with delay problem is a very general problem that can be used to model many other situations. One direct application involves modelling the order to read locations in a hard drive disk, with the metric space n uniform points, each representing a cylinder, with a request to read a file sent to the corresponding point it's located at, and the delay function representing how urgently that file needs to be read.

6.1 Online Service with Delay definition

In the online service with delay problem, we are given a metric space P with n points and a metrical distance function $d : P \times P \to \mathbb{R}_{\geq 0}$ between those points. We have a server which sits at some point $p \in P$, and can move to some other point $q \in P$ at any time by paying $d(p, q)$. We can visit as many points as we like in a single time-step.

Requests arrive over time, with request r arriving at point p_r at time t_r with a non-decreasing delay function $c_r : \mathbb{N} \to \mathbb{R}_{\geq 0}$. We fulfil request r by moving the server to point p_r at some time $t' \geq t_r$, paying delay cost $c_r(t' - t_r)$. In till we fulfil a request we are continuously charged the delay cost incurred 'so far'.

Within the online problem definition of 2.1 we have:

DEFINITION 6.1. *Online service with delay:*

- $\bar{S} = \{(P, d, p') \mid P = \{1, 2, ..., n\}, n \in \mathbb{N}\}$ *where* d *is a metrical distance function on* P *-* $d : P \times P \to \mathbb{R}_{\geq 0}$ *and* $p' \in P$ *is the point the server starts at* $\}$

- $\bar{T}(S) = 2^{\{r | r \textit{ a valid request}\}}$ *where a valid request is of the form* $r = (p_r, c_r)$ *with* $p_r \in P$, $c_r : \mathbb{N}_{\geq 0} \to \mathbb{R}$ *a non-decreasing delay function.*

- $\bar{D}(S,T) = \{(p_1, p_2, ..., p_j) \mid j \in \mathbb{N}_{\geq 1}, p_i \in P\}$ *is a series of points in* P *to visit. Note we must always visit at least one point, the point we want to be at when that time-step ends.*

- $c(S,T,\sigma)$ *: with* $\sigma = (\sigma_1, \sigma_2, ..., \sigma_m)$ *a strategy and with the path taken in a single decision in this strategy notated* $\sigma_i = (\sigma_{i_1}, \sigma_{i_2}, ..., \sigma_{i_j}) \in \bar{D}(S,T)$. *We define* c *as the sum of the movement costs (using 6.2) and the delay costs:*

$$c(S,T,\sigma) = \sum_{\tau=1}^{k} d(\sigma_\tau, \sigma_{\tau+1}) + \sum_{\substack{r \in T_1 \cup T_2 \cup ... \cup T_k \\ r \neq \emptyset}} c_r(t'_r - t_r) \tag{6.1}$$

where t'_r *is the time request* r *is served:* $t'_r = \min(\{k\} \cup \{\tau' \in \{1, ..., k\} | p_r \in \sigma'_\tau, \tau' \geq t_r\})$ *and we let* $d(\sigma_{\tau-1}, \sigma_\tau)$*, the distance between two decisions, be the distance travelled to move from where the server ended at in time-step* $\tau - 1$*, through all the points we are visiting in* σ_τ*, to end at the final point from* σ_τ*. Formally this is given by:*

$$d(\sigma_{\tau-1}, \sigma_\tau) = d(\sigma_{(\tau-1)_{[-1]}}, \sigma_{\tau_1}) + \sum_{i=2}^{|\sigma_\tau|} d(\sigma_{\tau_{(i-1)}}, \sigma_{\tau_i}) \tag{6.2}$$

and recall that $\sigma_{(\tau-1)_{[-1]}}$ *refers to the last entry in the* $\sigma_{(\tau-1)}$ *path.*

Note it would be unreasonable to visit the points in a way that doesn't minimise the movement costs, and so we can assume the points $(p_1, p_2, ..., p_j)$ visited within a single decision are in the order of the shortest path from where the server starts that timestep at, through the points we visit, to end at p_j. Thus in every reasonable strategy we assume the points visited within a timestep are visited in this way.

It is also unreasonable for the server to visit a point u where u does not have an unfulfilled request, and is a stop on the shortest path the server is taking to some node with an unfulfilled request. This is because visiting u, in that situation, requires us to pay a movement cost that is guaranteed to not allow us to achieve a lower cost in the future, and we can simply delay moving to u till one of the above conditions are met. Doing this delay means we will have more information - for example we might be able to fulfil some request that hasn't arrived yet, on our visit to u.

It is also unreasonable for the server to not visit a point u when at some nearby point v, but to then in a subsequent timestep, where the charged delay costs haven't changed, to then visit u from a further away point w. In particular, as we assume our algorithm is trying to achieve a good competitive ratio, it is unreasonable to - when we earlier judged it not worth paying the movement cost to visit u when at v - to then later judge it worthwhile to pay the higher movement cost to visit u from w - when nothing else about our situation has changed. This also includes the special case of not visiting a point from some node u, and then later from u when no costs have been charged or information has changed, visiting that point.

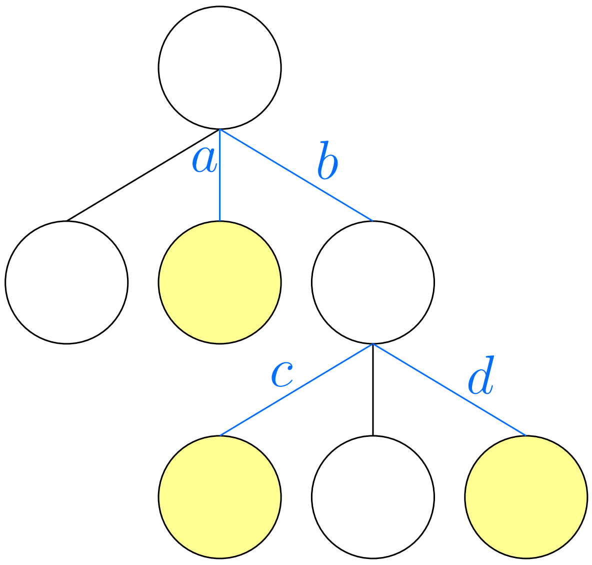
6.2 Reducing Multi Level Aggregation with Delay to Online Service with Delay

6.2.1 Intuition

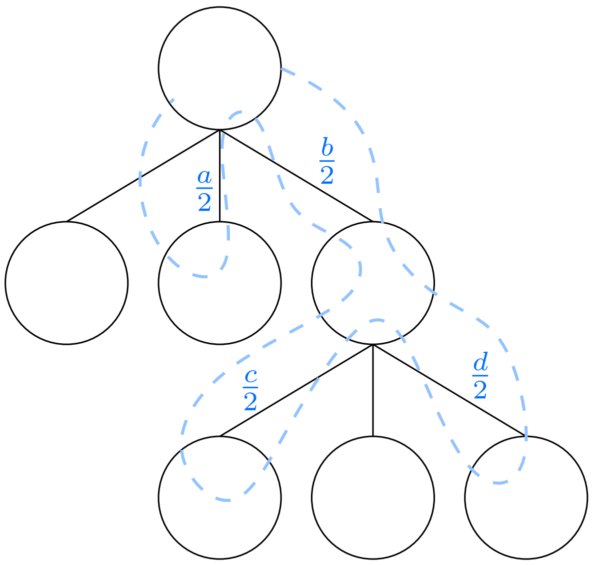
Multi level aggregation with delay features a tree metric space - which can be easily used as the metric space in Online service with delay - and requests that arrive over time, which can also easily be used without modification in a reduced OSD instance.

Thus the main challenge involves restricting the OSD server movement to act like MLAP ordering. We do this by, for every time-step in the MLAP instance, sending several time-steps in the reduced OSD instance. First we send a time-step during which the server is free to move to the various points in the tree, serving requests. Then we send a series of n immediately deadlining request at the root, forcing the OSD server to move back to the root and also end at the root. Having done this, our OSD server now moves by performing a partial Eulerian tour of the tree metric, fulfilling requests along this tour. This tour in OSD corresponds to an order of the visited leaf nodes in MLAP, fulfilling the same requests for the same movement costs while paying equal delay costs. A diagram showing this partial Eulerian tour is given in fig 6.1.

We consider a partial Eulerian tour to be an Eulerian tour of a subset of nodes of the tree, where those nodes also form a tree - for example, that shown in fig 6.1b. This is not to be confused with an Eulerian tour of a single child of the root, and all of that child's descendants.



(A) An example MLAP instance. An order serving the highlighted leaf nodes - as would be created by our reduction from the partial Eulerian tour shown in fig 6.1b - would pay for edges a, b, c, d at cost $a + b + c + d$.



(B) Reduced OSD instance created from MLAP instance in fig 6.1a. The server performs a partial Eulerian tour visiting leaf nodes $3, 5$ and 8, crossing each edge in the tour twice and so paying cost $2\cdot\frac{a}{2}+2\cdot\frac{b}{2}+2\cdot\frac{c}{2}+2\cdot\frac{d}{2} = a + b + c + d$.

FIGURE 6.1. Diagram of example metric spaces for a reduction from MLAP to OSD, with corresponding strategies drawn

6.2.2 Formal definition of reduction

REDUCTION 6.2. *Multi level aggregation with delay to Online service with delay.*

$\rho_S(S)$: *We are given the tree with points P, nodes (including non-leaf nodes) labelled $1, 2, .., n$ and edge weights d_e. We return an online service with delay instance starting input of the same tree, viewed* *as a metric space. Thus:*

$$
\begin{aligned}
S' = (P', d', root) \textit{ where} \\
& P' = P, \\
& d' = \textit{ the distance function for a tree metric space, with each edge from } d \\
& \quad \textit{having half the weight as in } P \\
& root = \textit{ the root of } P
\end{aligned}
$$

$\rho_T(S, T)$*: Given a series of time-steps* $T = (T_1, T_2, ..., T_m)$ *we send the time-step sequence, with the MLAP time-step* T_τ *getting the corresponding* τ*-sequence of time-steps* $T'_{\tau,0}, T'_{\tau,1}, ..., T'_{\tau,n}$*, with time-step* $T_{\tau,\phi}$*'s value specified later:*

$$T' = (T'_{1,0}, T'_{1,1}, T'_{1,2}, ..., T'_{1,2n},\ T'_{2,0}, T'_{2,1}, T'_{2,2}, ..., T'_{2,2n},\ ...,\ T'_{m,0}, T'_{m,1}, T'_{m,2}, ..., T'_{m,2n})$$

The time-step $T'_{\tau,\phi}$ *is as specified below, with the first time-step sending all the time-steps we received in the MLAP instance, and then* $2n$ *time-steps with a request deadlining at the root:*

$$
\begin{aligned}
T'_{\tau,0} &= \{(p_r, c'_r) \mid r \in T_i \textit{ a request}\} \\
\forall \phi \geq 1, T'_{\tau,\phi} &= \{(root, c'_{\tau,\phi deadline})\}
\end{aligned}
$$

with $c'_r(\tau, \phi) = c_r(\tau)$ *the same delay function, but stretched so that the time-step* $T'_{\tau,\phi}$ *has delay costs corresponding to those charged in* T_τ *and* $c'_{\tau,\phi\ deadline}$ *is the delay function deadlining immediately:*

$$
c'_{T,\Phi\ deadline} = \begin{cases} 0 & \textit{if } \tau \leq T \textit{ and } \phi \leq \Phi \\ \infty & \textit{if } \tau > T \textit{ or } \phi > \Phi \end{cases}
$$

$\rho_\Sigma(\sigma')$*: Given a strategy* $\sigma' = (\sigma'_{1,0}, \sigma'_{1,1}, ..., \sigma'_{1,2n}, \sigma'_{2,0}, \sigma'_{2,1}, ..., \sigma'_{m,2n})$ *we translate it to the MLAP strategy that in time-step* τ *submits an order for every point the server visited in the time-steps*
$(\tau, 0), (\tau, 1), (\tau, 2), ..., (\tau, 2n)$.

Thus:

$$\rho_\Sigma(\sigma'_{\tau,0}, \sigma'_{\tau,1}, \sigma'_{\tau,2}, ..., \sigma'_{\tau,2n}) = \bigcup_{\phi=0}^{2n} \sigma'_{\tau,\phi}$$

6.2.3 Analysis

THEOREM 6.3. *Reduction 6.2 is an exactly cost preserving online reduction from multi level aggregation with delay to online service with delay*

PROOF. We proceed by showing lemmas 6.4 and 6.5 which together directly prove our theorem.

LEMMA 6.4. *Every MLAP strategy σ has a corresponding strategy σ' in the reduced OSD instance (S', T') such that $\rho(\sigma') = \sigma$.*

PROOF. Let our MLAP strategy be denoted by $\sigma = (\sigma_1, \sigma_2, ..., \sigma_k)$.

Consider the OSD strategy which for $\sigma'_{(\tau,0)}$ performs a partial Eulerian tour of the subtree ordered in σ_τ and then spends all other $\sigma_{(\tau,\phi)}$ (i.e. all $\phi \geq 1$) sitting at the root. In particular:

$$\sigma'_{(\tau,0)} = \begin{cases} root & \text{if } \sigma_\tau = \varnothing \\ \text{partial Eulerian tour of } \sigma_\tau \text{ subtree} & \text{if } \sigma_\tau \neq \varnothing \end{cases}$$

$$\sigma'_{(\tau,\phi)} = root \quad \forall \phi \geq 1$$

By the definition of ρ_Σ it is clear $\rho_\Sigma(\sigma'_i) = \sigma_i$ as required. □

LEMMA 6.5. *Every reasonable strategy in the reduced facility instance σ' has equal cost to its corresponding MLAP strategy: $c'(\sigma') = c(\rho(\sigma'))$*

PROOF. Consider some reasonable OSD strategy for an OSD instance made by our reduction:

$$\sigma' = (\sigma'_{1,0},\ \sigma'_{1,1},\ ..., \sigma'_{1,2n},\ \sigma'_{2,0},\ \sigma'_{2,1},\ ..., \sigma'_{2,2n},\ ...,\ \sigma'_{m,2n})$$

We first show in order to be a reasonable strategy, during a τ-sequence $(\sigma'_{(\tau,0)}, \sigma'_{(\tau,1)}, \sigma'_{(\tau,2)}, ..., \sigma'_{(\tau,2n)})$, the server must move in a partial Eulerian tour ending at the root. Secondly we show the cost of performing this tour is equal to the order cost of σ_τ in MLAP, and that our delay costs are the same.

Note that as our OSD instance has the server start at the root, if in every sequence we end at the root, a trivial induction implies we can always assume we start at the root.

Consider the sequence $(\sigma'_{(\tau,0)}, \sigma'_{(\tau,1)}, \sigma'_{(\tau,2)}, ..., \sigma'_{(\tau,2n)})$ starting at the root.

Consider the steps $\sigma'_{(\tau,\phi)}$ and $\sigma'_{(\tau,\psi)}$, and without loss of generality assume $\phi < \psi$.
Assume - for the sake of contradiction - that these steps both involve visiting the same child of $root$, with a return to the root between those visits. Specifically, there exists leaf nodes $p, q \in P$ such that $p \in \sigma'_{\tau,\phi}$ and $q \in \sigma'_{\tau,\psi}$, where the lowest common ancestor a of p and q is not the root, and the server visits the root after visiting p in $\sigma'_{\tau,\phi}$ but before visiting q in $\sigma'_{\tau,\psi}$.

Note as in MLAP all requests must arrive at the leaf nodes of the tree, we can assume without loss of generality that p and q are both visited to fulfil requests on p and q: if p or q is a leaf node without a request then we are visiting a state - and so incurring a movement cost - for no reason, which is unreasonable and if p and q are not a leaf node they must either have some child leaf node with an open request which we can instead choose as p or q respectively, or we are again visiting a state and incurring a movement cost for no reason which is unreasonable.

As $\sigma'_{\tau,\phi}$ and $\sigma'_{\tau,\psi}$ both take place during the τ sequence, our reduction will have only sent the deadlining requests at the root between them. Thus, q is not visited in as if q was visited in $\sigma'_{\tau,\phi}$ then the request on q would have been fulfilled, a contradiction with when we earlier found q has an unfulfilled request.

Thus the algorithm that produced this strategy decided, when it passed through the lowest common ancestor a to visit p, not to fulfil q. However later when at the $root$, but in the same τ-sequence with no new information revealed or charged costs, the algorithm then decided to move to q. This strategy thus pays the movement cost $d(root, q)$ to fulfil q when it could have paid $d(a, q) < d(root, q)$. Any sensible algorithm that earlier judged it too costly to be worth fulfilling q, should consider the higher cost to fulfil it later also too costly. Further note that this argument can be applied simultaneously to every point in every revisit to that same subtree, such that once the server leaves a subtree it never revisits it.

Thus we can safely assume that in a τ-sequence in a reasonable strategy, we move from the root to a direct child of the root at most once. Thus the server, within a τ-sequence, must perform a partial Eulerian tour - except this tour can be cut short by ending at some leaf node instead of the root.

We now show the server must finish by returning to the root.

As a reasonable strategy can only leave the root in order to serve requests in some subtree, but there are at most n direct children of the root, and there are $2n + 1$ time-steps in the τ sequence for which the server must visit the root, on the $T_{\tau,2n}$ time-step the server follows the following cases:

*c*ase 1: The server has visited every direct child of the root. The server must fulfil the deadlining request at the root we receive in time-step T_{2n}, and so must visit the root in that time-step. Having visited the root, as we have already visited every direct child of the root in that τ-sequence, we cannot leave the root. Thus the server ends at the root.

*c*ase 2:The server had some time-step in which it did not visit a subtree, after which it must stay at the root for every subsequent time-step or else it is making different decisions at different time-steps when it is in the same state for both time-steps (as no new requests will arrive during the τ sequence, and the delay costs all stay frozen) - an unreasonable action.

Thus during the τ sequence $\sigma'_{\tau,0}, \sigma'_{\tau,1}, \sigma'_{\tau,2}, ..., \sigma'_{\tau,2n}$ the server must perform exactly a partial Eulerian tour of P.

Finally, note that this partial Eulerian tour has movement costs equal to the order costs of the corresponding MLAP decision: as every leaf node visited corresponds to an item in the corresponding MLAP order, and the partial Eulerian tour traverses each edge in the order subtree twice, and each edge has cost half that of its weight in MLAP, we pay the same amount per edge for the subtree. Hence our movement costs in OSD are equal to our order costs in the corresponding MLAP strategy.

Furthermore, our requests are fulfilled in both our OSD instance and the MLAP instance at corresponding times (accounting for the $(\tau', \phi) \to \tau$ stretching), with the delay functions taking on equal value, and so our OSD strategy and the corresponding MLAP strategy pay equal delay costs.

Thus as both our cost components are equal,

$$c'(\sigma') = c(\rho(\sigma))$$

as required. □

Lemmas 6.4 and 6.5 together meet conditions 2.3 and 2.4 for an exactly cost preserving online reduction and thus theorem 6.3 is proven as required. □

CHAPTER 7

Introducing Group Online Service with Delay and relating it to Online Facility Location

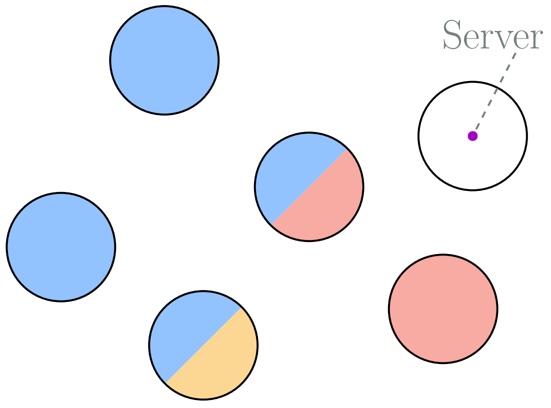
In this chapter we introduce group online service with delay (gOSD), a generalisation of OSD where a single request can be fulfilled by the server visiting any of several points specified by the request. We then give a reduction from online facility location to group online service with delay, proving theorem 7.3 and showing all of our problems - aside from metrical task system - can be exactly cost-preservingly online reduced to gOSD without asymptotically growing the metric space.

Finally we give an order-preserving online reduction from group online service with delay to metrical task system. Per theorem 4.1 this reduction must, and does, create superpolynomially many MTS states with regards to the number of points in the gOSD metric space. This completes our fig 1.1 reduction hierarchy.

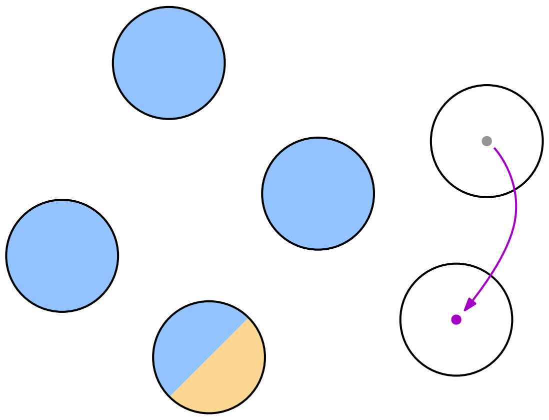
Group online service with delay is, similarly to online service with delay, a very general problem, with an example application including the same hard drive example as for OSD, but reflecting the common technique in some modern video games to place multiple copies of an important file in several physical locations in the HDD, so one file is always located 'near' the disk head.

7.1 Group Online Service with Delay definition

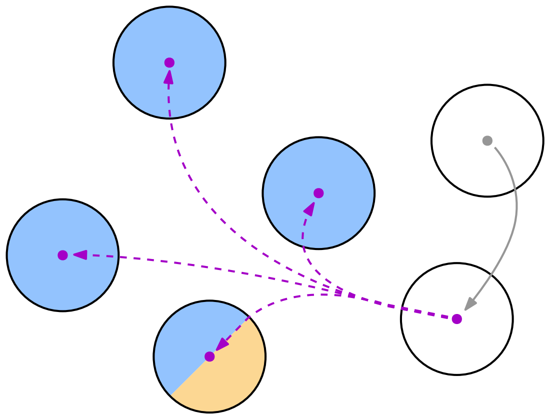
In Group online service with delay (gOSD), instead of arriving at a single point, requests arrive on a non-empty set of points. A request is fulfilled when the server visits any one of the points the request is on and upon fulfilment we pay the delay cost for the time we fulfilled it, and then no further delay costs are charged. An example gOSD instance, with an example strategy demonstrating how request fulfilment works, is shown in fig 7.1. All other details are the same as online service with delay (see section 6.1).



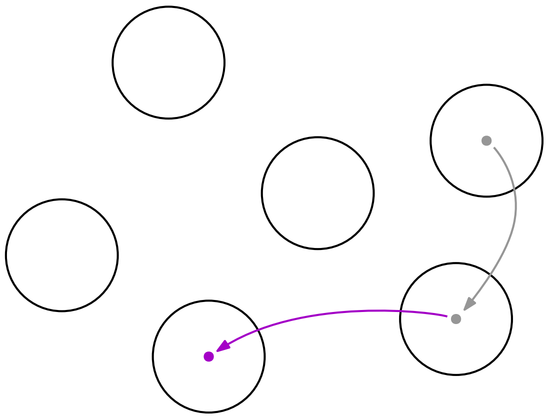
(A) Our server starts at the indicated point with the red, blue and yellow requests currently unfulfilled and on the nodes with their respective colours.



(B) Our node moves to the indicated point - paying the required movement cost as always - fulfilling the red request.



(C) Our server could fulfil the blue request by moving to any of the indicated nodes.



(D) Our server moves to the indicated node, fulfilling the blue and yellow requests.

FIGURE 7.1. Diagram of an example gOSD instance with three requests (red, blue and yellow), and an example strategy, with the strategies' effect on request fulfilment shown.

DEFINITION 7.1. *Group online service with delay:*

- $\bar{S} = \bar{S}_{OSD} = \{(P, d, p') \mid P = \{1, 2, ..., n\}, n \in \mathbb{N}\}$ *with* $d : P \times P \to \mathbb{R}_{\geq 0}$ *a metrical distance function and* $p' \in P$ *the point the server starts at.*
- $\bar{T}(S) = 2^{\{r \mid r \text{ a valid request}\}}$ *where a valid request is* $r = (P_r, c_r)$ *with* $P_r \subseteq P, P_r \neq \varnothing$, $c_r : \mathbb{N} \to \mathbb{R}_{\geq 0}$ *a non-decreasing function.*
- $\bar{D}(S, T) = \bar{D}_{OSD}(S, T) = \{(p_1, p_2, ..., p_j) \mid j \in \mathbb{N}_{\geq 1}, p_i \in P\}$*, a series of points in* P *to visit.*

- $c(S,T,\sigma)$ *: with* $\sigma = (\sigma_1, \sigma_2, ..., \sigma_m), \sigma_i = (\sigma_{i_1}, \sigma_{i_2}, ..., \sigma_{i_j}) \in \bar{D}(S,T)$ *a strategy. We define* c *as the sum of the movement costs (using 6.2) and the delay costs:*

$$c(S,T,\sigma) = \sum_{\tau=1}^{k} d(\sigma_\tau, \sigma_{\tau+1}) + \sum_{\substack{r \in T_1 \cup T_2 \cup ... \cup T_k \\ r \neq \varnothing}} c_r(t'_r - t_r) \tag{7.1}$$

where t'_r *is the time request* r *is served, which now occurs when any point the request is for is visited:* $t'_r = \min(\{k\} \cup \{\tau \in \{1,...,k\} | p_r \cap \sigma_\tau \neq \varnothing, \tau \geq t_r\})$

Our requirements for a reasonable gOSD strategy are the same as those for a reasonable OSD strategy.

7.1.1 Motivating group online service with delay

Group online service with delay - as indicated with the choice of name and the similarities in the problem definition - is a generalisation of online service with delay. That it is, in fact, a generalisation is clear with the existence of a trivial reduction from OSD to group OSD: a metric space and request sequence for OSD can be sent directly as the input to group OSD, with the group OSD decisions valid decisions for the OSD instance, and an exact correspondence between the OSD and gOSD requests, and the delay and movement costs paid.

The ‘group’ generalisation of a problem is a common generalisation of problems. The main motivation behind introducing group online service with delay is for its use in section 7.2, when we reduce online facility location to group online service with delay. In online service with delay, a request can only be fulfilled in a single way (by moving the server to it). However, with online facility location, a request can be fulfilled in several different ways, with the different choices of which potential facility to connect to a point.

Attempts to reduce online facility location to online service with delay generally failed because of this inability to capture multiple ways to fulfil a request, and there likely exist other problems such as caching and k-Server for which this is the case, although further investigation into these problems, and a proof it is impossible to reduce facility location to OSD, is needed. This extra ability of group online service with delay makes it an interesting extension of OSD, and the fact this is a significant new capability from an online reduction perspective - with high likelihood of allowing gOSD to capture facility location, while OSD can’t - hints that an algorithm for gOSD might need to solve new challenges compared to OSD.

7.2 Reducing Online facility location with Delay to Group Online Service with Delay

7.2.1 Intuition

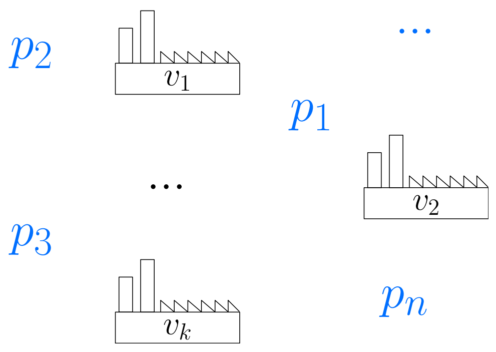
We are given a facility instance (as specified in section 5.1.2) with customer locations $P = \{1, 2, ..., n\}$, facility locations F, facility construction costs f_v for each $v \in F$ and a metrical distance function $d : P \sqcup F \to \mathbb{R}_{\geq 0}$.

We reduce this to an online facility location problem by creating a tree metric, with an example shown in fig 7.2, where again any reasonable gOSD strategy will commit partial Eulerian tours whose costs equal the cost of the corresponding facility orders, and which fulfil corresponding requests.

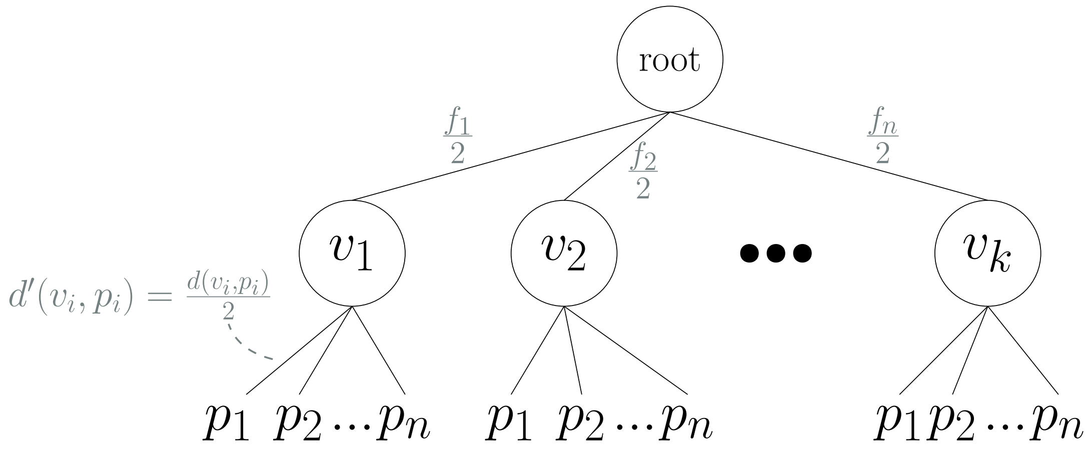
Our tree metric space has a root node with $|F|$ children, with each child corresponding to a facility $v \in F$ with distance $d(root, v) = \frac{f_v}{2}$. Each facility node then has $|P|$ children, with the distance from v_i to its child corresponding to $p_i \in P$ set to $\frac{d(v_i,p_i)}{2}$, half the distance between the facility and the customer in the original online facility location metric space.

Whenever we receive a request for a customer $p_i \in P$ we create a corresponding request in our gOSD instance on all the points corresponding to our customer - i.e. the i-th child of each facility. We again repeatedly send immediately deadlining requests at $root$, forcing the server to stay at the root aside from the occasional partial euclidean tour to serve some requests. If on one of these partial euclidean tours the server visits a node v representing a facility and then visits children of v corresponding to customers $p_1, p_2, ..., p_i$, we map that to the facility location decision to rent the facility v and connect it to customers $p_1, p_2, ..., p_i$.

Note on that partial euclidean tour we pay the same cost as we do to complete the order in facility location, and we have a correspondence between our requests.



(A) Our example online facility location instance, with facilities $v_1, v_2, ..., v_k$ and customers $p_1, p_2, ..., p_n$).



(B) The corresponding gOSD metric space. Note the leaf nodes are labelled by the customers they correspond to, but each is an individual leaf.

FIGURE 7.2. Diagram of an example online facility location instance, and the corresponding gOSD instance made by our reduction.

7.2.2 Formal definition

REDUCTION 7.2. *Online facility location to group online service with delay:*

$\rho_S(S)$: *We are given the facility starting input* $S = (P, F, d, f)$ *where* P *are the customer demand points* $P = \{1, 2, ..., n\}$, F *the potential facility location points,* $d : P \sqcup F \to \mathbb{R}_{\geq 0}$ *a metrical distance function and* $f : F \to \mathbb{R}_{\geq 0}$ *the facility construction costs. Let* $l = |F|$ *be the number of potential facility locations. We return a group online service with delay starting input:*

$$S' = (P', d', root) \textit{ where}$$
$$P' = \{root\} \cup F \cup (F \times P),$$
$$p' = root,$$

and $d'(i,j)$ the distance function for a tree metric as shown in fig 7.2 with root the root, the nodes labelled $v \in F$ a child of root with $d'(root, v) = \frac{f_v}{2}$, and the node labelled '(v,p)' with $v \in F, p \in P$ a child of v with $d'(v,(v,p)) = \frac{d(v,p)}{2}$.

$\rho_T(S,T)$: Given a series of time-steps $T = (T_1, T_2, ..., T_m)$ we send the time-step sequence:

$$T' = (T'_{1,0}, T'_{1,1}, T'_{1,2}, ..., T'_{1,2l},\ T'_{2,0}, T'_{2,1}, T'_{2,2}, ..., T'_{2,2l},\ ...,\ T'_{m,0}, T'_{m,1}, T'_{m,2}, ..., T'_{m,2l})$$

Each τ sequence starts by sending all the requests we just received on their corresponding gOSD points, and then $2l$ requests with an immediately deadlining request at the root. Thus:

$$T'_{\tau,0} = \{(P_r, c'_r) \mid r \in T_i \textit{ a request}\}$$

where $P_r := \{(v, p_r) \mid v \in F\}$ contains all the points corresponding to request r's customer

$$\forall \phi \geq 1\ T'_{\tau,\phi} = \{(root, c'_{\tau,\phi \textit{ deadline}})\}$$

with $c'_r(\tau,\phi) := c_r(\tau)$ the same delay function, but stretched so that the time-step $T'_{\tau,\phi}$ has delay costs corresponding to those charged in T_τ and $c'_{\tau,\phi \textit{ deadline}}$ is the delay function deadlining immediately:

$$c'_{\Gamma,\Phi \textit{ deadline}}(\tau,\phi) = \begin{cases} 0 & \textit{if } \tau \leq \Gamma \textit{ and } \phi \leq \Phi \\ \infty & \textit{if } \tau > \Gamma \textit{ or } \phi > \Phi \end{cases}$$

$\rho_\Sigma(\sigma')$: Given a strategy $\sigma' = (\sigma'_{1,0}, \sigma'_{1,1}, ..., \sigma'_{1,2l}, \sigma'_{2,0}, \sigma'_{2,1}, ..., \sigma'_{m,2l})$, we translate each τ sequence $\sigma'_{\tau,0}, \sigma'_{\tau,1}, ..., \sigma'_{\tau,2l}$ to the facility decision that builds a facility at each v visited by the server in that τ-sequence: $v \in F \cap (\sigma'_{\tau,0} \cup \sigma'_{\tau,1} \cup ... \cup \sigma'_{\tau,2l})$ and connects the facility built at v to every child of v the server visited in that time-step - i.e. to the set

$$L_v := \{p \mid (v,p) \in \sigma'_{\tau,0} \cup \sigma'_{\tau,1} \cup ... \cup \sigma'_{\tau,2l}\}$$

We show in the analysis section that every reasonable strategy for a reduced instance must have the server start and end a τ sequence at the $root$, and so if the server visits a point (v, p), it must have passed through v to get there, and so we only try to connect customers to facilities that we have actually rented.

Thus, formally:

$$\rho_\Sigma(\sigma'_{\tau,0}, \sigma'_{\tau,1}, \sigma'_{\tau,2}, ..., \sigma'_{\tau,2l}) = \{(v, L_v) \mid v \in F \cap (\sigma'_{\tau,0} \cup \sigma'_{\tau,1} \cup ... \cup \sigma'_{\tau,2l})\}$$

where L_v is as defined above.

7.2.3 Analysis

THEOREM 7.3. *Reduction 7.2 is an exactly cost preserving online reduction from the online facility location problem to group online service with delay.*

PROOF. We proceed by showing lemmas 7.4 and 7.5 which together directly prove our theorem.

LEMMA 7.4. *Every facility strategy σ has a corresponding strategy σ' in the reduced gOSD instance such that $\rho(\sigma') = \sigma$.*

PROOF. Let our facility strategy be denoted by $\sigma = (\sigma_1, \sigma_2, ..., \sigma_k)$.

Consider the gOSD strategy which maps σ_τ to the τ sequence that:

(1) For each facility rented in σ_τ, visits its corresponding point v_i, and then visits all the customers connected to v_i in σ_τ, in a partial Eulerian tour.
(2) Stays at the root for the remaining time-steps in the τ-sequence

Formally this is:

$$\sigma'_{(\tau,0)} = \begin{cases} root & \text{if } \sigma_\tau = \varnothing \\ \text{partial Eulerian tour through } \{(v,p) \mid (v, L_v) \in \sigma_\tau, p \in L_v\} & \text{if } \sigma_\tau \neq \varnothing \end{cases}$$

$$\forall \phi \geq 1, \sigma'_{(\tau,\phi)} = root$$

By the definition of ρ_Σ it is clear $\rho_\Sigma(\sigma'_i) = \sigma_i$ as required. □

LEMMA 7.5. *Every strategy in the reduced gOSD instance σ' has equal cost to its corresponding facility strategy, i.e.* $c'(\sigma') = c(\rho(\sigma'))$

PROOF. Consider some reasonable gOSD strategy for a gOSD instance made by our reduction: $\sigma' = (\sigma'_{1,0}, \sigma'_{1,1}, ..., \sigma'_{1,2l}, \sigma'_{2,0}, \sigma'_{2,1}, ..., \sigma'_{2,2l}, ..., \sigma'_{m,2l})$.

Similarly to the proof of lemma 6.5, we must first show that any reasonable strategy must spend each τ sequence performing an Eulerian tour starting and ending at the root. Then we can easily show the cost of performing this tour is equal to the order cost of σ_τ and we also have equal delay costs.

Note that as our gOSD instance has the server start at the root, if in every τ sequence we end at the root, a trivial induction implies we can always assume we start at the root.

Consider the τ sequence $(\sigma'_{\tau,0}, \sigma'_{\tau,1}, \sigma'_{\tau,2}, ..., \sigma'_{\tau,2l})$ starting at the root.

Similarly as to the proof of 6.5 we show that once the server visits a child of the root $v \in F$, after leaving that child's subtree and returning to the root, it can never revisit that child in a reasonable strategy.

In order to be reasonable, we can only visit a point in order to serve some request on it or pass through that point to visit some other point with a request on it. Thus when we return to the subtree the second time, we must be doing so to visit some point with a request on it. As we only send requests at the customer points, this request must be at some point (v, p). As we visited this subtree earlier, we have already visited v and decided not to visit (v, p) at cost $d(v, p)$. However we are now visiting p for cost at least $\frac{d(root,v)}{n} + d(v, p)$, an unreasonable decision.

Note within each subtree we must perform a partial Eulerian tour: we only have leaf nodes available to us, and once we visit a leaf node we fulfil the requests on it and so cannot return.

Thus as we start at the root, we perform a partial Eulerian tour within each subtree rooted at a child of $root$, and as we only visit each child of $root$ at most once, we must overall perform a partial Eulerian tour - except potentially ending at some leaf node instead of the root.

We now show that we must complete the partial Eulerian tour by returning to the root.

Consider the case where there is some time-step where the server stays at the root. As the state therefore stays unchanged for every subsequent time-step in the τ sequence - with all delay costs frozen and no requests fulfilled - in all subsequent decisions we must also stay at the root, as it would be unreasonable for the same algorithm to make a different decision given the same situation.

Consider the alternative case, where there is no time-step where the server only stays at the root. As the first time-step starts at the root, and every other time-step has a deadlining request at the root, we must always either start at, pass through, or end at the root. We can thus spend at most 2 time-steps in a single subtree, by first leaving the root and visiting the subtree, and then on the next time-step leaving the subtree and returning to the root. As we can only visit each subtree once we can thus avoid staying at the root for at most $2l$ time-steps. However, each τ sequence has $2l + 1$ time-steps, and thus on the last time-step we have no option but to end at the root.

Thus during the τ sequence $\sigma'_{\tau,0}, \sigma'_{\tau,1}, \sigma'_{\tau,2}, ..., \sigma'_{\tau,2l}$ the server must perform exactly a partial Eulerian tour of P.

Finally, note that this partial Eulerian tour has movement costs equal to the order costs of the corresponding facility decision: when we do a partial Eulerian tour of a subtree we pay the cost to move from the root to v, then for each customer we visit the distance from the facility to that customer, and then the cost to return to the root again - i.e. $2\frac{f_v}{2} + \sum_{(v,p) \text{ in tour}} 2\frac{d(v,p)}{2}$ which is exactly the cost we pay in facility to rent v and connect it to the customers.

Furthermore, our requests are fulfilled in both our gOSD instance and the facility instance at corresponding times (accounting for the $(\tau', \phi) \rightarrow \tau$ stretching), and so we pay equal delay costs.

Thus as both our cost components are equal,

$$c'(\sigma') = c(\rho(\sigma))$$

as required. $\square$

Lemmas 7.4 and 7.5 together meet conditions 2.3 and 2.4 for an exactly cost preserving online reduction and thus theorem 3.4 is proven as required. $\square$

7.3 Badly Reducing Group Online Service with Delay to Metrical Task System

In section 4.2 we showed that it's impossible to create an exact reduction from JRP to MTS without creating super-polynomially many MTS states with regards to the number of JRP items.

As reductions can be chained together, and each of our reductions between JRP, MLAP, facility, OSD and gOSD create an $O(n)$ sized metric space in our reduced problem's instance, for a metric space with n points in our primal problem, our superpolynomial lower bound on the number of MTS states needed for a reduction from JRP transitively applies to MLAP, facility, OSD and gOSD. In particular, if we could reduce a gOSD instance with n points to an MTS instance with polynomially many states, we could chain this together with our earlier reductions:

$$JRP \text{ with } n \text{ items} \xrightarrow{n+1} MLAP \xrightarrow{2(n+1)} OSD \xrightarrow{2(n+1)} gOSD \xrightarrow{\text{polynomial w.r.t } n} MTS$$

a contradiction.

Thus an exact reduction from any of JRP, MLAP, facility location, OSD and gOSD to MTS must create superpolynomially many states in MTS with regards to the size of their metric space. Having achieved this lowerbound on the number of MTS states, we now provide a reduction from gOSD to MTS showing an upperbound on the number of states needed. By chaining the reductions together this also provides a (superpolynomial) reduction for any of JRP, MLAP, facility and OSD to MTS.

7.3.1 Intuition

All of the problems investigated so far aside from MTS have a 'memory' aspect: they remember which requests so far have been fulfilled and which have not. MTS, however has no notion of unfulfilled requests, and has no aspect of memory/state aside from the current location of the server. This is what we exploit in 4.2: the only way MTS can remember something is by adding it as a 'state' in our metric space.

We want MTS to remember where in the metric space our server currently is, and what requests we have fulfilled so far. Thus each state is of the form (x, R) where x is a point corresponding to a point in the gOSD metric space, and R is the set of requests we have already fulfilled. In each time-step our cost vector's entry for (x, R) is the delay costs of all the requests we've received that aren't in R, and so correspond to the requests that remain unfulfilled.

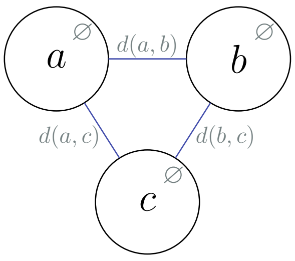
When we receive a new request r, we need to add new points to the MTS metric space to add the relevant new states (x, R) with r in R. We thus create the new points $\{(x, R) \mid x \in P, r \in R \subseteq \mathcal{R}\}$ where $\mathcal{R}$ is the set of all requests we've received so far. We define the distances between two points in our metric space as the shortest path between the points in an auxiliary graph. This auxiliary graph has the same points as our metric space. Two points that have fulfilled the same requests have an edge equal to their distance in the gOSD metric space, that is: $d'((x, R), (y, R)) = d(x, y)$. If we have two points (x, R) and (x, R') corresponding to the same gOSD point but different fulfilled requests, we connect them with an edge of weight 0 if all the requests in the symmetric difference of R and R' are in gOSD fulfilled by the server being on point x. i.e. those edges in

$$\{((x, R), (y, R')) \mid x = y, x \in P_r \; \forall r \in R \ominus R'\}$$

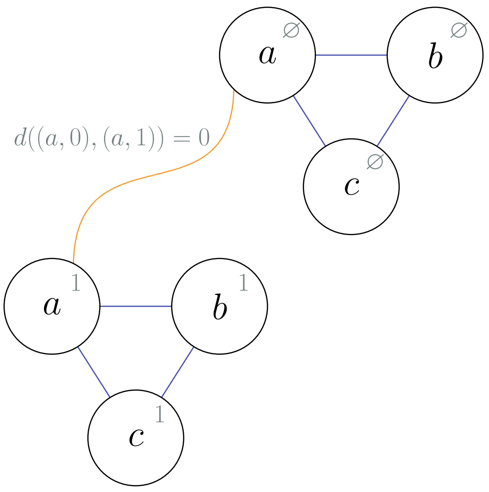
where $R \ominus R' = (R \setminus R') \cup (R' \setminus R)$ is the symmetric difference of R and R'.

An example of this auxiliary graph is shown in fig 7.3.

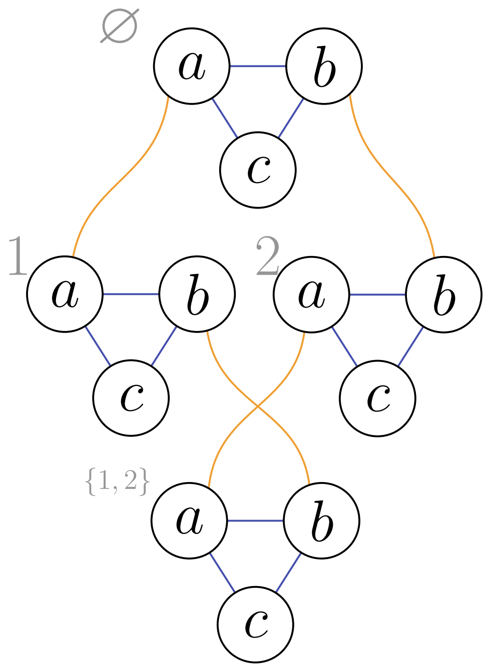
If the MTS server moves from point (x, R) to (x_k, R'), we move the OSD server through the corresponding points $x_1, x_2, ..., x_k$ which fulfil a request on the shortest path between (x, R) and (x_k, R') in the auxiliary graph.



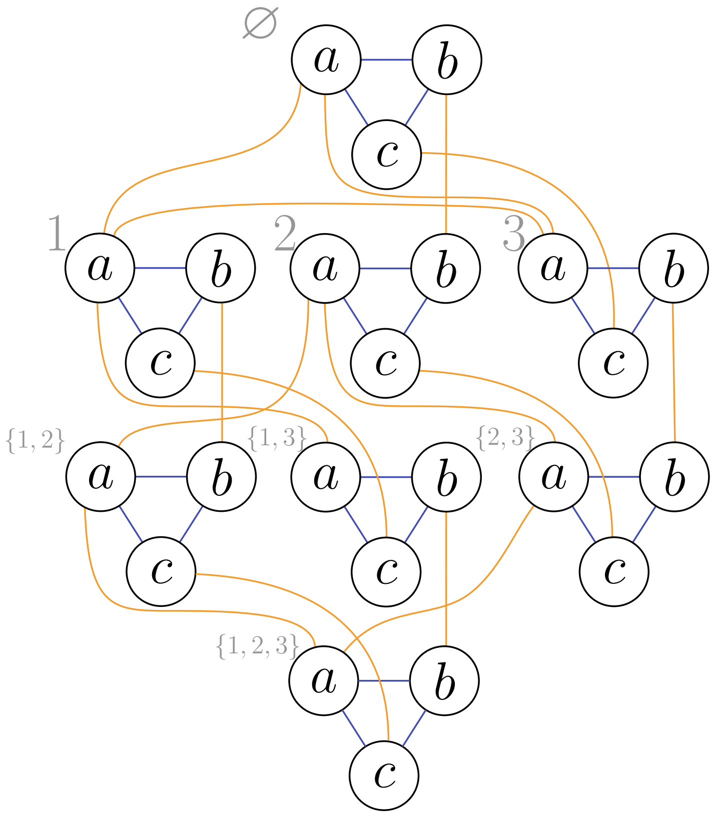
(A) Our initial auxiliary graph for a gOSD metric space with three points $\{a,b,c\}$. Edges coloured navy have weight equal to the distances between their corresponding points in the MTS metric space. The grey $\varnothing$ in each node denotes that those nodes correspond to MTS points $(x,\varnothing)$ with no requests fulfilled.



(B) Our auxiliary graph after we receive the first request 1, with $P_1 = \{a\}$. Navy edges continue to have the corresponding gOSD distance, while orange edges have weight 0.



(C) Our auxiliary graph after we've received a second request 2 with $P_2 = \{b\}$.



(D) Our auxiliary graph after we receive a third request 3 with $P_3 = \{a,c\}$. The numbers next to each cluster of nodes denotes the subset of requests those nodes have fulfilled.

FIGURE 7.3. Diagram of our auxiliary graph for an example gOSD instance. The metric space is the same set of points, with the distance between two points the length of the shortest path between them in the graph.

7.3.2 Formal definition of reduction

REDUCTION 7.6. *Group online service with delay to metrical task system:*

$\rho_S(S)$: *We are given the group online service with delay starting input* $S = (P, d, p')$. *We return a metrical task system starting input:*

$$S' = (P'_0, d'_0, (p', \varnothing)) \text{ where}$$
$$P'_0 = \{(p, \varnothing) \mid p \in P\}$$
$$d'_0((x, R), (y, R')) = d(x, y)$$

$\rho_T(S, T)$: *Given a series of time-steps* $T = (T_1, T_2, ..., T_m), T_i = \{r \mid r = (P_r, c_r) \text{ a request}\}$ *we send the time step sequence* $T' = (T'_1, T'_2, ..., T'_m)$ *with the time-step* $T'_i = (P'_i, d'_i, r_i)$ *where those terms are as defined below.*

P'_i *is the metric space as defined at time* i. *We first define* $\mathcal{R}_i = T_1 \cup T_2 \cup ... \cup T_i$ *the set of all gOSD requests received so far. We let our states be the Cartesian product of the points from gOSD with all possible subsets of* $\mathcal{R}$:

$$P'_i = P \times 2^{\mathcal{R}_i}$$

We define $d'(p, q)$ *as the shortest path in an auxiliary graph* G_i. *We define* G_i *as having the same points as* P'_i. *We have edges in the graph for every edge we define a weight for in the following:*

$$E_{(x,R),(y,R')} = \begin{cases} d(x, y) & \text{if } R = R' \\ 0 & \text{if } R \ominus R' \neq \varnothing \text{ and } x \in P_r \forall r \in R \ominus R' \end{cases}$$

We send the cost vector r_i *with a cost for the state* (x, R) *of:*

$$r_i((x, R) = \sum_{r \in \mathcal{R} \setminus R} c_r(i) - c_r(i - 1)$$

$\rho_\Sigma(\sigma_i)$: *If* σ_i *is a move from* (x_1, R_1) *to* (x_j, R_j), *we let* $(x_1, R_1), (x_2, R_2), ..., (x_j, R_j)$ *be the nodes in the shortest path in the auxiliary graph from* (x_1, R_1) *to* (x_j, R_j). *We return the gOSD strategy* $(x_1, x_2, ..., x_j)$, *the points in the gOSD metric space on that path, and remove any duplicate points* $x_i = x_i + 1$ *and points which don't fulfil any requests.*

We show the auxiliary graph is a valid metric space and that our reduction produces timestep sequences that are valid for MTS in the analysis section 7.3.3

Note this reduction works for an even stronger version of gOSD, where the delay cost we are charged in each time-step is some general function of the unfulfilled requests, rather than just a linear combination of functions for each unfulfilled request.

7.3.3 Analysis

THEOREM 7.7. *Reduction 7.6 is an order preserving online reduction from group online service with delay to metrical task system*

PROOF. We first show that our time-step inputs are valid in lemma 7.8, showing that we always modify our metric space correctly to only ever add points. Once we've shown we have a valid reduction, we show lemmas 7.9 and 7.10 directly showing conditions 2.1 and 2.2 of an order preserving online reduction and thus proving the theorem.

LEMMA 7.8. *The output of our reduction on a gOSD instance (S, T) is a valid instance of MTS with*

$$\rho_S(S) \in \bar{S'}_{MTS}, \rho_T(T) \in \bar{T'}_{MTS}(\rho_S(S))$$

PROOF. Our reductions initial starting input $S' = (P'_0, d'_0, (p', \varnothing))$ is trivially valid and in $\bar{S'}_{MTS}$. Note we relabel our points as (x, R) instead of $\{1, 2, ...n_0\}$.

We go through each time-step condition and show it is valid.

(1) $P'_{i-1} \subseteq P'_i$
$P'_{i-1} = P \times 2^{R_{i-1}} = P \times 2^{R_1 \cup R_2 ... \cup R_{i-1}}$ which is clearly a subset of $P'_i = P \times 2^{R_1 \cup R_2 ... \cup R_{i-1} \cup R_i}$ as if we have $(x, R) \in P'_{i-1}$ then $x \in P$, $R \in R_1 \cup ... \cup R_{i-1} \implies R \in R_1 \cup ... R_i \implies (x, R) \in P'_i$ as required.

(2) d'_i metrical. As our distances are defined as the length of the shortest path between two points in a graph with all edge weights ≥ 0, we have a metrical distance function.

(3) $d'_i((x, R), (y, R')) = d'_{i-1}((x, R), (y, R'))$ for every pair of points $(x, R), (y, R')$ in P'_{i-1}
As the distance between our two points is the shortest path in the auxiliary graph, in order to

decrease the distance there must be some new shorter path between (x, R) and (y, R') that passes through the new points we have added. That shortest path visits a series of points to fulfil the requests in $R \ominus R'$. The points that have been added in the auxiliary path include in their fulfilled requests, requests that have just been received and so are not in R or R': thus every state $(z, R^{\dagger})$ with these newly received requests fulfilled has a corresponding state $(z, R'^{\dagger})$ where $R'^{\dagger}$ has those newly received requests removed. We can thus replace this path through the newly created points, with a path through these corresponding points which will have a same length, but be present in the original graph. Thus this new path through the auxiliary graph cannot achieve a shorter distance, and so $d'_i((x, R), (y, R')) = (d'_{i-1}((x, R), (y, R'))$ as required.

(4) $r_i \in \mathbb{R}^{n_i}_{\geq 0}$: we defined the cost vector entry for every point as the increase in the relevant delay functions for that point. As the delay functions are non-decreasing, this value must be ≥ 0 as required.

□

LEMMA 7.9. *Our reduction meets condition 2.1: Every reasonable gOSD strategy σ has a corresponding strategy σ' in the reduced MTS instance such that $\rho(\sigma') = \sigma$.*

PROOF. Consider some reasonable gOSD strategy $\sigma = (\sigma_1, \sigma_2, ..., \sigma_m)$ where each decision is a path in the metric space from the start point to the end point, along the shortest path through some desired set of points that serve a series of requests: $\sigma_\tau = (\sigma_{\tau,1}, \sigma_{\tau,2}, ..., \sigma_{\tau,j_\tau})$.

Consider the MTS strategy that, in each time-step, moves to the point $(\sigma_{\tau,j_\tau}, R_\tau)$ where R_τ is the set of requests that have been fulfilled so far:

$$R_\tau := \{r \mid r \in T_1 \cup T_2 \cup ... \cup T_\tau,\;\; P_r \cap (\sigma_1 \cup \sigma_2 \cup ... \cup \sigma_\tau) \neq \varnothing\}$$
$$\sigma'_\tau = (\sigma_{\tau,j_\tau}, R_\tau)$$

Note that the mapping $\rho(\sigma')$ will map each decision back to the cheapest path of points in the metric space that fulfils the requests that σ fulfilled in that timestep, ending at the same point as σ does. As σ is a reasonable strategy, it must also visit those intermediate points in the way to serve those requests as cheaply as possible - otherwise it is making an extraneous cost for no reason. Thus each MTS decision must get mapped back to the same gOSD decision, and so $\rho(\sigma') = \sigma$ as required. □

LEMMA 7.10. *Our reduction meets condition 2.2: For every pair of strategies in the reduced MTS instance $\bar{\sigma'}, \bar{\omega'}$, the cheapest pair of strategies σ', ω' that produce the same gOSD strategy:*

$$\sigma' = \min_{\sigma''}\{\sigma'' \in \Sigma_{MTS} \mid \rho(\sigma'') = \rho(\bar{\sigma'})\} \qquad \omega' = \min_{\omega''}\{\omega'' \in \Sigma_{MTS} \mid \rho(\omega'') = \rho(\bar{\omega'})$$

have order preserved by ρ:

$$\begin{aligned} c'(\sigma') < c'(\omega') &\quad\Longrightarrow\quad c(\rho_\Sigma(\sigma')) < c(\rho_\Sigma(\omega')) \\ c'(\sigma') = c'(\omega') &\quad\Longrightarrow\quad c(\rho_\Sigma(\sigma')) = c(\rho_\Sigma(\omega')) \\ c'(\sigma') > c'(\omega') &\quad\Longrightarrow\quad c(\rho_\Sigma(\sigma')) > c(\rho_\Sigma(\omega')) \end{aligned}$$

PROOF. Consider some MTS strategy $\sigma' = (\sigma'_1, R_1), (\sigma'_2, R_2), ..., (\sigma'_m, R_m)$

As condition 2.2 only requires an ordering between the cheapest ways to produce a unique gOSD strategy, I first show that we can assume we only ever move to states that 'add' fulfilled requests, i.e. that

$$R_1 \subseteq R_2 \subseteq ... \subseteq R_m$$

Consider we have some $R_i \nsubseteq R_{i+1}$. Thus there must exist $r \in R_i$ such that $r \notin R_{i+1}$. Consider the strategy σ'' where we add r to R_{i+1}:

$$\sigma'' = (\sigma'_1, R_1), ..., (\sigma'_i, R_i), (\sigma'_{i+1}, R_{i+1} \cup \{r\}), ..., (\sigma'_m, R_m)$$

As we have already fulfilled r, and ρ only has the server visits points required to fulfil a request, and everything else about the path is unchanged, $\rho(\sigma'') = \rho(\sigma')$. Furthermore $c'(\sigma'') \leq c'(\sigma')$:

- As we have reduced the size of the set difference, with $|R_i \ominus (R_{i+1} \cup \{r\})| \leq |R_i \ominus R_{i+1}|$ (given $r \in R_i$), our new point is closer in the auxiliary graph to (σ'_i, R_i) and so we pay no more in movement costs
- As we are now at a state with more fulfilled requests, and our delay costs are positive, we pay no more to fulfil the cost vectors

Thus whenever σ' 'removes' a request, there exists a cheaper strategy that keeps this request and maps to the same gOSD strategy.

As the cost ordering conditions only apply to the cheapest reduced strategy that produces a given primal strategy, we consider the relations in conditions 2.2 for two strategies

$$\omega' = (\omega_1', R_1^{\omega'}), (\omega_2', R_2^{\omega'}), ..., (\omega_m', R_m^{\omega'})$$
$$\mu' = (\mu_1', R_1^{\mu'}), (\mu_2', R_2^{\mu'}), ..., (\mu_m', R_m^{\mu'})$$

for which we assume without loss of generality only ever fulfil requests:

$$R_1^{\mu'} \subseteq R_2^{\mu'} \subseteq ... \subseteq R_m^{\mu'}, \quad R_1^{\omega'} \subseteq R_2^{\omega'} \subseteq ... \subseteq R_m^{\omega'}$$

We now show that these strategies have equal cost to their induced gOSD strategy: $c'(\sigma') = c(\rho(\sigma'))$ with $\sigma' \in \{\omega', \mu'\}$.

Note that now that we only ever add requests to $R_i^{\sigma'}$, and when ever we add requests the induced gOSD strategy must visit a point that fulfils that strategy by the definition of ρ, we have a correspondence between the fulfilled requests in the induced gOSD strategy at a time i and the strategies in $R_i^{\sigma'}$. As our cost vector entry is the increase in delay costs for those requests not in $R_i^{\sigma'}$, which are those requests the gOSD strategy has left unfulfilled, we have equal delay costs.

Furthermore note that our σ' is reduced to an OSD strategy that visits the same points along the shortest path from $(\sigma_i', R_i^{\sigma'})$ to $(\sigma_{i+1}', R_{i+1}^{\sigma'})$, with some extraneous points removed that don't affect the path length. Thus we also pay equal movement costs between our MTS and gOSD strategies.

Thus $c'(\sigma') = c(\rho(\sigma'))$ as required. The relative ordering conditions in 2.2 directly follows. □

Lemma 7.8 thus shows reduction 7.6 is a valid reduction from gOSD to MTS, and lemmas 7.9 and 7.10 together meet conditions 2.1 and 2.2 for an order preserving online reduction and thus theorem 7.7 is proven. □

This gives the very first known deterministic algorithm for group online service with delay. As the work function algorithm on MTS is $2n - 1$ competitive, and we have an order-preserving online reduction which is thus a competitive ratio preserving online reduction, and $n_{MTS} = n_{gOSD} \times |2^{\{r|r \text{ a request}\}}| = n2^{\varrho}$, where ϱ is the number of requests we receive in gOSD, this algorithm is $(n2^{\varrho+1} - 1)$-competitive.

It is highly unlikely that this algorithm for gOSD is optimal, with this result mostly a side effect of our desire to show the existence of the reduction, rather than a serious attempt at producing a decent algorithm: theorem 4.1 meant from the outset we never expected this to be a 'good' reduction. The existence of a randomised $O(\log^2 n)$-competitive randomised algorithm for online service with delay (Azar et al., 2021) means we can expect a polynomial - and probably a polylog - competitive algorithm for gOSD, however showing that remains an open question.

The deterministic algorithm that chains $OSD \to gOSD \to MTS$ is also technically the first deterministic algorithm for online service with delay, again with a $(n2^{\varrho+1} - 1)$-competitive ratio. This is similarly a side-effect of our desire to show the reduction exists rather than a serious attempt to produce a good competitive ratio.

Finally, while this deterministic algorithm gives a bad exponential competitive ratio, there exist randomised polylog-competitive algorithms for MTS such as in Bartal et al. (1997) - the first polylog such algorithm - and Bubeck et al. (2021) - the current state of the art - which could, given an exponential number of states, produce a polynomially-competitive algorithm for gOSD. However, non-trivial work to show these randomised algorithms work on modifiable MTS (where we add points to the metric space) is needed, as the equivalency between non-modifiable and modifiable MTS no longer holds for randomised algorithms. As these randomised algorithms use a randomised metric embedding into a hierarchically separated tree, and building this metric embedding in an online fashion has a $\Omega(\log \Delta)$ lowerbound, where Δ is the aspect ratio of the metric as shown in Bartal et al. (2020), keeping the online metric space embedding reasonably performant will be a challenge. For this and other reasons, initial attempts to find a polynomially-competitive algorithm for gOSD will probably find greater success trying to modify the existing OSD algorithms to work in the gOSD setting.

CHAPTER 8

The offline version of Online Service with Delay is NP-hard

In this chapter we give a quick proof that finding the offline optimal solution to the online service with delay problem is NP-hard. While outside of the reduction hierarchy in fig 1.1, and the NP-hardness itself known in folklore, the implications for a reduction to MTS - as discussed in section 8.2 - are interesting, novel, and ultimately inspired theorem 4.1 and the fig 1.1 reduction hierarchy.

8.1 Proof Offline (Online) Service with Delay is NP-hard

Normally in online problems we are purely aiming for a good competitive ratio and have no regard for how computationally complex our algorithm is. We care even less about the computational complexity of the offline problem - that is, of computing the optimal strategy given the entire timestep sequence in advance.

However - despite the lack of attention normally given to the computational complexity - results about the time complexity of the offline problem for online service with delay have interesting implications for a reduction to metrical task system, explored in section 8.2.

THEOREM 8.1. *The offline solution to the online service with delay problem is NP-hard*

We proceed by reducing the travelling salesman problem to offline OSD.

REDUCTION 8.2. *A polynomial-time offline reduction from the offline problem of online service with delay to travelling salesman.*

Given an instance of a metric TSP where cities can be visited multiple times and the salesman doesn't need to travel back to the starting city, with cities the points $p_1, p_2, ..., p_n$ and distances between them given by the metrical distance function $d'(p_i, p_j)$, we construct the following instance of the OSD problem:

We leave the metric space the same , with points $P = (p_1, p_2, ..., p_n)$ and distance function $d = d'$.

For every point $p_i \in P$ create request r_i with cost function $c_{r_i}(t)$ a deadline at time $t = 1$:

$$c_{r_i}(t) = \begin{cases} 0 & t \leq 1 \\ \infty & t > 1 \end{cases}$$

And thus

$$T = (T_1, \varnothing)$$
$$T_1 = \{(p_i, c_{r_i}) | i = 1, ..., n\}$$

Note that this is a valid instance of the OSD problem from definition 6.1.

We map the optimal OSD strategy $OPT_{OSD} = (p_1, p_2, ..., p_k)$ - a sequence of cities to visit - to the TSP strategy that visits those same cities in the same order. We show the optimal OSD strategy allows such a mapping in the following lemmas.

LEMMA 8.3. *The optimal solution to the OSD instance pays 0 delay costs and visits all points at least once.*

PROOF. Assume for the sake of contradiction that the optimal solution has non zero delay costs.

As all of our delay costs c_{r_i} have values of either 0 or ∞, any strategy pays either infinite delay costs or no delay costs. Thus we can improve on the optimal solution by immediately fulfilling any unserved requests at $t = 1$, reducing the infinite delay cost to a finite movement cost, improving the optimal solution - a contradiction.

As this process fulfils all requests, and every point has a request, every point must be visited by the server at least once. □

LEMMA 8.4. *There is a strategy in our OSD instance that pays zero delay costs and maps to the optimal TSP solution. That is,*

$$\exists\, \sigma' \in \Sigma_{OSD} \text{ s.t. } \rho(\sigma') = OPT_{TSP}$$

and σ' pays only movement costs.

PROOF. The optimal solution for the TSP instance describes a path that visits all points in the metric space at least once. Consider the OSD strategy that follows the same path as the TSP instance, immediately visiting those points in the same order as TSP.

Note firstly this is a valid OSD strategy as we visit all points and so fulfil all requests, and that we do so before any requests deadline. Thus we pay no delay costs and so only pay movement costs for this strategy. Further note that this strategy maps to a TSP strategy that visits those same cities in the same order, and so maps to OPT_{TSP}. □

LEMMA 8.5. $\rho(OPT_{OSD}) = OPT_{TSP}$

PROOF. Assume, for the sake of contradiction, that $\rho(OPT_{OSD} \neq OPT_{TSP})$ - i.e. the optimal OSD strategy does not correspond to the optimal TSP strategy.

By lemma 8.3 we know OPT_{OSD} pays only movement costs and visits every point in some path.

As $\rho(OPT_{OSD})$ by assumption is not optimal, OPT_{TSP} must visit all the cities in some path shorter than OPT_{OSD}. By lemma 8.4, there exists an OSD strategy σ' that pays no delay costs such that $\rho(\sigma') = OPT_{TSP}$, and follows this shorter path tour. As σ' and OPT_{OSD} pay only movement costs, and σ' follows a shorter path than OPT_{OSD}, $c(\sigma') < OPT_{OSD}$, a contradiction. □

Lemma 8.4 thus shows we have a valid reduction from offline OSD to the travelling salesman problem. Offline OSD is thus NP-hard, concluding the proof of theorem 8.1.

8.2 Offline Metrical Task System is P (and implications)

For metrical task system, it is well known that "it is easy to construct a dynamic programming algorithm that gives an optimal [...] [offline strategy] for any [time-step] sequence" (Borodin et al., 1992) - specifically, offline metrical task system is in P. But we just showed offline OSD is NP-hard! We thus observe the following:

OBSERVATION 8.6. *Any order-preserving online reduction ρ can also be used to make an offline reduction.*

This is because our reduction function $\rho(T)$ does not receive the decisions the algorithm for our reduced problem is making and so we can, given all the timestep inputs, produce the timestep output all at once.

We then give all of this timestep output to our offline algorithm for the reduced problem, which produces a strategy we map back to our primal problem. As our online reduction was an order-preserving online reduction, every strategy in the primal problem - including the optimal - must have some corresponding strategy in the reduced problem. Because it is cost preserving, whatever strategy maps to the optimal must be cheaper than every other problem, and so the optimal reduced strategy maps to the optimal primal strategy.

Hence we get the following result:

THEOREM 8.7. *Any online order-preserving reduction ρ from OSD to MTS that, has a total run time for all the timestep inputs that is polynomial, must create a superpolynomial sized MTS input with regard to the number of OSD states n, assuming that $P \neq NP$.*

PROOF. Assume, for the sake of contradiction, that there exists ρ an online reduction from OSD to MTS which runs in polynomial time, and creates a polynomially sized MTS input with regards to the number of OSD states n.

Per observation 8.6 we can treat ρ as an offline reduction.

Let us have an instance of the travelling salesman problem with n' points in the form required for reduction 8.2, which from section 8.1 is an NP-hard problem.

Using the reduction 8.2 we can create an OSD instance with $n = n'$ points and n requests.

We can now use ρ on this instance to create an MTS instance with polynomially sized input, in polynomial time. Per (Borodin et al., 1992) we can find the offline optimal strategy to this MTS instance in polynomial time, and then convert that back to the optimal OSD strategy, and then back to the optimal solution for the travelling salesman problem instance.

We thus have a polynomial-time algorithm to solve the travelling salesman problem, a contradiction assuming $P \neq NP$. □

MTS is considered a significant online problem because it has a good known-tight competitive ratio of $2n - 1$, and is very flexible allowing many problems to be reduced to it. However, this simple proof shows it is very difficult to reduce OSD to MTS *well*. In particular, your reduction has to do one of the following:

- Produce a superpolynomial number of MTS states. This is bad as this causes a superpolynomial competitive ratio.
- Do something strange and complex to produce polynomially many MTS states, but superpolynomially many cost vectors, in a way where those cost vectors can't be simplified into only polynomially many while still being correct.
- Do something very strange and complex to run in superpolynomial time, in a way without any 'busy work' that could be removed to run in polynomial time while still being correct, without just solving the OSD instance directly (as then the reduction is pointless), while producing a polynomially sized MTS output.

This result has been strengthened by theorem 4.1, in particular with definition 2.7 of an order preserving online reduction capturing a broad class of behaviour most 'reasonable' online reductions exhibit, which must be avoided as part of the 'doing something very strange and complex'. Thus, while it might still be possible to reduce OSD to MTS, such a reduction would need to be very complex and exhibit strange behaviour. Hence, the flexibility of MTS that allows many problems to be reduced to it, is of limited practical use.

CHAPTER 9

Conclusion

We, in this thesis, having introduced a formalised notion of an online reduction, have shown the reduction hierarchy in fig 1.1 that relates JRP, MLAP, online facility location, OSD and gOSD. Some of these relations such as that between JRP and MLAP were previously known in the folklore, however many of them are novel. An open problem is showing that the remaining reduction possibilities between the problems in fig 1.1 - such as a reduction from OSD to JRP, or from online facility location to MLAP - are either not possible, not possible without a bad competitive ratio, or are possible with a good competitive ratio. Another area of further research with regards to the reduction hierarchy is to investigate how other problems, such as k-Server or min-cost perfect matching with delay, fit or don't fit into the hierarchy.

We have further introduced the group online service with delay (gOSD) problem, with the strong motivation for its study as a natural extension of OSD that is likely needed for the reduction from online facility location, with standard OSD unable to capture the online facility location problem. We also gave an online reduction from gOSD to metrical task system, that was constrained by the need for superpolynomial many states in the MTS metric space per theorem 4.1, that nevertheless induces the first known deterministic algorithm for group online service with delay with a $n2^{(\text{num requests}+1)}$-competitive ratio. This suggests another open problem, to try to improve on this upper bound by achieving a polynomial or polylog competitive ratio. Further areas of research include finding a lower bound for gOSD, and to investigate the other natural extensions of OSD in the group setting such as k-gOSD - where there are k servers instead of just one - and gOSD with the delay cost a function of the set of unfulfilled requests, rather than just a linear combination of functions associated with each individual unfulfilled request - a setting which our reduction to MTS continues to work for.

Finally we have shown no online reduction, under certain reasonable assumptions, can exist between any problem in the fig 1.1 reduction hierarchy and metrical task system without creating superpolynomially many states, and thus a superpolynomial induced competitive ratio. We also gave a reduction from gOSD, and so by combining reductions from all the problems in the reduction hierarchy, to MTS. Further areas of research include trying to apply a similar proof, of the need for superpolynomially many MTS states, to other problems with similar aspects such as k-Server, and to investigate how these proofs of a 'bad' relationship between problems impacts the performance of algorithms such as the work function algorithm that work, with minor modification, on multiple problems.

Finally, the particular choices behind the design of the online reduction framework was heavily guided by the experience of exploring reductions between the various problems investigated in this thesis. While care was taken to try to keep the definition as general as possible, further areas of research would include expanding this framework to new situations. Particular areas that, in the course of making this thesis, seemed interesting but weren't pursued as they weren't relevant to the investigated problems included modifying the competitive-ratio preserving online reductions to work when the black-box algorithm for the reduced problem is a randomised algorithm; exploring a randomised reduction, where the produced reduced problem input is randomised instead of deterministic, which could help relate randomised and deterministic algorithms by containing the randomness to a well-defined area; and approximate order-preserving reductions, where the ordering of strategies is only maintained between problems whose costs differ by more than some factor.

Bibliography

Aris Anagnostopoulos, Russell Bent, Eli Upfal, and Pascal Van Hentenryck. 2004. A simple and deterministic competitive algorithm for online facility location. *Information and Computation*, 194(2):175–202.

Yossi Azar, Arun Ganesh, Rong Ge, and Debmalya Panigrahi. 2021. Online Service with Delay. *ACM Transactions on Algorithms*, 17(3):1–31.

Yossi Azar, Chay Machluf, Boaz Patt-Shamir, and Noam Touitou. 2022. Competitive Vertex Recoloring. preprint, In Review.

Yossi Azar and Noam Touitou. 2019. General Framework for Metric Optimization Problems with Delay or with Deadlines. In *2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 60–71. ISSN: 2575-8454.

Yossi Azar and Noam Touitou. 2020. Beyond Tree Embeddings – a Deterministic Framework for Network Design with Deadlines or Delay. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1368–1379. ISSN: 2575-8454.

Yair Bartal, Avrim Blum, Carl Burch, and Andrew Tomkins. 1997. A polylog(n)-competitive algorithm for metrical task systems. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, STOC '97, pages 711–719. Association for Computing Machinery, New York, NY, USA.

Yair Bartal, Nova Fandina, and Seeun William Umboh. 2020. Online probabilistic metric embedding: a general framework for bypassing inherent bounds. In *Proceedings of the Thirty-First Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '20, pages 1538–1557. Society for Industrial and Applied Mathematics, USA.

Marcin Bienkowski, Martin Böhm, Jaroslaw Byrka, Marek Chrobak, Christoph Dürr, Lukáš Folwarczný, Łukasz Jeż, Jiří Sgall, Nguyen Kim Thang, and Pavel Veselý. 2016. Online Algorithms for Multi-Level Aggregation. ArXiv:1507.02378 [cs].

Allan Borodin, Nathan Linial, and Michael E. Saks. 1992. An optimal on-line algorithm for metrical task system. *Journal of the ACM*, 39(4):745–763.

Sébastien Bubeck, Michael B. Cohen, James R. Lee, and Yin Tat Lee. 2021. Metrical Task Systems on Trees via Mirror Descent and Unfair Gluing. *SIAM Journal on Computing*, 50(3):909–923. Publisher: Society for Industrial and Applied Mathematics.

Niv Buchbinder, Tracy Kimbrel, Retsef Levi, Konstantin Makarychev, and Maxim Sviridenko. 2013. Online Make-to-Order Joint Replenishment Model: Primal-Dual Competitive Algorithms. *Operations Research*, 61(4):1014–1029.

Ryder Chen, Jahanvi Khatkar, and Seeun William Umboh. 2022. Online Weighted Cardinality Joint Replenishment Problem with Delay. In Miko\laj Bojańczyk, Emanuela Merelli, and David P. Woodruff, editors, *49th International Colloquium on Automata, Languages, and Programming (ICALP 2022)*, volume 229 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 40:1–40:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany. ISSN: 1868-8969.

Marek Chrobak and Lawrence L. Larmore. 1998. Metrical task systems, the server problem and the work function algorithm. *Online Algorithms*, pages 74–96. Publisher: Springer, Berlin, Heidelberg.

Dimitris Fotakis. 2008. On the Competitive Ratio for Online Facility Location. *Algorithmica*, 50(1):1–57.

Elias Koutsoupias and Christos H. Papadimitriou. 1995. On the *k*-server conjecture. *Journal of the ACM*, 42(5):971–983.

Ngoc Mai Le, Seeun William Umboh, and Ningyuan Xie. 2023. The Power of Clairvoyance for Multi-Level Aggregation and Set Cover with Delay. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*.

OEIS Foundation Inc. 2022. Entry A000984 in the On-Line Encyclopedia of Integer Sequences. Published electronically at http://oeis.org/a000984.

APPENDIX A

Table of common notation

While we have tried to keep the amount of new notation introduced in this thesis to a minimum, the introduction of online reductions, as well as the need to unify related concepts between our many problems (to avoid introducing the different notation for the point-set or the distances or the requests each of JRP, MLAG, facility and OSD use in the literature), necessitated the introduction of a still sizeable amount of notation. This table aims to summarise the meaning of symbols that are intentionally used in multiple chapters with the same or a related meaning.

TABLE A.1. Table of common notation

Symbol	Meaning
$\bar{S}$	The set of all valid starting inputs of some online problem. From definition 2.1.
$\bar{T}(S)$	The set of all valid timestep input sequences, for starting input S, for some online problem. From definition 2.1.
$\bar{D}(S,T)$	The set of all valid decision sequences, for starting input S and timestep input T, for some online problem. If a single decision, valid at one timestep, is valid at all timesteps, may be written as the set of all valid decisions for a single timestep for notational simplicity. From definition 2.1.
c	The cost function - given an instance of an online problem and a strategy, outputs the cost. From definition 2.1.
σ	A strategy - a sequence of decisions an algorithm makes in an online problem. From definition 2.3.
S	A particular starting input, part of an instance of an online problem. From definition 2.2.
T	A particular timestep input, part of an instance of an online problem. From definition 2.2.
Σ_P	The set of all possible strategies for problem P, normally for some contextual relevant instance of P.

TABLE A.1. Table of common notation

Symbol	Meaning
T_τ,T_i,$T_{\tau,\phi}$	A single timestep from our timestep input, at time $\tau/i/\tau'/(\tau,\phi)$.
τ,i	A generic index to represent the time, as the index of a timestep input / decision.
ϕ	A generic timestep for the time 'within' a τ-sequence in the reduced problem. e.g. $T_{\tau,\phi}$ is the ϕ-th timestep received after the first timestep corresponding to time τ in the primal problem.
ρ	An online reduction. May also be used to denote ρ_Σ, the mapping of a strategy. From definition 2.4.
ρ_S	Part of an online reduction, a function to map the starting input from problem P to R. From definition 2.4.
ρ_T	Part of an online reduction, a function to map the timestep input from problem P to R. From definition 2.4.
ρ_Σ	Part of an online reduction, a function to map a decision from problem R to P. From definition 2.4.
P	The primal problem - we reduce from the primal problem P to the reduced problem R. May also be used to denote P, the point set.
R	The reduced problem - we reduce form the primal problem P to the reduced problem R.
P	The point set - the set of points in our metric space. May also be used to denote P the primal problem.
n	$\|P\|$, the size of our metric space.
d	The distance function for our metric space.
P', d'	Our metric space P and d, but for our reduced problem rather than the primal problem.
r	A request
c_r	The delay cost function for request r.
p_r	The point/item from the metric space request r is 'on'.
t_r	The time request r arrives.
t'_r	The time request r is fulfilled.
m	The last timestep in a problem instance.
k	The last timestep received 'so far' - i.e. the last timestep in a strategy.