Online Algorithms for Temperature Aware Job Scheduling Problems

Thesis submitted for the degree of

Doctor of Philosophy

at the University of Leicester

By

Martin David Birks BSc

Department of Computer Science

University of Leicester

2012

Online Algorithms for Temperature Aware Job Scheduling Problems

Martin Birks

Abstract

Temperature is an important consideration when designing microprocessors. When exposed to high temperatures component reliability can be reduced, while some components completely fail over certain temperatures. We consider the design and analysis of online algorithms; in particular algorithms that use knowledge of the amount of heat a job will generate.

We consider algorithms with two main objectives. The first is maximising job throughput. We show upper and lower bounds for the case where jobs are unit length, both when jobs are weighted and unweighted. Many of these bounds are matching for all cooling factors in the single and multiple machine case.

We extend this to consider the single machine case where jobs have longer than unit length. When all jobs are equal length we show matching bounds for the case without preemption. We also show that both models of preemption enable at most a slight reduction in the competitive ratio of algorithms. We then consider when jobs have variable lengths. We analyse both the models of unweighted jobs and the jobs with weights proportional to their length. We show bounds that match within constant factors, in the non-preemptive and both preemptive models.

The second objective we consider is minimising flow time. We consider the objective of minimising the total flow time of a schedule. We show NP-hardness and inapproximability results for the offline case, as well as giving an approximation algorithm for the case where all release times are equal. For the online case we give some negative results for the case where maximum job heats are bounded. We also give some results for a resource augmentation model that include a 1-competitive algorithm when the extra power for the online algorithm is high enough. Finally we consider the objective of minimising the maximum flow time of any job in a schedule.

Acknowledgements

I would like to thank everyone who has helped and supported me during my PhD studies and the creation of this thesis. First and foremost, I would like to thank my supervisor Dr Stanley P. Y. Fung for his help, advice and insights that have been invaluable throughout my research. I know that I have benefited immensely from his guidance, this thesis simply would not have been possible without it.

A great thanks is also due to all staff in the Department of Computer Science at the University of Leicester during all of my time in Leicester, through both my undergraduate and PhD studies they have always been friendly and helpful.

I would like to thank my colleagues in the office G1 and all of the other GTA's within the department that have been a great source of encouragement whenever it was needed. All of my good friends from my time in Leicester have been fantastic, always there to listen to me during the ups and downs, as well as providing all important entertainment outside of work.

I am extremely grateful to my family who have provided unlimited support for me throughout my whole life. Finally, I'd like to thank Emma for always being there whenever I needed her to be. Without her support, it all would have been more difficult.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In this thesis we present algorithms and analysis for a variety of optimisation problems. The particular problems that are studied are all *scheduling* problems, with the additional constraint of *temperature awareness*. In this work we consider this very real problem within a theoretical context.

Considering these problems under a theoretical context will contribute towards understanding how knowledge of temperature can be used to create better scheduling algorithms for microprocessors. This work also helps with the understanding of the limitations of what can be achieved when designing algorithms for environments where temperature is a major constraint.

1.1 Scheduling

Scheduling is the process by which resources are allocated to tasks. Deciding on the best way to allocate resources to particular tasks is an important problem that occurs in an assortment of different contexts, and with a variety

of different motivations and objectives. With infinite resources all tasks could be completed instantly but in the real world this is rarely, arguably never, a possibility. Because of this, decisions often have to be made that result in some task being completed before others.

Depending on the specific problem it is possible that completing certain tasks before others will be considered as somehow 'better' than if those tasks were completed in a different order. What constitutes a better order will depend on an objective being decided to assess the quality of a given schedule. There are many situations where it may not be possible to complete all the available tasks and so in these cases decisions will have to be made on which tasks to complete, and which tasks to ignore.

To create the best, or *optimal*, schedule it is common that some objective has to be maximised or minimised while some constraint or constraints (e.g. available resources) have to be maintained. It is scheduling algorithms that we use to provide solutions to these optimisation problems.

These problems are common in an array of real world scenarios. For example, a manufacturing company will have to balance the orders it accepts against its capacity to fulfil those orders. There are likely to be several different possibilities regarding which orders to accept and when to allocate certain resources to them but the company will have to decide which is best for their particular situation. In other words, a schedule will need to be constructed that best meets the companies objectives (e.g. maximising long term profit) while not violating any of its constraints (e.g. resource limitations).

Within the context of computing, scheduling algorithms are used at several different levels, such as packet forwarding in routers, or printer spooling.

The context focused on in this work is the use of scheduling algorithms to allocate jobs within microprocessors. There are many different ways that this problem can be composed with various different complexities: the algorithm may have access to several machines that may or may not be identical; the machine(s) may be able to run at different speeds; jobs may have different priorities and so on. There are also many different ways that the quality of a schedule can be assessed, such as minimising how long a job has to wait to be scheduled on average, or minimising the maximum waiting time of any job.

The problem of creating and analysing scheduling algorithms is an important area for theoretical research and has been extensively studied (without temperature constraints) for a wide range of different models and objectives. When studying these many problems in scheduling theoretically, it is common to define the problem using 3-field notation [31]. A review of several different scheduling problems and their representation in 3-field notation can be found in [16].

1.1.1 Computational Time Complexity

An important area in the study of scheduling algorithms is regarding the computational time complexity of the algorithms. This typically involves analysing algorithms to find how much time they take to create a schedule and how this grows with the number of jobs that need to be scheduled, commonly denoted by n. An algorithm is said to run in polynomial time if the time it takes to find a solution to a problem grows polynomially with n.

A decision problem is said to belong to the complexity class NP if it is possible to decide whether a given solution to the problem is correct in polynomial time. A subset of NP is the complexity class P, which contains all the decision problems that can be solved in polynomial time. A problem is said to be NP-complete if it is in NP and any other problem in NP can be reduced to it in polynomial time. It is unknown whether $P = NP$, although this is a widely studied and important problem. For more details on computational complexity theory, see e.g. [27]. Many important optimisation problems are in NP but no polynomial time algorithms are known that find an optimal solution.

One strategy for finding efficient algorithms for NP-complete problems is to design *approximation* algorithms. An approximation algorithm is an algorithm that produces a solution with a guaranteed worst case quality called an *approximation ratio*. This is the ratio of the quality of the solution produced by the approximation algorithm compared to the optimal solution. For example, an approximation ratio of 2 means the solution of the approximation algorithm will be no worse than 1/2 the quality of the optimal solution. As it is unlikely that an efficient algorithm can be found that solves an NP-complete problem optimally, it can be useful to design an approximation algorithm that provides a lower quality solution, but in polynomial time.

1.2 Online Algorithms

A problem is called *online* if not all of the information about the problem is available to an algorithm at the start of execution. Information will then

become available to the algorithm as time progresses. In the context of job scheduling problems, this often means that knowledge of future jobs and their properties is not necessarily available to the algorithm at a given moment in time.

There are a large number of real world problems that are inherently online. A microprocessor within a computer is unlikely to know much information about future jobs as these will be dependent on the input of the user. It is therefore clearly important to design good quality algorithms that perform well even without this knowledge of the future, as this is how these algorithms are likely to be used in reality.

Not having complete knowledge of the future can lead to decisions being made that compromise the quality of the schedule produced by the algorithm. Therefore an online algorithm has to balance the potential future impact of the decisions it makes, against making the best decisions for the present situation.

1.2.1 Competitive Analysis

When designing algorithms it is important to have a method of deciding their quality. We use *competitive analysis* for analysing the quality of an online algorithm. This was proposed in [47] and has since become the standard framework for analysing this type of algorithm. Competitive analysis involves comparing the quality of the solution produced by the online algorithm to that of some optimal *adversary*.

The adversary is offline, meaning it will possess all of the information

about future jobs that the online algorithm cannot be aware of. The adversary then uses this information to make optimal decisions that the online algorithm could not make, as it does not have all the information necessary to make such a decision. There are different types of adversaries with different powers, these are described in Section 1.4.

The result of a competitive analysis is a *competitive ratio* $c \geq 1$. This is the worst case ratio of how well the online algorithm performs against the adversary for some chosen objective, over all possible inputs. For example, if a scheduling algorithm is 2-competitive for throughput of jobs, then it will always schedule no less than $1/2$ of the number of jobs that the optimal adversary will schedule.

More formally we define the competitive ratio of some online algorithm $\mathcal{A}$ as c for some maximisation objective (e.g. maximising job throughput) where

$$|OPT| \leq c \times |\mathcal{A}| + \alpha$$

holds for all possible inputs, with $|OPT|$ and $|\mathcal{A}|$ the values of the schedules produced by OPT and $\mathcal{A}$ respectively. The competitive ratio of the online algorithm for a minimisation objective (e.g. minimising the schedule's flow time) is defined as

$$|\mathcal{A}| \leq c \times |OPT| + \alpha.$$

Throughout this thesis we will assume that $\alpha = 0$, this is often known as a *strict* competitive ratio. When analysing randomised algorithms we use the same definitions but consider the expected value of the online algorithm, i.e. $E[|\mathcal{A}|]$.

Note that the smaller the competitive ratio the better the algorithm performs and if an online algorithm has a competitive ratio of exactly 1 for some objective, this means that over any input it will perform as well as an offline algorithm. For a more detailed discussion of competitive analysis and online algorithms see [14].

1.2.2 Resource Augmentation

Resource augmentation for online algorithms was proposed in [35]. This technique involves giving the online algorithm more resources, as a way to compensate it for the lack of future information. Examples of resource augmentation would be giving the online algorithm an extra machine compared to the adversary, or increasing the speed of a machine. Analysing resource augmentation is useful to show the effect that increasing resources can have in improving algorithm performance, and is a technique that is widely used in the study of online algorithms. A good example of this is in [43], where they show that several well known algorithms with poor performance for certain problems can perform optimally for these problems with only a moderate increase in resources.

1.3 Preemption

An important consideration with scheduling algorithms is whether jobs can be interrupted between being started and completing, this is known as *preemption*. There are three models that we consider regarding preemption:

No Preemption Once a job J is started by an algorithm on a machine m

it cannot be interrupted and m is unable to start another job until J has completed.

Preemption with restarts A job J that is started by an algorithm can be interrupted but if this occurs the algorithm loses any progress already made in the execution of J. This means that if J is restarted, it must be from the beginning.

Preemption with resumes A job J that is started by an algorithm can be interrupted and the algorithm retains all progress already made in the execution of J. This means that J resumes from the point it was preempted if an algorithm chooses to schedule it at a later time.

When considering preemption it is important to distinguish between the offline and online models. When we are considering the offline case, preemption with restarts is equivalent to allowing no preemption. This is because the schedule of an offline algorithm that uses preemption with restarts can never be made worse by removing any parts of jobs that were started but not completed. It is therefore always possible to create a schedule without preemption from a schedule that uses restarts when the algorithm is offline.

Combining multiple machines and preemption allows for the possibility that jobs can be *migrated* between machines. This is when a job is started on some machine but then preempted and moved to a different machine. This might occur for many reasons, such as the new machine is a different temperature, or the new machine executes jobs at a different speed to the current machine.

1.4 Randomisation

Algorithms can generally be split into two different categories: *deterministic* and *randomised*. Deterministic scheduling algorithms will always generate the same schedule if given the same set of inputs and the same resources. Randomised algorithms will take a set of inputs and resources and can generate several schedules, each with a given probability.

For competitive analysis of deterministic algorithms, the adversary always has the same power while the analysis of randomised algorithms can depend on which type of adversary is used, and how much power it has.

There are 3 types of adversary (as described in [10], but shown here for completeness), listed in order of increasing strength:

Oblivious Adversary knows the full details of the algorithm; does not know the result of any random decisions that have been made; serves the input offline and optimally.

Adaptive Online Adversary knows the full details of the algorithm and any random decisions that have been made; but serves the input online.

Adaptive Offline Adversary knows the full details of the algorithm and any random decisions that have been made; serves the input offline and optimally.

The adaptive offline adversary is so strong that randomisation can never give an algorithm any additional power over it [10]. When considering randomised algorithms in this work we will always use an oblivious adversary.

1.5 Scheduling Objectives

We now examine in some more detail the two online scheduling problems that are considered in this work.

1.5.1 Maximising Throughput

Maximising the throughput of jobs is a common goal for scheduling algorithms. This is simply maximising the total number of jobs that are completed by an algorithm. For these problems jobs usually have release times, processing times/lengths and deadlines. Before the release time of a job no algorithm can schedule the job, and an online algorithm will not know it exists. The processing time/length of a job is how much work the algorithm has to do after starting the job before it is completed. If a job is not completed in a schedule before its deadline then that schedule will gain no value for that job. In some cases the jobs have weights and the objective changes to maximising the total weighted throughput of jobs, where the weight of the job is the value gained by a schedule for completing it.

1.5.2 Minimising Flow time

Minimising the flow time of jobs is another common goal for online scheduling algorithms. The flow time of a job is the difference between its release time and the time it is completed. There are two sub-versions of this problem that we consider: minimising the maximum flow time of a job in a schedule and minimising the total/average flow time of a schedule. If the release times of all jobs are the same then the problem of minimising the maximum flow

time becomes the same as minimising the makespan of a schedule, which is another widely studied scheduling objective.

For minimising the total/average flow time of a schedule it is well known that the shortest remaining processing time (SRPT) is optimal for the single machine case [44] (and references therein).

1.6 Temperature Awareness

High temperatures are a problem when designing computational devices as they can reduce the reliability of components, incur high cooling costs and even cause permanent device failures [21]. Over recent years there has been a trend towards the increasing popularity of devices that are both small/compact and powerful. This has only accentuated the problem as dissipating the heat generated by small devices is more difficult than in a larger device.

Significant amounts of research have been done to address this issue. A lot of this research has been done at the level of microprocessor architecture and hardware but in this work we look at the problem from an algorithmic perspective. The temperature of a processor is related to both its cooling mechanism and power use, with power use being a convex function of the processor's speed [33]. One technique that takes advantage of this to manage temperature is called *Dynamic Voltage Scaling* (DVS) and algorithms using this technique have been analysed both empirically and theoretically.

A related objective to controlling temperature is the objective of minimising energy use. The paper that instigated the study of this objective for the

DVS model is [53]. In this work the authors first study the offline problem giving a $O(n \log^2 n)$ time algorithm that computes the energy optimal schedule. They go on to give two natural algorithms for the online version of the problem with their competitive ratios given in terms of α where the function P that relates power to speed is $P(s) = s^\alpha$, where s is the processor speed. The algorithm AVR is $2^{\alpha-1}\alpha^\alpha$-competitive, while OA has a tight bound of α^α, with respect to energy usage.

In [4] they show the effectiveness of energy aware scheduling empirically. They use an algorithm with the aim of "dynamically monitoring and reclaiming the 'unused' computation time..." and show that this is a powerful approach. They give experimental results that show this dynamic reclaiming can give a 50% energy saving over a static algorithm.

Some microprocessor systems are limited to work at a finite number of discrete power levels, as opposed to a continuous range of power levels limited only by some minimum and maximum power setting. This model is considered in [40] where they give algorithms and experimental results to show the effectiveness of their techniques.

Procrastination scheduling is a technique that has been created to exploit the amount of slack that jobs have available to reduce energy use. This has been discussed in [34] in combination with DVS. Their objective and motivation was to minimise energy wastage in the overhead between switching the processor between idle and running modes. It was shown that significant savings in energy use can be made by extending idle periods between jobs when this is possible.

For more details and results for the problem of energy efficient scheduling

see the survey in [1].

Note that the problem of managing temperature is related to, but not the same as the problem of minimising energy use. Although the temperature of a processor is related to the amount of power that is used, there are several differences and this leads to different techniques being used for the different problems. For example job migrations from a hot to a cool processor can be very useful in managing temperature, while this is not a technique that is used to manage energy use; in fact it may cost extra energy in the overhead costs associated with migrating a job.

In some ways it can be argued that managing temperature is more important than managing energy: "If the processor in a mobile device exceeds its energy bound, then the battery is exhausted. If a processor exceeds it thermal threshold, it is destroyed." [7] They also emphasise that power management schemes focus on reducing cumulative power use, while to reduce temperature more consideration must be given to the power usage at a certain instant. It has also been shown that "...many low-power techniques have little or no effect on operating temperature" [45]. See the survey in [33] and references therein for more details on the relationship between the two problems.

An example of some empirical work considering temperature awareness can be be found in [51]. They use temperature awareness as a tool in their algorithm to improve the performance of the microprocessor, rather than with minimising maximum temperature being the primary goal itself. They show that considering temperature when scheduling jobs can actually be used to give performance benefits within a computer.

In [21] they also use empirical methods to judge the quality of their algorithms. They aim to give algorithms that reduce hot spots and temperature variation across multiprocessor systems, with minimal performance costs. They present two algorithms, both set a temperature threshold (85 °C) and when a processor reaches this the first algorithm migrates the job to a cooler core, while the second slows down the speed of the core (DVS). They show that these techniques do indeed achieve "...low and balanced temperature profiles at minimal performance cost." [21]

The problem of using DVS in temperature aware algorithms has been theoretically analysed in [7], building on the work that initiated this study [53]. They both assume a constant ambient temperature and Newton's law of cooling. They also assume that the machine has an infinite maximum speed and so every job will always be completed, with the aim to minimise the maximum temperature of the schedule produced. This is clearly not an assumption that entirely holds in reality, but is an important step in studying how DVS can be used to manage temperature. In [7] they show that one of the algorithms from [53] is constant-competitive with respect to temperature, while two of the algorithms from that work are not. They then propose a new algorithm that is constant-competitive with respect to temperature and e-competitive with respect to maximum speed.

One issue that has been given particular attention (and has been briefly mentioned above) is managing *hot spots* within a processor, which is when a particular part of the processor gets very hot, possibly even while the rest of the chip stays cool. This is particularly relevant when multicore processors are considered. A model called HotSpot was proposed in [46],

that has become widely used for simulating temperature in microprocessor systems. This model takes into account that different temperatures occur in different parts of the processor and so is good for simulations that aim to avoid these hot spots occurring.

The HotSpot model has been used by many researchers, for example [42] where they give a temperature aware algorithm for MPEG-2 decoding. They show that in this case temperature awareness does indeed achieve a "thermally safe state" [42] although a performance drop of about 12% in frame rate is suffered in order to achieve this.

In [15] they also consider spacial temperature effects, such as neighbouring processor cores and heat sinks. They use DVS techniques and consideration of heat transfers to create a formulation for assigning tasks and creating a schedule for the NP-hard problem they propose. They show that with these considerations significant reductions in peak temperature can be achieved with an average of around 10 °C and up to around 30 °C in their experiments. They also show that exploiting the amount of slack that jobs have available can be a useful technique in managing temperature.

Using DVS to manage temperature is a technique that is most often used at the hardware level and is often effective in reducing the maximum temperature of a schedule. However many devices have a *temperature threshold* that cannot be exceeded without causing problems, including permanent failures. This motivates a different version of the problem where the aim of an algorithm is to create a schedule that maximises or minimises a property such as job throughput, while ensuring that the temperature remains under a certain threshold. This model was proposed in [19], and was the main motivation

behind this work.

1.7 Models and Notation

We consider several different models throughout this work but they all share some common features and notations that will be described here. Any model specific notation will be detailed in the relevant chapter.

We consider a model where time is split into *discrete time steps*. An algorithm will have access to $m \geq 1$ parallel, identical machines with the same fixed processor speed. At each time step $[t, t+1)$ an algorithm takes a decision for each machine deciding whether to schedule a job on that machine, and if so which job. For simplicity, instead of saying $[t, t+1)$ we will just say that an algorithm schedules a job at time t.

Each job J will have a release time r_J, a processing time p_J, and a heat contribution per time step h_J. The release time of a job will be an integer and before this time has been reached an online algorithm will have no information about the job or even know it exists; an offline algorithm will know the job exists and its properties but will not be able to start the job before this time. The processing time of a job is also an integer and indicates how many time steps a job requires to complete. For the case where all jobs are of equal length we will just refer to all jobs having the same processing time p. The heat contribution per time step of a job is a real number that represents how much heat running that job will generate over a time step, this is explained in more detail later. The work in [51] demonstrates that it is possible to estimate the temperature of a job efficiently and online. In

some models a job J my also have a weight which will be denoted by w_J. This is the value an algorithm gets for successfully completing that job.

For the first objective of maximising throughput every job J will have an integer deadline d_J. If a job is not completed before its deadline by a schedule then that schedule will gain no value for that job. We say J has a *tight deadline* if it must be scheduled immediately upon its release to meet its deadline, i.e. $r_J + p_J = d_J$; we will refer to such a J as a *tight job*.

We will refer to the algorithm that is currently being analysed as $\mathcal{A}$. For any $q \in \{1, 2, ..., m\}$, we use $\mathcal{A}_q$ to denote the schedule produced by the algorithm $\mathcal{A}$ on machine q. The jobs scheduled by $\mathcal{A}_q$ at some time t are denoted by $\mathcal{A}_q(t)$. We consider every machine to have a separate temperature when $m > 1$. For simplicity, we assume that there is no heat transfer between different machines. It has been suggested that this heat transfer is small [51] (and references therein), and to calculate such heat transfer would also require information of the physical layout of the processing units. Therefore we calculate the temperature of each machine independently. We denote the temperature of a given machine q at a time t as $\tau_{q,t}$, although when $m = 1$ we will use τ_t for simplicity.

When we are analysing online algorithms we will refer to the adversary as OPT. We will denote the temperature of a machine's schedule OPT_q at a time t as $\tau'_{q,t}$, or τ'_t when $m = 1$.

Without ambiguity we will also refer to the schedules generated by $\mathcal{A}$ and OPT as $\mathcal{A}$ and OPT respectively, and the value obtained by these schedules as $|\mathcal{A}|$ and $|OPT|$.

We follow the model in [7] and Fourier's Law for modelling cooling; this

states that the rate of cooling is proportional to the difference between the temperature of the machines and their environment, and that the environment's temperature is assumed to be constant. Each system will have a cooling factor $R > 1$. This is how much the temperature of a machine cools in each time step. To calculate the temperature of a machine q after executing one time step of a job J at time t we have $\tau_{q,t+1} = (\tau_{q,t} + h_J)/R$. If the machine is idle for a time step we calculate the temperature as if a job J is being scheduled but with $h_J = 0$. We use a parameter R to model cooling because the cooling of a system can vary significantly, as will be explained in more detail in Chapter 2.

Each machine will have a *thermal threshold* T which is the maximum allowed temperature of any machine at the end of any time step (after the cooling has taken place). We set the intial temperature to 0 and the thermal threshold to 1 without loss of generality. In some cases, as a form of resource augmentation, we may allow an online algorithm to have an increased thermal threshold. This will be described in more detail in Chapter 5.

We describe a job J as being *pending* for an algorithm at some time t if the job has been released but not expired, i.e. $r_J \leq t \leq d_J - p'_J$ where p'_J is the remaining processing time of J at time t, and J has not been completed by the algorithm before t.

We describe a job J as *admissible* for an algorithm at time t if it is pending at t and not too hot to be executed on some machine. As a job J takes p_J steps to complete, the temperature at any point during these p_J steps must not exceed the threshold. It is easy to show that as long as the temperature after executing for p_J consecutive time steps is within the threshold, then

that must also be the case for any intermediate time steps.

As the thermal threshold is 1 and the cooling factor per time step is R, it must be that $h_J \leq R$ for all jobs, otherwise it is impossible to run one time step of the job without exceeding the thermal threshold. We call the model where jobs are allowed to have this maximum job heat the *full heat* model. In some models we will limit the maximum job heat further so that $\forall J : h_J \leq R - \epsilon$ for some $\epsilon > 0$. We call this model the *bounded heat* model.

1.8 My Contribution

The contributions that have been made in this thesis will now be detailed. Note that parts of this work have already been published in journals and conference proceedings [11–13].

There are two broad optimisation objectives that have been studied for assessing the quality of algorithms during this work. The first objective was to maximise throughput. The work from [19] on the throughput of unit length jobs in temperature aware systems was extended to show results for all cooling factors and multiple machines. In Chapter 2 we first show some results for the model where jobs are unweighted and unit length.

- We show an upper bound for a class of algorithms called 'reasonable algorithms', for all $R > 1$ and $m \geq 1$. We also give a lower bound that shows that reasonable algorithms are in fact optimal for all values of R when $m = 1$. For $m \geq 2$ we give some weaker lower bounds.

The results show how the competitiveness depends on R: specifically, it increases as R gets smaller, and tends to infinity when R tends to 1.

In Chapter 2 we also show some results for the weighted unit length case for multiple machines. These extend work done by the other authors in [11]. The bounds in this work are given in terms of W, where this is the ratio of the maximum to minimum job weights.

- We show a lower bound for deterministic algorithms on multiple machines of $\Omega((mW)^{1/m})$. The upper bound for a deterministic algorithm is $O(mW^{1/m})$, therefore for constant m this gives a tight bound of $\Theta(W^{1/m})$. We give an upper bound for the randomised single machine fully online case of $O(\log^{1+\epsilon} W)$ which is only slightly larger than the semi-online case upper bound of $O(\log W)$. We also show that with both multiple machines and randomisation, the same bounds as in the single machine case hold.

In Chapter 3 we then consider the single machine case where jobs are unweighted and all have equal length.

- We initially consider the non-preemptive case, and show a lower bound for all deterministic algorithms. We then show that the algorithm Coolest First gives a matching upper bound and give a slightly weaker upper bound for the more general class of non-idling algorithms.

  Next we consider preemption and show several lower bound results. We give lower bounds for the preemptive restart or resume models that are slightly lower than those of the non-preemptive case. This shows that preemption is not going to help a lot in reducing the competitiveness. Moreover, we give a lower bound that shows that preemption with restarts does not have any advantage at all over the non-preemptive

case when $R^p > R + 1$ where p is the length of each job. The exact bounds depend on the values of R and p, and will be stated in the theorems, but the charts in Figure 1.1 give an idea on how the bounds for the different models change depending on the values.

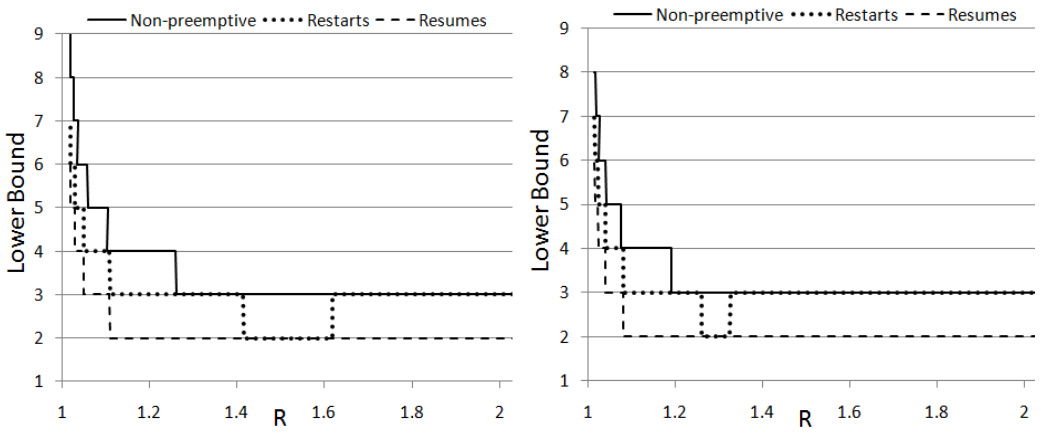

Figure 1.1: Bounds for the different models when (left) $p = 2$, (right) $p = 3$.

In Chapter 4 we consider the case where jobs have variable lengths on a single machine.

- First we consider the case where jobs are unweighted. We show an upper bound for non-idling algorithms that is dependant on R and L, where L is the length of the longest job. We then show a lower bound for the non-preemptive case that shows this is almost optimal for all R and L. We also show some lower bounds for both types of preemption and show that the non-preemptive algorithm is optimal within a constant factor for any fixed R and all L.

- Next we consider the case where the value of a job is related to its length, i.e. the value an algorithm gets for scheduling a job J is p_J.

We show an upper bound for the Longest First algorithm. We then show a lower bound that shows that this is optimal for small enough R , and almost optimal when R is large. We also note that these results still hold when $w_J = p_J^\alpha$ for some $\alpha \geq 1$.

The second objective we considered was to minimise the flow time of a schedule. In Chapter 5 we initially consider the case where the objective is to minimise the average/total flow time of a schedule produced by an algorithm. We give some results for the offline case.

- We extend the proof of [19] to show that when $R = 2$ and the heat of a job is bounded to $1 + \delta$ for any $\delta > 0$, then the problem of scheduling them optimally is still NP-hard. We modify the proof of [37] to show that it is NP-hard to get better than $O(n^{1/2-\epsilon})$ approximation of the total flow time of a schedule, where n is the number of jobs released.

  We also show an upper bound of 2.618 for an approximation algorithm for the offline case where all jobs have release time 0 and $R \geq 2$. This algorithm builds on the algorithm for minimising the makespan of a schedule as in [6].

When the maximum heat of a job is allowed to be exactly R then it can be trivially shown that no algorithm can give a bounded competitive ratio in the online case. If the maximum heat of a job is restricted to be no hotter than $R - 1$ then it can be easily shown that any algorithm is 1-competitive. Therefore we consider the case where the maximum allowed heat of a job is $R - \epsilon$ for some $0 < \epsilon < 1$.

- We show a lower bound for all deterministic online algorithms that increases as ϵ decreases, increasing to infinity as ϵ approaches 0. We also show a lower bound of 2 for all deterministic algorithms for all values of ϵ.

  We show that non-idling algorithms will have a competitive ratio of at least $\Omega(n)$ for all $0 < \epsilon < 1$.

Next we allow the online algorithm to have a larger thermal threshold of $1 + \epsilon$ while the optimal algorithm still has a threshold of 1 as a form of resource augmentation. The maximum allowed temperature of a job is not limited any lower than R (as this is the hottest that OPT can schedule). Note that when $\epsilon \geq 1/(R-1)$ a 1-competitive upper bound is trivial because it is always possible to schedule any job at any time step.

- We first show a lower bound that again increases as ϵ decreases, increasing to infinity as ϵ approaches 0. We then show a counter example for the algorithm coolest first that shows it will have a competitive ratio of at least $\Omega(n)$ for all possible ϵ values, and we then show that non-idling algorithms have a competitive ratio of at least $\Omega(n)$ as long as $\epsilon < 1/R^2$. This means that when $R = 2$ and $\epsilon < 1/4$, there cannot be a 1-competitive algorithm.

  We then show an upper bound on the Hottest First algorithm that shows it is 1-competitive as long as $\epsilon \geq \frac{R^2+R+1}{(R-1)(R+1)^2}$. This means that when $R = 2$ and $\epsilon \geq 7/9$, Hottest First is 1-competitive.

All of the results in Chapter 5 are shown primarily for the case of minimising total flow time, however in the final section we give some results for the

objective of minimising the maximum flow time of any job in the schedule.

- We show that Coolest and Hottest First have trivially unbounded competitive ratios in the bounded heat and higher thermal threshold models for all possible ϵ values. We also show that two variants of FIFO have unbounded competitive ratios for both models.

In Chapter 6 we provide a summary of the results in this thesis and indicate some potential directions for future work.

Chapter 2

Unit Length Throughput

We now examine the problem of maximising the throughput of unit length jobs. This work extends the simple model defined in [19], where they used a single machine, unweighted jobs and a fixed cooling factor, $R = 2$. All jobs in this model are unit length, with each job taking exactly one time step to complete i.e. $\forall J : p_J = 1$. We consider both the weighted and unweighted versions of this problem, for both single and multiple machines.

2.1 Motivation

The results we give in this chapter are for all values of $R > 1$. This is because the main motivation of modelling the problem using unit-length jobs is to represent the job slices given to the processor by the operating system [19]. As such the actual cooling factor R relates to the length of this time quantum and the 'RC constant' of the processing unit (a thermal property related to how quickly the unit cools). Different systems appear to have very different

values for these parameters (see e.g. [39]) and it is therefore important that we can design and analyse algorithms for different values of R.

We also extend the work to consider multiple machines. Multicore processors have become increasingly popular over recent years. One of the main motivations of using multicore processors is that it is a useful tool in coping with the heat a processor generates (see e.g. [28] and references therein). A result of power usage being a convex function of processor speed is that it is more power (and therefore temperature) efficient to run several processors at a lower speed, than it is to run one processor at a high speed and still carry out the same amount of work.

In addition to this, the use of multiple machines means that algorithms have more decisions to make when scheduling jobs, which can increase the scope for increasing the quality (or if the decisions are made poorly, then decreasing the quality) of the schedules produced. As one of the main motivations for using multiple machines was in an effort to reduce heat issues, it follows that it is important to maximise the extra capabilities that multicore systems provide. This means that the design and analysis of temperature aware algorithms for multicore systems is needed in order that these systems can reach their full potential.

2.2 Previous Results

Without temperature it is trivial to find optimal algorithms for the unweighted version of the maximum throughput problem on both single and multiple machines, however the weighted version of this problem has at-

tracted a considerable amount of interest. The current best known deterministic bounds for the problem without temperature on a single machine are an upper bound of 1.828 [25] and a lower bound of 1.618 [2, 18]. Randomisation helps with this problem against an oblivious adversary and as such the best known randomised upper bound is 1.582 [17] and the lower bound is 1.25 [18].

The problem of maximising throughput and allowing the use of multiple machines with weights but without temperature has also been researched. There are different models, but we will only consider here the case where an algorithm's machines are all identical. For this case the best known algorithm is $((1 - (\frac{m}{m+1})^m)^{-1})$-competitive and there is a lower bound of 1.25 [17].

There has also been some work done that uses the temperature model as given in this work, which was proposed in [19]. For the single machine case without job weights and when $R = 2$ they demonstrated that algorithms that can be classified as 'reasonable' are all 2-competitive and that this is optimal. They also showed that computing the optimal schedule offline is NP-hard, even in the special case where all jobs have the same deadline and are released at time 0.

2.3 Unweighted Jobs

We first give some results for the version of the problem where jobs are unweighted. Using standard 3-field notation this problem can be described as $P|\textit{online-}r_i, h_i, p_i = 1|\sum U_i$.

We now analyse a class of algorithms called *reasonable algorithms* that

was originally defined in [19] for the single machine case. The definition is repeated here for completeness.

They first define the notion of a job J_1 *dominating* a job J_2 if $d_{J_1} \leq d_{J_2}$ and $h_{J_1} \leq h_{J_2}$. If at least one of the inequalities is strict then we say that J_1 *strictly dominates* J_2. We now use this definition to define a notion of reasonable algorithms, extended to the multiple machine case.

Definition 2.1. *An online algorithm is called* reasonable *if, at any time step t, it schedules jobs such that (i) a machine is left idle only if there is no job admissible on that machine, or that all such admissible jobs are already scheduled on other machines at time t, and (ii) any job scheduled on a machine is not dominated by some other pending jobs not scheduled on any machine at time t.*

Note that when considering the multiple machine case it may be that, at a certain time step, a pending job J is admissible on an idle machine q or dominates the job scheduled on machine q, but J is not scheduled on machine q; this is possible if J is scheduled on another machine q' instead, and this does not violate the definition of reasonable algorithms.

Throughout this chapter we give bounds in terms of the value U, which is the largest integer $u \geq 1$ such that Inequality (2.1) holds, for any fixed $R > 1$.

$$R^u < (u+1)R - u. \tag{2.1}$$

2.3.1 Reasonable Algorithms Upper Bound

We now show an upper bound on the competitiveness of all reasonable algorithms. The general strategy for the competitive analysis is to map, or charge, the completed jobs in OPT to those in $\mathcal{A}$, such that (1) all jobs in OPT are charged to some jobs in $\mathcal{A}$, and (2) each job in $\mathcal{A}$ is being charged to by a bounded number of jobs in OPT.

Before describing the charging scheme we define the notion of a *relative heating step* as originally defined in [19].

Definition 2.2. *Whenever, at some time u and for some machine q, $\mathcal{A}_q$ schedules a job J that is strictly hotter than a job K scheduled by OPT_q (with an idle time step in OPT being treated as though a job K with $h_K = 0$ is being executed) we call this a* relative-heating step.

Note that whenever $\tau_{q,u} > \tau'_{q,u}$ for some time u and machine q a relative-heating step must have occured before time u on machine q.

We use a charging scheme that is based on the one used in [19]. For a job J that has been scheduled by OPT_q at time u the charge of J is as follows:

Type-1 Charge: If $\mathcal{A}_q$ has also scheduled a job K at time u then charge J to K.

Type-2 Charge: If $\mathcal{A}_q$ is idle and hotter than OPT_q at time u but not at time $u+1$, that is $\tau_{q,u} > \tau'_{q,u}$ and $\tau_{q,u+1} \leq \tau'_{q,u+1}$, then there must have been a relative-heating step on machine q before time u. Job J is charged to the job K executed by $\mathcal{A}_q$ at the last relative heating step before u.

Type-3 Charges: These charges are for the case when $\mathcal{A}_q$ is idle at time u and either $\mathcal{A}_q$ is cooler or the same as OPT_q at u, or $\mathcal{A}_q$ is hotter than OPT_q at time $u+1$, more formally $\tau_{q,u} \leq \tau'_{q,u}$ or $\tau_{q,u+1} > \tau'_{q,u+1}$. We divide Type-3 charges into two sub-types:

Type-3a: Type-3a charges are used when $\tau_{q,u} + h_J \leq R$. This means J must be scheduled by some $\mathcal{A}_{q'}$ (where q' may or may not be the same as q) at some time $t \leq u$, otherwise $\mathcal{A}_q$ would schedule J instead of staying idle, by definition of the algorithm.

To find a job to charge J to we construct a chain of jobs $J, J', ..., J^*$ like that constructed in [19]. The chain will be uniquely defined by J. If at time t, $OPT_{q'}$ is idle or schedules a job J' such that $h_{J'} \geq h_J$ then set $J^* = J$, that is the last job in the chain is the copy of the job J in $\mathcal{A}_{q'}$'s schedule. Otherwise $OPT_{q'}$ schedules some job J' at t such that $h_{J'} < h_J$. It will be shown in Lemma 2.3 that in this case J' must be scheduled by some $\mathcal{A}_{q''}$ at some time $t' \leq u$. Job J' is then added to the chain.

Then we repeat this process: if at time t' $OPT_{q''}$ is idle, or schedules a job J'' such that $h_{J''} \geq h_{J'}$, then we end the chain and have that $J^* = J'$. Otherwise J'' must be scheduled by some $\mathcal{A}_{q'''}$ at some time $t'' \leq u$, and we add J'' to the chain and the process continues. This process must end at some point as it deals with strictly cooler and cooler jobs and there are only a finite number of jobs. Job J is then charged to J^*.

Type-3b: Type-3b charges are used when $\tau_{q,u} + h_J > R$, i.e. J would

be too hot to be scheduled on machine q (if it is still pending). It must be the case that $\tau_{q,u+1} > \tau'_{q,u+1}$, because with the other Type-3 case of $\tau_{q,u} \leq \tau'_{q,u}$, as J is scheduled on OPT_q, it must be that $\tau'_{q,u} + h_J \leq R$, so $\tau_{q,u} + h_J \leq R$ too, contradicting the condition of Type-3b charge. It also follows that $\tau_{q,u} > \tau'_{q,u}$ as $\mathcal{A}_q$ is idle at u so it cannot change from cooler than or equal to OPT_q to hotter than OPT_q.

With a Type-3b charge job J is charged to the latest job already scheduled by $\mathcal{A}_q$ before time u. Such a job must exist because there must have been at least one relative heating step on machine q in order for $\mathcal{A}_q$ to be hotter than OPT_q at time u.

Although much of the charging scheme is the same as the one in [19], there are important differences that arise from the fact that we may have $R < 2$, and new techniques are needed to handle them. In particular, the correctness of the Type-3a charges is not straightforward and is established in the following lemma.

Lemma 2.3. *Consider a Type-3a chain $J_0, J_1, ..., J_{n-1}$ where $J_0 = OPT_q(t)$ is the job at the beginning of the chain that generates the charge. Let t_i be the time J_i is scheduled in $\mathcal{A}$, q' be the machine that schedules J_{n-1} in $\mathcal{A}$, i.e. $J_{n-1} = \mathcal{A}_{q'}(t_{n-1})$, and let $J_n = OPT_{q'}(t_{n-1})$. If $h_{J_n} < h_{J_{n-1}}$, then J_n must be scheduled by some machine in $\mathcal{A}$ on or before t (so the chain continues).*

Proof. We prove by contradiction. For simplicity denote r_{J_i}, d_{J_i} and h_{J_i} by r_i, d_i and h_i respectively. Suppose J_n is not scheduled by any machine in $\mathcal{A}$ on or before t, we show that:

(1) $d_n > d_0$.

(2) $\tau_{q,t} + h_0 > R$.

The second statement contradicts the definition of a Type-3a charge.
(1) We prove the stronger claim that $d_n > d_i$ for all $0 \leq i < n$. The claim is proven by induction along the chain. By the definition of a Type-3a chain and the assumption in the lemma we have that $h_n < h_{n-1} < ... < h_1 < h_0$.

First we show that $d_n > d_{n-1}$. We have that $h_n < h_{n-1}$ and that J_n is not scheduled on or before t by $\mathcal{A}$. This means that if $d_n \leq d_{n-1}$ then $\mathcal{A}$ would not have chosen to schedule J_{n-1} as J_{n-1} would be strictly dominated by J_n.

Now assume $d_n > d_{i'}$ for all $i \leq i' \leq n-1$. We show $d_n > d_{i-1}$. For each step in the chain we have some $\mathcal{A}_r$ that schedules J_i at time t_i and at the same time step OPT_r schedules J_{i+1}, that is $J_i = \mathcal{A}_r(t_i)$ and $J_{i+1} = OPT_r(t_i)$. The job J_i is in turn scheduled by some OPT_s at time t_{i-1} (which may be equal to t_i) and $\mathcal{A}_s$ schedules a job J_{i-1} at the same time step, as the previous step in the chain. We consider three cases.

Case 1: $t_{i-1} < t_i$, as in Figure 2.1. (In Figures 2.1 - 2.3, all jobs may appear to be on the same machine but it is only for ease of presentation; they may actually be scheduled on different machines.) Job J_i is pending at t_{i-1} in $\mathcal{A}$ because it must have been released, as it was scheduled by OPT_s at this time, but it is only scheduled by $\mathcal{A}_r$ at a later time u. Since $h_i < h_{i-1}$, it must be that $d_i > d_{i-1}$, otherwise J_i strictly dominates J_{i-1} and so $\mathcal{A}_s$ would not have scheduled J_{i-1} if this were the case. By induction hypothesis $d_n > d_i$, and hence $d_n > d_{i-1}$.

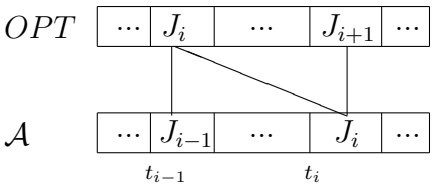

Figure 2.1: $t_{i-1} < t_i$

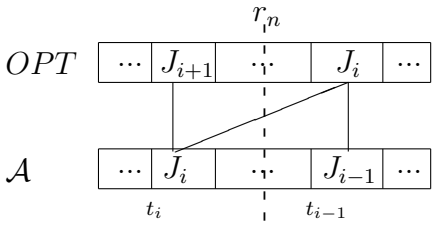

Figure 2.2: $t_{i-1} \geq t_i$ and $t_{i-1} \geq r_n$

Case 2: $t_{i-1} \geq t_i$ and $t_{i-1} \geq r_n$, as in Figure 2.2. This means that J_n must be pending in $\mathcal{A}$ at t_{i-1}. We also know that $h_n < h_{i-1}$. This means that $d_n > d_{i-1}$ otherwise job J_{i-1} would be strictly dominated by J_n and so $\mathcal{A}$ would not have scheduled J_{i-1}.

Case 3: $t_{i-1} \geq t_i$ and $t_{i-1} < r_n$, as in Figure 2.3. For this case to occur there must exist a job J_k such that $k > i$, that is J_k is added to the chain later than J_i, and J_k is scheduled by some $\mathcal{A}_x$ at time $t_k > t_{i-1}$ but is scheduled by some OPT_y at time $t_{k-1} \leq t_{i-1}$. If this is not the case it would be impossible for the chain to have continued and ended at time $t_{n-1} \geq r_n$.

As $k > i$ this means $h_{i-1} > h_i > h_k$. The job J_k must be pending in $\mathcal{A}$ at time t_{i-1}, as it has already been scheduled by OPT, but it has not yet been scheduled on any machine in $\mathcal{A}$, so it must be that $d_k > d_{i-1}$ otherwise J_k would have strictly dominated J_{i-1} and so $\mathcal{A}_s$ would not schedule J_{i-1} at t_{i-1}. By induction we can assume that $d_n > d_k$ so we can also conclude that $d_n > d_{i-1}$.

(2) As $d_n > d_0$ (by (1)) and J_n has not been scheduled on or before t, it must be too hot to be scheduled by $\mathcal{A}_q$ at time t, i.e. $\tau_{q,t} + h_n > R$, and as we know that $h_0 > h_n$ it follows that $\tau_{q,t} + h_0 > R$ as well. □

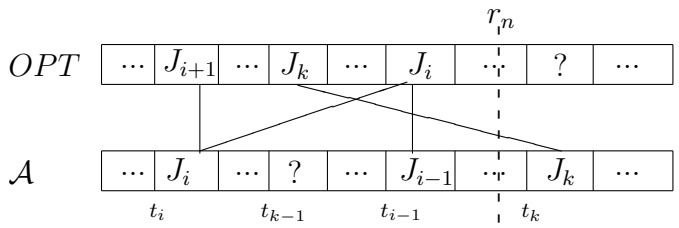

Figure 2.3: $t_{i-1} \geq t_i$ and $t_{i-1} < r_n$

The next two lemmas help bound the number of Type-3b charges to a job.

Lemma 2.4. *If $\mathcal{A}_q$ remains idle at time t then $\tau_{q,t} - \tau'_{q,t} > \tau_{q,t+1} - \tau'_{q,t+1}$.*

Proof. Let $\alpha = \tau_{q,t} - \tau'_{q,t}$. Suppose OPT_q schedules a job J at time t (if OPT_q remains idle then treat it as executing a job with $h_J = 0$). Then $\tau_{q,t+1} - \tau'_{q,t+1} = \frac{\tau_{q,t}}{R} - \frac{\tau'_{q,t}+h_J}{R} = \frac{\alpha - h_J}{R} < \alpha$, since $R > 1$ and $h_J \geq 0$. □

Before we proceed with the rest of the proof we define a notion of *OPT-only jobs.*

Definition 2.5. *We refer to a job as an* OPT-only job *if it is started by OPT_q at some time t when $\mathcal{A}_q$ is idle, and at time t the job is too hot to be scheduled by $\mathcal{A}_q$.*

Note that for an OPT-only job to exist at some time t for some machine q it must be that $\tau_{q,t} > \tau'_{q,t}$.

Lemma 2.6. *For any machine q, OPT_q can schedule at most $U - 1$ OPT-only jobs in a time interval where $\mathcal{A}_q$ stays idle when, after each OPT-only job has been completed by OPT_q, the temperature of OPT_q is still lower than that of $\mathcal{A}_q$.*

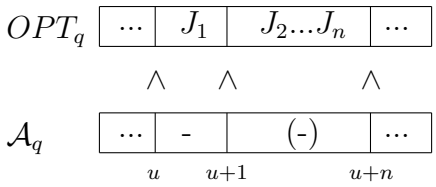

Figure 2.4: The scenario with $\mathcal{A}$ remaining idle and OPT scheduling n jobs. The vertical inequality signs between the schedules show the relation between the temperatures.

Proof. Suppose there are n jobs $J_1, J_2, ..., J_n$ in OPT_q that are scheduled consecutively by OPT_q but are too hot to be scheduled by $\mathcal{A}_q$; that $\mathcal{A}_q$ remains idle until all n jobs have been scheduled; and that after each job has been completed by OPT_q, the temperature of OPT_q is less than that of $\mathcal{A}_q$. Suppose J_1 is scheduled at time u. See Figure 2.4. We will assume that between any of the jobs $J_1, ..., J_n$ in OPT_q's schedule, there are no idle time slots or jobs that are not OPT-only. It will be shown that the inclusion of these will not make OPT_q able to schedule more OPT-only jobs.

Initially it must be that $\tau_{q,u} \leq 1$. As $\mathcal{A}_q$ stays idle for all n time steps we also have $\tau_{q,u+i} \leq \frac{1}{R^i}$ for $i = 1, ..., n$. By the definition of OPT-only jobs all of the jobs $J_1, ..., J_n$ are too hot to be scheduled by $\mathcal{A}_q$. Therefore, each job J_i, for every $1 \leq i \leq n$, has a heat contribution

$$h_{J_i} > R - \tau_{q,u+i-1} \geq R - \frac{1}{R^{i-1}}. \tag{2.2}$$

We now consider the temperature of OPT_q after executing these jobs. Initially $\tau'_{q,u} \geq 0$. After executing i jobs, $1 \leq i \leq n$, the temperature is given

by the recursive formula $\tau'_{q,u+i} = \frac{\tau'_{q,u+i-1}+h_{J_i}}{R}$, which can be solved to give

$$\tau'_{q,u+n} \geq \frac{R(1-R^{-n})}{R-1} - \frac{n}{R^n}. \tag{2.3}$$

By the definition of the lemma we must have that after the n jobs have been completed, $\tau'_{q,u+n} < \tau_{q,u+n}$. Together with Inequalities (2.2) and (2.3), this gives

$$\frac{R(1-R^{-n})}{R-1} - \frac{n}{R^n} < \frac{1}{R^n}.$$

This is equivalent to

$$R^{n+1} < (n+2)R - (n+1).$$

From the definition of U it follows that $U-1$ is the maximum number of OPT-only jobs that can be scheduled before $\mathcal{A}$ starts another job .

We now return to the assumption of OPT_q scheduling all of the n jobs consecutively. Our aim is to upper-bound the number of jobs that can be scheduled by OPT_q but are too hot to be scheduled by $\mathcal{A}_q$ in a time period when $\mathcal{A}_q$ remains idle but is still hotter than OPT_q. Suppose $J_1, ..., J_n$ are not consecutive then there are time steps in-between where OPT_q is idle or schedules a job that is not too hot to be scheduled by $\mathcal{A}_q$. By Lemma 2.4, such time steps will reduce the difference in temperature between OPT_q and $\mathcal{A}_q$, thus will not help OPT_q schedule more hot jobs. □

Theorem 2.7. *Any job scheduled by $\mathcal{A}_q$ for any q can receive at most $U+1$ charges, therefore any reasonable algorithm $\mathcal{A}$ is $(U+1)$-competitive for any fixed $R > 1$.*

Proof. It is clear that each job in $\mathcal{A}_q$ can receive at most one Type-1 charge. In addition to this each job in $\mathcal{A}_q$ receives at most one Type-2 charge. This is because in between any two time steps that satisfy Type-2 conditions there must be a relative-heating step, so Type-2 charges will be assigned to distinct relative heating steps. A job can also receive at most one Type-3a charge because the chains formed are disjoint and uniquely defined by the original job that generated the Type-3a charge. If a job J receives a Type-1 charge it cannot receive both a Type-2 charge and a Type-3a charge. This is because a Type-3a charge cannot be made to a job at a relative heating step that also receives a Type-1 charge, otherwise the chain would continue further as shown by Lemma 2.3. This means that any job can receive at most 2 charges from the set {Type-1, Type-2, Type-3a}.

Next we upper-bound the number of Type-3b charges a job can receive. Type-3b charges from a job in OPT_q at time u are made to the job most recently scheduled by $\mathcal{A}_q$ before u. Thus, any jobs making Type-3b charges to a job J in $\mathcal{A}_q$ must come from an idle time interval in $\mathcal{A}_q$ immediately after J. Moreover, it was already shown that $\mathcal{A}_q$ is hotter than OPT_q before and after the scheduling of such Type-3b jobs in OPT_q. Thus they satisfy the conditions in Lemma 2.6, and so the number of Type-3b charges to a job is at most $U - 1$, where U is the largest integer $n \geq 0$ satisfying Inequality (2.1). □

Table 2.1 shows the competitive ratio of any reasonable algorithm for different values of R. It can be seen that the competitive ratio increases as R decreases. The competitive ratio is still fairly reasonable for moderate values

of R, but goes to infinity as R gets very close to 1.

R	**Upper Bound**
$R \geq 2$	2
$2 > R > \frac{-1+\sqrt{13}}{2} \approx 1.30278$	3
$1.30278 \geq R \gtrsim 1.15091$	4
$1.15091 \gtrsim R \gtrsim 1.09128$	5
$1.09128 \gtrsim R \gtrsim 1.0614$	6
$1.0614 \gtrsim R \gtrsim 1.04421$	7
$1.04421 \gtrsim R \gtrsim 1.03339$	8
$1.03339 \gtrsim R \gtrsim 1.02612$	9
$1.02612 \gtrsim R \gtrsim 1.02100$	10
1.01	15
1.001	45
1.0001	142

Table 2.1: Table of upper bounds for reasonable algorithms

2.3.2 Single Machine Lower Bounds

In this section we show a lower bound for all deterministic algorithms when $m = 1$, showing that in the single machine case reasonable algorithms are in fact optimal for all $R > 1$. We also show that the simple algorithm Hottest First performs worse than reasonable algorithms.

Theorem 2.8. *When $m = 1$, any deterministic algorithm has a competitive ratio of at least $U + 1$ for any fixed $R > 1$.*

Proof. Fix some deterministic algorithm $\mathcal{A}$. At time 0 release a job J that has a very large deadline and a heat contribution of just below R, that is $d_J = 2D + U + 1$ for a large D and $h_J = R - \epsilon$ for a sufficiently small $\epsilon > 0$. D must be large enough so that OPT's temperature will have cooled down from 1 to ϵ after D steps, i.e. $R^D \geq \frac{1}{\epsilon}$. If $\mathcal{A}$ never schedules J then OPT

does and $\mathcal{A}$ has an unbounded competitive ratio. Otherwise $\mathcal{A}$ schedules J at some time u. We consider two cases:

Case 1: $u < D$. In this case OPT chooses not to start J yet. At time $u+i$ for $i = 1, 2, ...$ the temperature of $\mathcal{A}$ is given by $\tau_{u+i} = \frac{R-\epsilon}{R^i}$. At each such time $u+i$, as long as $\tau_{u+i} > \tau'_{u+i}$, the adversary releases a tight job J_i with $d_{J_i} = u+i+1$ and $h_{J_i} = R - \frac{R-\epsilon}{R^i} + \delta$ for a very small $\delta > 0$. $\mathcal{A}$ will be (just) too hot to schedule the job due to scheduling job J while OPT can and will schedule each of them i.e. they are OPT-only jobs. We will prove that this stops after U steps. OPT will then remain idle for D time steps until it is cool enough to schedule J. Since $u + 1 + U + D < 2D + U + 1$, OPT can still finish J before its deadline.

Case 2: $u \geq D$. In this case OPT will schedule J to start at time 0. Since $u \geq D$, at time $u+1$ OPT's temperature will be less than ϵ. Similar to the previous case, at each time $u+i$ where $i = 1, 2, ...$ we release a tight job where $d_{J_i} = u+i+1$ and $h_{J_i} = R - \frac{R-\epsilon}{R^i} + \delta$. $\mathcal{A}$ will be just too hot to run any of these jobs, and this continues as long as $\tau_{u+i} > \tau'_{u+i}$, so OPT can finish all these jobs.

In both cases OPT will schedule $U+1$ jobs (the U tight jobs that are generated in successive time steps plus J) whereas $\mathcal{A}$ will only ever be able to schedule J. Thus the competitive ratio is $U+1$.

The temperature of OPT at time $u+i$ is given by

$$\tau'_{u+1} \leq \epsilon, \quad \tau'_{u+i} = \frac{\tau'_{u+i-1} + R - \frac{R-\epsilon}{R^{i-1}} + \delta}{R} \text{ for } i > 1.$$

This can be solved so we have

$$\tau'_{u+i} \leq \frac{R}{R-1}\left(1-\frac{1}{R^{i-1}}\right)-\frac{i-1}{R^{i-1}}+\left(\frac{\epsilon}{R^{i-1}}+\frac{1}{R-1}\left(1-\frac{1}{R^{i-1}}\right)\left(\frac{\epsilon}{R^{i-1}}+\delta\right)\right).$$

The sequence of jobs continues as long as $\tau'_{u+i} < \tau_{u+i}$. Hence U is the largest value of n such that $\tau'_{u+n} < \tau_{u+n}$. This gives

$$\frac{R}{R-1}\left(1-\frac{1}{R^{n-1}}\right)-\frac{n-1}{R^{n-1}}+\left(\frac{\epsilon}{R^{n-1}}+\frac{1}{R-1}\left(1-\frac{1}{R^{n-1}}\right)\left(\frac{\epsilon}{R^{n-1}}+\delta\right)\right) < \frac{1}{R^{n-1}}-\frac{\epsilon}{R^n}. \quad (2.4)$$

If we ignore all terms involving ϵ and δ, this is equivalent to

$$R^n < (n+1)R - n$$

As long as the above inequality holds strictly, we can always choose sufficiently small ϵ and δ so that Inequality (2.4) still holds. Thus the sequence indeed stops after U steps with U given by Inequality (2.1). □

Therefore by comparing Theorems 2.7 and 2.8 the lower bound matches the upper bound, and thus reasonable algorithms are optimal for all ranges of values of R when $m = 1$.

We now show a non-optimal lower bound on the competitiveness of Hottest First. Hottest First schedules the hottest admissible job at each time step that there is an admissible job available. Note that it is not a reasonable algorithm as it may schedule a job that is strictly dominated by another.

Theorem 2.9. *Hottest First is at least* $(U+2)$*-competitive for any* $R > 1$ *when* $m = 1$.

Proof. At time 0 we release a job J with a heat contribution of $R - \epsilon$ for a sufficiently small $\epsilon > 0$ and a deadline of $2 + U + D$ for a large D. D must be large enough so that the temperature of OPT can cool down from 1 to ϵ after D idle time steps. We also release a job Z with a heat contribution of 0 and a tight deadline of 1. $\mathcal{A}$ will start J at time 0.

At each time step i for every $1 \leq i \leq U$ we release a job K_i with a tight deadline of $1 + i$ and a heat contribution of $R - \frac{R-\epsilon}{R^i} + \delta$ for a very small $\delta > 0$. $\mathcal{A}$ will be just too hot to schedule the jobs due to scheduling $\mathcal{A}$, making each K_i OPT-only. It is clear from the proof of Theorem 2.8 that this can continue for all U time steps. We won't release any more jobs so $\mathcal{A}$ will only be able to schedule J.

OPT will schedule Z at time 0, then each of the jobs K_i, for $1 \leq i \leq U$, as soon as it is released. At time $1 + U$ the temperature of OPT will be at most 1. This means at $1 + U + D$ OPT will have cooled down to ϵ and so be cool enough to schedule J before its deadline. OPT will therefore have scheduled $U + 2$ jobs compared to the 1 job scheduled by $\mathcal{A}$, concluding the proof. □

2.3.3 Multiple Machines Lower Bound

The lower bound from the previous section does not work for any larger number of machines. We now show some weaker lower bounds for any number of machines.

Theorem 2.10. *Any deterministic algorithm has a competitive ratio of at least* $2 - \frac{1}{U}$ *for any* $R > 1$ *and any number of machines* m*.*

Proof. Fix any deterministic algorithm $\mathcal{A}$. At time 0 release a set $\mathcal{J}$ of m tight jobs each with the maximum heat contribution, that is for every $J_i \in \mathcal{J}$, $d_{J_i} = 1$ and $h_{J_i} = R$. Suppose $\mathcal{A}$ schedules x of the $|\mathcal{J}|$ jobs at time 0. We set a threshold of λm for some $0 \leq \lambda \leq 1$ to be fixed later. This gives the following two possibilities:

Case 1: $x < \lambda m$. OPT can schedule all the m jobs and no further jobs are released. This gives a competitive ratio of $\frac{m}{x}$ which is at least $\frac{1}{\lambda}$.

Case 2: $x \geq \lambda m$. Starting at time 1 we release m copies of the U tight jobs as in the single machine lower bound in Theorem 2.8. The value for U is given by Inequality (2.1) as before. Only on each of the $m - x$ machines that do not start a J-job can these U jobs be scheduled, as any machine that schedules a J-job will remain too hot to schedule any of the U jobs. OPT will remain idle for the first time step and be able to schedule U jobs on each of its m machines. This gives a competitive ratio of $\frac{mU}{x+(m-x)U} = \frac{mU}{x(1-U)+mU} \geq \frac{U}{\lambda+(1-\lambda)U}$.

We must now find the best value for λ. As the ratio $\frac{1}{\lambda}$ decreases with an increase in λ, while $\frac{U}{\lambda+(1-\lambda)U}$ increases with λ, to find the optimal λ value we need to find the value for λ where these two ratios are equal for a given U. This gives $\lambda = \frac{U}{2U-1}$, and using this λ we get a lower bound of $\frac{1}{\lambda} = 2 - \frac{1}{U}$. □

The lower bound in Theorem 2.10 becomes 1 if $R \geq 2$. Below we give different lower bounds which are better for the case $R \geq 2$. The lower

bounds use jobs with certain heat contribution properties as stated in the lemma below.

Lemma 2.11. *Consider the case of a single machine with an initial temperature of 0. There exists heat contributions for jobs J and K such that: K cannot be scheduled immediately after J but can be scheduled with one idle time step in-between; while J can be scheduled immediately after K.*

Proof. The heat contributions of J and K have to satisfy the following

$$\frac{h_J}{R^2} + \frac{h_K}{R} > 1, \quad \frac{h_K}{R^2} + \frac{h_J}{R} \leq 1, \quad \frac{h_J}{R^3} + \frac{h_K}{R} \leq 1$$

$$0 \leq h_J \leq R, \quad 0 \leq h_K \leq R.$$

Set $h_J = \frac{R^2}{R+1} - \epsilon$ and $h_K = \frac{R^2}{R+1} + \epsilon$ where $\epsilon > 0$ is a sufficiently small positive real number. Then we can verify that all conditions are indeed satisfied (for sufficiently small ϵ)

$$\frac{h_J}{R^2} + \frac{h_K}{R} = \frac{1}{R+1} - \frac{\epsilon}{R^2} + \frac{R}{R+1} + \frac{\epsilon}{R} = 1 + \epsilon\left(\frac{1}{R} - \frac{1}{R^2}\right) > 1,$$

$$\frac{h_K}{R^2} + \frac{h_J}{R} = \frac{1}{R+1} + \frac{\epsilon}{R^2} + \frac{R}{R+1} - \frac{\epsilon}{R} = 1 - \epsilon\left(\frac{1}{R} - \frac{1}{R^2}\right) < 1,$$

$$\frac{h_J}{R^3} + \frac{h_K}{R} - 1 = \frac{1}{R(R+1)} - \frac{\epsilon}{R^3} + \frac{R}{R+1} + \frac{\epsilon}{R} - 1 = \frac{1-R}{R(R+1)} + \epsilon\left(\frac{1}{R} - \frac{1}{R^3}\right) < 0,$$

$$0 < \frac{R^2}{R+1} < R.$$

□

Theorem 2.12. *For any $R > 1$, any deterministic algorithm with $m = 2$*

machines has a competitive ratio of at least $\frac{3}{2}$.

Proof. Fix a deterministic algorithm $\mathcal{A}$ and call jobs with heat contributions h_J and h_K as stated in Lemma 2.11 J-jobs and K-jobs. At time 0 release one J-job with a deadline of 3. If $\mathcal{A}$ schedules the job on machine q at time 0 then release two K-jobs at time 1 with tight deadlines of 2. By Lemma 2.11, machine q will not be able to schedule a K-job at all, as the K-jobs have tight deadlines and are too hot to be scheduled straight after the J-job. In this case $\mathcal{A}$ can schedule at most two jobs. OPT will remain idle on both machines for time 0 and schedule both K-jobs at time 1, one on each machine, followed by the J-job at time 2 on either machine, for a total of three jobs. This is possible by Lemma 2.11. This gives a competitive ratio of at least $\frac{3}{2}$ in this case.

Otherwise $\mathcal{A}$ will not schedule any jobs at time 0. In this case release two K-jobs at time 2 with tight deadlines of 3. $\mathcal{A}$ cannot schedule all three jobs because once the J-job is scheduled on a machine, that machine cannot schedule a K-job in the next time step. OPT will start the J-job at time 0 and at time 2 will have cooled down enough to schedule both K-jobs giving a total of 3 jobs. This means in this case $\mathcal{A}$ will also have a competitive ratio of at least $\frac{3}{2}$, thus concluding the proof. □

Theorem 2.13. *For any* $R > 1$*, any deterministic algorithm with* $m \geq 3$ *machines has a competitive ratio of at least* $\frac{6}{5}$.

Proof. Fix a deterministic algorithm $\mathcal{A}$ and again we use J-jobs and K-jobs as defined in Lemma 2.11. At time 0 release m J-jobs with deadlines of 3. If $\mathcal{A}$ schedules $\frac{m}{3}$ or more of the J-jobs at time 0 then release m K-jobs at

time 1 with tight deadlines. By Lemma 2.11 any of the machines in $\mathcal{A}$ that scheduled a J-job cannot schedule a K-job. This means that $\mathcal{A}$ misses at least $\frac{m}{3}$ K-jobs, thus completing at most $2m - \frac{m}{3} = \frac{5m}{3}$ jobs. OPT will leave all of its machines idle at time 0, schedule all m K-jobs at time 1 and then all m J-jobs at time 2. This gives a competitive ratio of $\frac{6}{5}$.

Otherwise $\mathcal{A}$ schedules fewer than $\frac{m}{3}$ jobs at time 0. In this case release m K-jobs at time 2 with tight deadlines of 3. If $\mathcal{A}$ schedules another $\frac{m}{3}$ or more J-jobs at time 1, then it misses at least $\frac{m}{3}$ K-jobs at time 2, again by Lemma 2.11. Thus $\mathcal{A}$ completes at most $2m - \frac{m}{3} = \frac{5m}{3}$ jobs. Otherwise, $\mathcal{A}$ schedules fewer than $\frac{m}{3}$ J-jobs at time 1. It schedules at most m jobs (either J or K-jobs) at time 2. Thus the number of jobs that $\mathcal{A}$ completes is at most $\frac{m}{3} + \frac{m}{3} + m = \frac{5m}{3}$. In both cases, OPT can complete $2m$ jobs by scheduling all J-jobs at time 0 and all K-jobs at time 2. Thus the competitive ratio is also $\frac{6}{5}$. □

All these multiple machine lower bounds are far lower than the corresponding (optimal) single machine lower bound and the multiple machine upper bound, thus it is entirely possible to give better algorithms in the multiple machine case. However we observe that in order to do this, some machines must stay idle even when there are admissible jobs:

Theorem 2.14. *For any $R > 1$ and any number of machines m, for any algorithm that always schedules all machines with jobs as long as an admissible one is available, the competitive ratio is at least the same as that given in Theorem 2.8.*

Proof. We simply consider a job instance which consists of m identical copies

of jobs used in the lower bound construction of Theorem 2.8. All machines will then run the jobs at time 0 and thus miss all later jobs, while OPT remains idle at the first time step and schedules all $U + 1$ jobs on each of its machines. □

2.4 Weighted Jobs

We now show some results for the version of the problem where jobs have weights. Some results have been given for the weighted problem with temperature by the other authors in [11]. They consider any $R > 1$ and show that it is not possible to get a constant competitive deterministic algorithm for this problem on a single machine. We denote the upper bound of the maximum possible job weight as W and w.l.o.g. we assume that the minimum job weight is 1. When W is known in advance this model is called *semi-online*, as in this model the algorithm has partial knowledge of future inputs. All of the results given by the other authors are for the semi-online case.

They consider two versions of the problem. The first model is called the *full heat* model. Recall that in the full heat model we have that $\forall J : h_J \leq R$ as already explained and defined in Section 1.7. Using the standard 3-field notation this problem is defined as $1|online\text{-}r_i, h_i, p_i = 1|\sum w_i U_i$ for the single machine case and $P|online\text{-}r_i, h_i, p_i = 1|\sum w_i U_i$ with multiple machines.

They also define a *bounded heat* model where all jobs have a maximum heat bounded away from R, in particular $\forall J : h_J \leq R(1 - \delta)$, for any fixed

$\delta > 0$. We will see that if the heat contribution is bounded away from full heat some interesting and, arguably, more useful results can be obtained. Moreover, full heat is not necessarily reasonable because it effectively means that once any job (of any positive heat contribution) has been scheduled, then no full heat job can be scheduled, no matter how long afterwards. In practice, after some finite amount of idle time, the machine is effectively at the ambient temperature and can run other jobs. Also, a full heat job will almost 'burn' the machine (starting from the ambient temperature) in just one 'quantum' of time, which is perhaps not really that reasonable. Using the standard 3-field notation this problem is defined as $1|\textit{online-}r_i, h_i \leq R(1-\delta), p_i = 1|\sum w_i U_i$ for the single machine case and $P|\textit{online-}r_i, h_i \leq R(1-\delta), p_i = 1|\sum w_i U_i$ with multiple machines.

For the single machine full heat model, in [11] they give an algorithm that is $O(\log W)$-competitive and a matching lower bound of $\Omega(\log W)$ for the randomised version of this problem. They also give an $O(\log(1/\delta))$-competitive randomised algorithm for the bounded heat model, and show that this is optimal. For the full heat multiple machine problem they show an upper bound of $O(mW^{1/m})$ on a deterministic algorithm.

We extend the single machine results from [11] to the multiple machine case. We also show upper bounds for the fully online cases that are only slightly higher than those for the corresponding semi-online cases. We also give a lower bound for the full heat multiple machine problem that matches the existing upper bound within constant factors when the number of machines is fixed.

2.4.1 Full Heat Randomised Bounds

Now we consider the randomised version of the multiple machine full heat problem.

Upper Bound

We show an upper bound for an algorithm that uses the classify-and-randomly-select technique [3]. This analysis works both in the semi-online and fully online case.

When W is known in advance, we first partition the weight range $[1, W]$ in to $\lceil \ln W \rceil$ classes, with the ranges $[1, e], (e, e^2], ..., (e^{\lfloor \ln W \rfloor - 1}, e^{\lfloor \ln W \rfloor}], (e^{\lfloor \ln W \rfloor}, W]$. In each class, the job weights differ by a factor of at most e. We then randomly choose a class and use a reasonable algorithm to schedule the jobs from the chosen class on all machines. All other jobs are ignored.

When W is not known in advance we use a technique from [3] to convert the given semi-online algorithm into a fully online algorithm, with a slightly higher competitive ratio. We now describe the algorithm. We define a function $f(\epsilon) = \sum_{i=1}^{\infty} \frac{1}{i^{1+\epsilon}}$, noting that $f(\epsilon) < \infty$ for all $\epsilon > 0$. At each time step the algorithm holds a set of k classes $\mathcal{A}_i$, $1 \leq i \leq k$, with $k = 0$ at the beginning. Each class is associated with an integer label; a class with label j denotes the fact that each job J in this class has weight $e^{j-1} < w_J \leq e^j$.

When a job J arrives with a weight $e^{j-1} < w_J \leq e^j$, the algorithm checks if there is already a class with label j. If the class exists then J is associated to this class. Otherwise the algorithm creates a new class, sets its label to j, and associates J to the new class. When each class $\mathcal{A}_i$ is created, if the

algorithm has not already selected a class, then the algorithm selects $\mathcal{A}_i$ with a conditional probability such that the absolute probability that it is selected is $p_i = \frac{1}{f(\epsilon)} \frac{1}{i^{1+\epsilon}}$. Since $\sum_{i=1}^{\infty} p_i = 1$ these probabilities are valid. It is also clear that although W is not known to the algorithm in advance, at most $\lceil \ln W \rceil$ classes are eventually created, and the job weights within a class differ by a factor of at most e. When a class has been selected the algorithm then ignores all jobs not in this class. We then run an $O(1)$-competitive reasonable algorithm for unweighted instances for jobs in this class on all machines, ignoring the job weights.

We now show the competitiveness of this algorithm using techniques similar to those for the single machine case in [11].

Theorem 2.15. *The above algorithm is $O(\log W)$-competitive, when W is known in advance. If W is not known in advance, the above algorithm is $O(\log^{1+\epsilon} W)$-competitive for any $\epsilon > 0$.*

Proof. From Theorem 2.7 in Section 2.3.1, it must be that for a fixed R the competitive ratio of reasonable algorithms is constant. We denote the competitive ratio of the online algorithm for the unweighted case as c_R. Let $\mathcal{A}_i$ and OPT_i be the schedule for the ith class of the online algorithm and the optimal schedule respectively. It must be that for each class i $|OPT_i| \leq c_R e |\mathcal{A}_i|$ since the algorithm is c_R competitive for the number of jobs, and a factor at most e is lost on job weights. It is clear that it must also be that $\sum |OPT_i| \geq |OPT|$. Therefore when W is known in advance

$$E[|\mathcal{A}|] = \frac{\sum |\mathcal{A}_i|}{\lceil \ln W \rceil} \geq \frac{\sum |OPT_i|/c_R e}{\lceil \ln W \rceil} \geq \frac{|OPT|}{c_R e \lceil \ln W \rceil}.$$

This makes the algorithm $O(\log W)$-competitive in the case when W is known in advance.

In the fully online case we have that

$$E[|\mathcal{A}|] = \sum_{i=1}^{\lceil \ln W \rceil} p_i |\mathcal{A}_i| \geq \sum_{i=1}^{\lceil \ln W \rceil} p_{\lceil \ln W \rceil} |\mathcal{A}_i| \geq \frac{\sum_{i=1}^{\lceil \ln W \rceil} |OPT_i|}{c_R e f(\epsilon) \lceil \ln W \rceil^{1+\epsilon}} \geq \frac{|OPT|}{c_R e f(\epsilon) \lceil \ln W \rceil^{1+\epsilon}}.$$

Thus the algorithm is $(c_R e f(\epsilon) \lceil \ln W \rceil^{1+\epsilon})$-competitive in the fully online case.

□

Although this bound is given for the multiple machine case, it still applies to the single machine case. This means that we have extended the semi-online results from [11] to show that the above algorithm is $O(\log^{1+\epsilon} W)$-competitive for the fully online single machine case, as well with multiple machines.

Lower Bound

We now give a lower bound that shows that this is optimal when W is known in advance, and almost optimal in the fully online case. We use Yao's principle [52] and specify a probabilistic construction of the adversary and bound the competitive ratio against deterministic algorithms with inputs over this distribution. This lower bound again uses techniques similar to those for the single machine case in [11].

Theorem 2.16. *Any randomised algorithm has a competitive ratio of at least* $\Omega(\log W)$.

Proof. Choose a large enough positive integer n, and let J_i, for all $1 \leq i \leq n$, be a job with $w_{J_i} = 2^{i-1}$ and $h_{J_i} = R$. At each time step $i - 1$ we release m

copies of job J_i, each with a tight deadline of i.

At each time step i, for $1 \leq i \leq n$, with conditional probability of $1/2$ the sequence stops, and with conditional probability $1/2$ the adversary continues to release job J_{i+1}. The probability is conditional on the fact that the time step i is actually reached, i.e. the adversary has not already stopped at some earlier time step. If J_n is released, then the sequence stops. Thus, we have a total of n different input sequences, appearing with probability $1/2, 1/4, ..., 1/2^{n-1}, 1/2^{n-1}$ as shown for 1 machine in Figure 2.5.

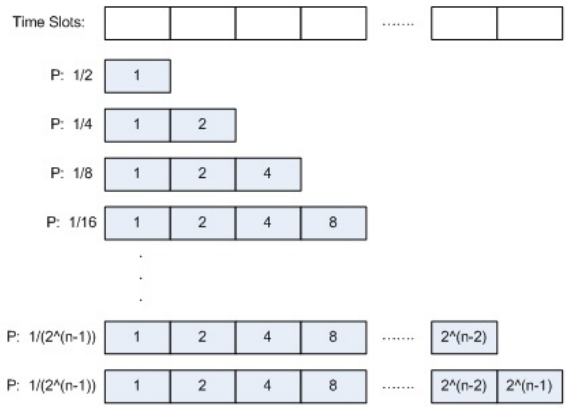

Figure 2.5: Input distribution for the randomised lower bound

Since all jobs have full heat, any algorithm can schedule at most one job on any machine. Hence, without loss of generality, we can restrict our attention to a particular machine q and a deterministic schedule $\mathcal{A}_{q,j}$ of the following form: do not start any of the jobs $J_1, ...J_{j-1}$ on machine q and start the job J_j for some $1 \leq j \leq n$. Then $\mathcal{A}_{q,j}$ gets the value of job J_j if the adversary releases J_j , i.e. the adversary has not stopped releasing jobs before J_j. This happens with marginal probability $1/2^{j-1}$. Otherwise, if the adversary stopped before

releasing J_j, no value is obtained. Thus $E[|\mathcal{A}_{q,j}|] = (1/2^{j-1})(2^{j-1}) = 1$, note that this is independent of j. Thus the expected value obtained by any online algorithm is $E[|\mathcal{A}|] = \sum_{i=1}^{m} E[|\mathcal{A}_{i,j}|] = m$.

Meanwhile OPT will always schedule the last set of jobs released by the adversary, one on each machine

$$\begin{aligned} E[OPT] &= m\left(\sum_{i=1}^{n-1}\left(\frac{1}{2^i}\right)(2^{i-1}) + \left(\frac{1}{2^{n-1}}\right)(2^{n-1})\right) \\ &= m\left(\frac{n-1}{2} + 1\right) \\ &= \frac{m(n+1)}{2}. \end{aligned}$$

Therefore the competitive ratio is at least $(n+1)/2$. Since for the maximum W possible $W \leq 2^{n-1}$, we have proven a lower bound of $\Omega(\log W)$ on the competitive ratio. □

This matches the semi-online upper bound given in Theorem 2.15 meaning the algorithm given is optimal within a constant factor. Both bounds also match those given for the single machine case in [11] showing that allowing the use of extra machines does not provide any substantial benefit in the randomised and weighted full heat case.

2.4.2 Full Heat Deterministic Lower Bound

We now show a lower bound on deterministic algorithms for the multiple machine full heat version of the problem.

Theorem 2.17. *No deterministic algorithm for the full heat problem can be*

better than $(mW)^{1/m}$*-competitive, when* $W \geq m^{m-1}$.

Proof. Fix a deterministic algorithm $\mathcal{A}$. Consider a sequence of jobs J_i for $1 \leq i \leq m$, where $r_{J_i} = i - 1$; $d_{J_i} = i$ and $h_{J_i} = R$. The weights of the jobs are given by $w_{J_1} = 1$, and $w_{J_i} = (c-1)\sum_{j=1}^{i-1} w_{J_j}$ for $i > 1$ where $c > 1$ is some value to be chosen later. Each J_i is released successively. Note that the job J_i cannot be scheduled on any machine that has already scheduled another job J_j for $j < i$ as all of the jobs have full heat contribution. If $\mathcal{A}$ chooses not to schedule some job J_k on any machine then the subsequent jobs will not be released. In this case $\mathcal{A}$ has scheduled the first $k-1$ jobs and so has a weighted throughput of $\sum_{i=1}^{k-1} w_{J_i}$. OPT will schedule all k jobs and so have a weighted throughput of $\sum_{i=1}^{k-1} w_{J_i} + w_{J_k}$. As $w_{J_k} = (c-1)\sum_{i=1}^{k-1} w_{J_i}$, this gives OPT a weighted throughput of $c\sum_{i=1}^{k-1} w_{J_i}$ giving a competitive ratio of c in this case.

Otherwise $\mathcal{A}$ does not miss any of the J_i jobs, $1 \leq i \leq m$. This means that $\mathcal{A}$ has scheduled exactly one job on each of its machines. At time m we then release m jobs $X_1, ..., X_m$ that have heat contributions of R, tight deadlines of $m+1$ and weights of $(c\sum_{i=1}^{m} w_{J_i})/m$. $\mathcal{A}$ will be too hot to schedule any of them, while OPT can skip all the J_i and schedule all $X_1, X_2, ..., X_m$, one on each machine. This gives OPT a weighted throughput of $c\sum_{i=1}^{m} w_{J_i}$ while $\mathcal{A}$ has a weighted throughput of $\sum_{i=1}^{m} w_{J_i}$ which again gives a competitive ratio of c.

Each job J_i for $i > 1$ has weight $(c-1)\sum_{j=1}^{i-1} w_{J_j}$ and $w_{J_1} = 1$. Solving

the recursion we get $w_{J_i} = c^{i-1} - c^{i-2}$. Thus

$$w_{X_i} = c\left(\frac{\sum_{i=2}^{m}(c^{i-1} - c^{i-2}) + 1}{m}\right) = \frac{c^m}{m}.$$

The minimum job weight is 1 and the maximum job weight is either w_{J_m} or w_{X_m}. The ratio of maximum to minimum job weights is therefore

$$W = \max\{c^{m-1} - c^{m-2}, \frac{c^m}{m}\}$$

Suppose we have $c^{m-1} - c^{m-2} \leq \frac{c^m}{m}$, then $W = \frac{c^m}{m}$, i.e. $c = (mW)^{1/m}$. This is the desired competitive ratio. The condition $c^{m-1} - c^{m-2} \leq \frac{c^m}{m}$ is equivalent to $cm \leq c^2 + m$, which, with the value of $c = (mW)^{1/m}$, is satisfied if, for example, $W \geq m^{m-1}$. □

For any fixed m, this matches the upper bound of $O(mW^{1/m})$ for the algorithm given in [11], showing that this algorithm is optimal within a constant factor.

2.4.3 Bounded Heat Randomised Bounds

We now show some bounds for randomised algorithms with multiple machines where the maximum heat of a job is bounded to $R(1-\delta)$. The algorithm is based on the m-machine unit job scheduling algorithm DMIX-m in [17] and the schedule partitioning idea for the single machine case in [11]. We use the DMIX-m algorithm here because it is online and has a competitive ratio of $(1 - (\frac{m}{m+1})^m)^{-1}$ which is no greater than 2 for all possible values of m. We use a lemma from [11] that is restated here for completeness.

Lemma 2.18. *[11] Suppose an algorithm runs jobs of heat at most h every $k \geq 1$ time slots, and keeps idle at other slots. If $h \leq R(1 - 1/R^k)$, then the temperature does not exceed 1 at any point.*

Now we describe the algorithm. We consider the case where all jobs have a maximum heat contribution of $R(1 - 1/R^k)$. First we ignore the heat contributions of jobs and use the DMIX-m algorithm to produce a m-processor schedule S. The schedule S is then partitioned into k sub-schedules $S_1, ..., S_k$. Each sub-schedule S_i schedules the same job as S during the time slots of the form $(i-1) + jk$ for $j = 0, 1, 2, ...$ That means S_1 schedules the same job as S during each of the time slots $0, k, 2k, ...$ and stays idle at all other slots. Similarly S_2 schedules the same job as S during each of the time slots $1, k+1, 2k+1, ...$ and stays idle at all other slots and so on for each $S_1, ..., S_k$. The algorithm randomly picks and executes one of the schedules with probability $1/k$.

Theorem 2.19. *For any $0 < \delta \leq 1/R$, if the maximum job heat is bounded to $R(1 - \delta)$ then the above algorithm is $O(\log(1/\delta))$-competitive.*

Proof. We set $k = \lceil \log_R(1/\delta) \rceil$ and partition the schedule S as above into $S_1, ..., S_k$. By Lemma 2.18 it is clear that none of the sub-schedules will ever exceed the temperature threshold as each schedules no more than one job every k time steps. We will set $c_m = (1 - (\frac{m}{m+1})^m)^{-1} \leq 2$, which is the competitive ratio of DMIX-m. Let OPT' denote the optimal offline schedule for the same input instance but without heat considerations.

It is clear that $|OPT| \leq |OPT'|$ because although the schedule of OPT' may not be feasible with heat considerations, its weighted throughput must

be at least that of OPT, otherwise this would contradict its optimality. We must also have from the definition of the algorithm that $|S| = \sum_{i=1}^{k} |S_i|$ and from the competitiveness of DMIX-m that $|OPT'| \leq c_m|S|$. The expected weighted throughput $E[|A|]$ of our algorithm is equal to $1/k$ of the total weighted throughput of $S_1, ..., S_k$ i.e. $E[|A|] = \frac{1}{k}\sum_{i=1}^{k} |S_i|$.

Combining these we get that $|OPT| \leq c_m k E[|A|]$. Therefore, by our definition of k, we have that the algorithm is $O(\log(1/\delta))$-competitive. □

We now give a lower bound that shows that this algorithm is optimal up to a constant factor. Again this is extended from the proof for the single machine case in [11].

Theorem 2.20. *If the maximum job heat is bounded to $R(1 - 1/R^k)$, for large enough k, no randomised algorithm can have competitive ratio better than k.*

Proof. Set H to be the maximum job heat, that is $H = R(1 - 1/R^k)$. Let $n \geq 1$ be the largest integer such that $H/R^n + H/R > 1$ (It is required that $H \geq R/2$ i.e. $R^k \geq 2$ for such an n to exist). Let x be a real number in $(0, 1)$, to be determined later. The proof is similar to that of Theorem 2.16. Let J_i, for all $1 \leq i \leq n$, be a job with $w_{J_i} = 1/x^{i-1}$ and $h_{J_i} = H$. At each time step $i - 1$ we release m copies of job J_i, each with a tight deadline of i. Then at each time step i $(1 \leq i < n)$, with conditional probability $1 - x$ the sequence stops, and with conditional probability x the adversary continues to release job J_{i+1}. The probability is conditional on the fact that the adversary has not stopped releasing jobs before time step i. If J_n is released, then the sequence stops.

Since $H/R^n + H/R > 1$, any algorithm can schedule at most one of those jobs on any one machine. Hence, similar to Theorem 2.16, we can restrict our attention to a particular machine q and a deterministic schedule $\mathcal{A}_{q,j}$ that does not start any of the jobs $J_1, ...J_{j-1}$, then starts the job J_j for some $1 \leq j \leq n$. Thus $E[|\mathcal{A}_{q,j}|] = (1/x^{j-1})(x^{j-1}) = 1$ and so the expected value obtained by any online algorithm is $E[|\mathcal{A}|] = \sum_{i=1}^{m} E[|\mathcal{A}_{i,j}|] = m$.

As before OPT will always schedule the last set of jobs released by the adversary, one on each machine

$$\begin{aligned} E[OPT] &= m\left(\sum_{i=1}^{n-1}(x^{i-1}(1-x))\left(\frac{1}{x^{i-1}}\right) + (x^{n-1})\left(\frac{1}{x^{n-1}}\right)\right) \\ &= m\left((n-1)(1-x)+1\right). \end{aligned}$$

Therefore, the competitive ratio is at least $(n-1)(1-x)+1$. Choose x to be arbitrarily close to 0, then the ratio can be made arbitrarily close to n. The condition of $H/R^n + H/R > 1$ is equivalent to $n < \log_R(RH/(R-H))$. It follows that the lower bound is $\lceil \log_R(RH/(R-H)) \rceil - 1$. Since $H = R(1-\delta) = R(1-1/R^k)$ we have that the lower bound is equal to $\lceil \log_R(R^2(1-1/R^k)/(1/R^{k-1})) \rceil - 1 = \lceil \log_R(R^{k+1} - R) \rceil - 1$. This lower bound is equal to k for any $R > 1$ and sufficiently small δ. □

This again almost matches the upper bound given meaning the algorithm given is optimal within a constant factor. As with the full heat case both bounds also match those given for the single machine case in [11] showing that allowing the use of extra machines does not provide any substantial benefit.

2.5 Summary

In this chapter we considered maximising the throughput of jobs in the model with unit length jobs, that are both unweighted and weighted, with single and multiple machines. For the unweighted single machine case we show that reasonable algorithms are optimal for all cooling factors, while matching lower bounds have not been found for the case with multiple machines. All of the bounds given increase as the cooling factor decreases. We also show that the algorithm Hottest First performs worse than reasonable algorithms.

For weighted jobs we show that the bounds for the multiple machine cases are the same as those for the single machine case, in the randomised full heat and bounded heat models. We also show a fully online algorithm for the randomised full heat model with a competitive ratio only slightly higher than in the semi-online case. We also show a lower bound for the deterministic full heat case on multiple machines that matches the existing upper bound for a fixed number of machines, within constant factors.

Chapter 3

Equal Length Throughput

We now extend the unit length problem to consider the model where all jobs have the same length, but this length is larger than unit length. We will denote this length by p, where $p > 1$ is an integer i.e. $\forall J : p_J = p > 1$. We mainly consider the unweighted version of this problem. All results in this chapter are for all $R > 1$ and for the one machine case.

For simplicity, when using jobs that are longer than unit length we denote the total heat contribution of a job J over all p_J time steps as h^*_J, defined as

$$h^*_J = \sum_{i=0}^{p_J-1} \frac{h_J}{R^i} = \frac{R^{p_J} - 1}{R^{p_J-1}(R-1)} h_J \tag{3.1}$$

This takes into account the cooling at each step. With this definition the temperature after executing J at time t for its entire duration (without pre-emption) is given simply by $\tau_{t+p_J} = \tau_t / R^{p_J} + h^*_J / R$.

It is simple to extend the restriction of $h_J \leq R$ from the unit length case to show that we also have that no job with $h^*_J > R$ can be admissible for the

non-preemptive and preemptive restarts model. Therefore for these models we assume that $h_J^* \leq R$ for all jobs. We also assume the same for the resume model (this will be discussed in Section 3.5).

3.1 Motivation

Considering longer than unit length jobs is useful for several reasons: first, if jobs (for example, CPU processes) are already partitioned into unit length slices as in the unit length model, credit will be given to each completed slice even though the job itself may not be completed. Thus to correctly account for the value gained, the scheduler needs to be aware of the longer, un-partitioned jobs.

Secondly, because jobs have longer than unit length, the scheduling may now involve preemption. This could be useful, for example, as an urgent job may arrive while another less urgent job is already running. Depending on the application domain, it may not be possible (or it may be costly) to have preemption, making it useful to analyse the power of preemption which is not possible in the unit-length model.

3.2 Previous Results

Without temperature considerations the case that involves one machine scheduling jobs that have an equal, but longer than unit, length has been studied. For the case of unweighted equal length jobs on one machine and no preemption allowed, a greedy algorithm has been shown with a competitive ratio of

2 [8] and this has been shown to be optimal for the deterministic case [29]. For the deterministic case with preemptive restarts allowed an optimal 3/2-competitive algorithm has been shown [20]. When preemptive resumes are allowed it is well known it is possible to produce an optimal schedule for the unweighted problem, see for example [49].

The randomised version of the equal length job problem on one machine has also been studied. Without preemption a 5/3-competitive algorithm is known [20], and a lower bound of 4/3 has been shown [29]. For the case where preemptive restarts are allowed as well as randomisation, a lower bound of 6/5-competitive is also shown [20].

For the version of the one machine non-preemptive problem where jobs are of equal length and have weights it is well known that no constant competitive ratio can be achieved. An upper bound of $1 + w_{max}/w_{min}$ has been shown for a deterministic algorithm where w_{max} and w_{min} are the maximum and minimum allowed job weights respectively [38] and they show that the $1 + \ln(w_{max}/w_{min})$ lower bound from [5] applies to this case against an oblivious adversary. They also show a randomised algorithm that achieves an upper bound of $(1+e)(1+\ln(w_{max}/w_{min}))$, which is therefore only a constant factor of $1+e$ away from optimal [38]. The results from [38] also apply to the model where $m \geq 1$ parallel/identical machines are allowed.

The current best algorithm for the weighted equal length job problem on one machine where preemption is allowed has an upper bound of 4.24 [48]. This upper bound applies to both the restarts and resumes models of preemption, although the algorithms are slightly different for the different models. The current best lower bound for this problem is 4 [50].

The best algorithms for maximising the throughput of equal length jobs without preemption on 2 machines have a 3/2-competitive ratio [23, 30] and this is optimal. For a general number of machines m a lower bound of 6/5 [23] is known, and the best known algorithm without preemption is deterministic and has a competitive ratio of $e/(e-1) \approx 1.582$ for large m [22].

For the multiple machine problem where jobs are weighted, equal length and always have tight deadlines (i.e. the jobs are intervals), an algorithm has been shown to be 2-competitive when m is even and $2 + 2/(2m-1)$-competitive when m is odd [26].

For the model of equal length jobs with temperature considerations we are not aware of any previous work.

3.3 The Non-Preemptive Model

We now give some results for the non-preemptive version of the problem. Using standard 3-field notation this problem is defined as $1|\mathit{online}\text{-}r_i, h_i, p_i = p|\sum U_i$.

The competitive ratios for the non-preemptive case are given in terms of a value E_1 (defined in terms of p and R) that is the largest integer $e_1 > 0$ such that $(e_1 + R)R^p > R^{e_1 p+1} + e_1$ holds. Note that E_1 increases when R decreases, and also as the job length p decreases. Table 3.1 shows the values of E_1 for different values of R and p.

E_1	$p = 2$	$p = 3$	$p = 4$
1	$R > \sqrt[3]{2} \approx 1.260$	$R > \sqrt[4]{2} \approx 1.189$	$R > \sqrt[5]{2} \approx 1.149$
2	$1.260 \gtrsim R \gtrsim 1.105$	$1.189 \gtrsim R \gtrsim 1.075$	$1.149 \gtrsim R \gtrsim 1.059$
3	$1.105 \gtrsim R \gtrsim 1.058$	$1.075 \gtrsim R \gtrsim 1.041$	$1.059 \gtrsim R \gtrsim 1.032$
4	$1.058 \gtrsim R \gtrsim 1.037$	$1.041 \gtrsim R \gtrsim 1.026$	$1.032 \gtrsim R \gtrsim 1.020$
5	$1.037 \gtrsim R \gtrsim 1.026$	$1.026 \gtrsim R \gtrsim 1.018$	$1.020 \gtrsim R \gtrsim 1.014$

Table 3.1: Table of E_1 values for selected R and p values.

3.3.1 Lower Bound

We now show a lower bound for deterministic algorithms in the non-preemptive model. Recall the definition of OPT-only jobs as given in Chapter 2, modified here for the single machine case: we refer to a job as an *OPT-only job* if it is started by OPT at some time t when $\mathcal{A}$ is idle, and at time t the job is too hot to be scheduled by $\mathcal{A}$.

Theorem 3.1. *No deterministic non-preemptive algorithm can be better than $(E_1 + 2)$-competitive.*

Proof. Fix any deterministic algorithm $\mathcal{A}$. At time 0 release a job J with a deadline of $2D + (E_1 + 3)p + 1$ and a total heat contribution $h_J^* = R - \epsilon$ for a sufficiently small $\epsilon > 0$, where

$$D = \left\lceil \log_R \left(\frac{1}{R^{p-1}\epsilon} \right) \right\rceil$$

is the time for an idle machine's temperature to reduce from 1 to $R^{p-1}\epsilon$. The value $R^{p-1}\epsilon$ is the maximum temperature at which an algorithm can start J, since $\frac{R^{p-1}\epsilon}{R^p} + \frac{R-\epsilon}{R} = 1$. If $\mathcal{A}$ does not schedule J at all then OPT schedules it at time 0, and the competitive ratio of $\mathcal{A}$ is infinite, so we can assume $\mathcal{A}$

schedules J at some time u.

Case 1: If $u \leq D + p$, then OPT remains idle at u and at time $u + 1$ we release a job K that has a tight deadline and a heat contribution $h_K = 0$. OPT schedules this job, while $\mathcal{A}$ must continue with J as it is non-preemptive, reaching a temperature of $1 - \frac{\epsilon}{R}$ at time $u + p$. At time $t = u + ip + 1$ for $i = 1, 2, ...$, the temperature of $\mathcal{A}$ is given by (ignoring the small ϵ) $\tau_{u+ip+1} = \frac{1}{R^{(i-1)p+1}}$. At each such t, as long as $\tau_{u+ip+1} > \tau'_{u+ip+1}$, the adversary releases a tight job J_i with the smallest possible heat contribution such that $h^*_{J_i} > R - \frac{\tau_{u+ip+1}}{R^{p-1}}$. $\mathcal{A}$ will be (just) too hot to schedule the jobs, while OPT will schedule each of them (i.e. they are OPT-only jobs). The temperature of OPT at time $u + ip + 1$ for every $i \geq 1$ is given by

$$\tau'_{u+p+1} = 0, \quad \tau'_{u+ip+1} = \frac{\tau'_{u+(i-1)p+1}}{R^p} + \frac{h^*_{J_{i-1}}}{R} \text{ for } i \geq 2$$

This can be solved to give

$$\tau'_{u+ip+1} = \frac{R^p - R^{p-(i-1)p}}{R^p - 1} - \frac{i-1}{R^{(i-1)p+1}}$$

Strictly speaking the above value of τ'_{u+ip+1} should use a $>$ instead of $=$ sign as $h^*_{J_i}$ is defined to be just hotter than $R - \frac{\tau_{u+ip+1}}{R^{p-1}}$. Moreover, as can be seen later, the initial temperature at time $u + p + 1$ may not actually be zero. However these differences can be made arbitrarily small so for convenience we use the equal sign here. A full analysis incorporating this detail can be done similar to that in Theorem 2.8.

The maximum number of OPT-only jobs is the largest integer $i > 0$ such that $\tau_{u+ip+1} > \tau'_{u+ip+1}$ holds, which is satisfied if

$$\frac{1}{R^{(i-1)p+1}} > \frac{R^p - R^{p-(i-1)p}}{R^p - 1} - \frac{i-1}{R^{(i-1)p+1}}$$

This can be simplified to give

$$(i + R)R^p > R^{ip+1} + i \tag{3.2}$$

which is the same as the formula that defines E_1.

After completing J_{E_1} at time $(u + E_1 p + 1) + p$, OPT will then remain idle for D steps so that it is cool enough to schedule J, at which point this will be done. Since $u \leq D + p$, it can still meet its deadline.

Case 2: If $u > D + p$ then OPT schedules J at time 0. Since $u > D + p$, at time $u + 1$ OPT's temperature will be less than $R^{p-1}\epsilon$, i.e. almost 0. For a sufficiently small ϵ we can then continue as in the previous case, with all E_1 jobs and K being scheduled by OPT while $\mathcal{A}$ cannot schedule any more jobs.

This concludes the proof as J is the only job that $\mathcal{A}$ is able to schedule while OPT is able to schedule $J_1, ..., J_{E_1}$ as well as J and K, making $\mathcal{A}$ no better than $(E_1 + 2)$-competitive.

□

3.3.2 Upper Bound: Coolest First

In this section the algorithm Coolest First will be shown to have an optimal competitive ratio. Coolest First schedules a job whenever one is admissible, and if several are admissible will always schedule the job with the smallest heat contribution. Whenever several jobs have the same smallest heat contribution it will always schedule a job in this set that has the smallest or equal smallest deadline. Coolest First will never preempt a job.

Before proving the competitive ratio, we need two lemmas to bound the number of OPT-only jobs during an idle period of $\mathcal{A}$. The two lemmas are similar, but in the second one, OPT starts those jobs one time step later.

Lemma 3.2. *If $\mathcal{A}$ completes a job at time u then OPT can start at most E_2 OPT-only jobs on or after time u but before $\mathcal{A}$ starts the next job, where E_2 is the largest integer $e_2 \geq 0$ such that $(e_2+1)R^p > R^{e_2 p} + e_2$ holds.*

Proof. For $\mathcal{A}$ we must have $\tau_u \leq 1$. We first assume OPT runs these OPT-only jobs consecutively starting from u without idle time or other jobs in between; we will return to this assumption later. As $\mathcal{A}$ remains idle while OPT starts jobs, the temperature of $\mathcal{A}$ after i jobs have been run by OPT is given by

$$\tau_{u+ip} \leq \frac{1}{R^{ip}} \tag{3.3}$$

For each OPT-only job J_i it must be too hot to be admissible in $\mathcal{A}$, i.e.

$$h^*_{J_i} > R - \frac{\tau_{u+(i-1)p}}{R^{p-1}}$$

Therefore the temperature of OPT after it has run i jobs is given by

$$\tau'_u \geq 0, \quad \tau'_{u+ip} = \frac{\tau'_{u+(i-1)p}}{R^p} + \frac{h^*_{J_i}}{R}$$

This can then be solved to give

$$\tau'_{u+ip} \geq \frac{R^p - R^{p-ip}}{R^p - 1} - \frac{i}{R^{ip}} \tag{3.4}$$

As long as $\tau_{u+(i-1)p} > \tau'_{u+(i-1)p}$ it remains possible for OPT to schedule the i-th OPT-only job J_i at time $u + (i-1)p$. The maximum number of OPT-only jobs is thus the largest $i > 0$ such that $\tau_{u+(i-1)p} > \tau'_{u+(i-1)p}$, which using inequalities (3.3) and (3.4) is upper bounded by

$$\frac{1}{R^{(i-1)p}} > \frac{R^p - R^{p-(i-1)p}}{R^p - 1} - \frac{i-1}{R^{(i-1)p}}$$

This can be simplified to

$$(i+1)R^p > R^{ip} + i \tag{3.5}$$

which is the same as the formula that defines E_2.

We now return to the assumption that these OPT-only jobs are consecutive. Our aim is to upper-bound the number of jobs that can be scheduled by OPT but are too hot to be scheduled by $\mathcal{A}$ in a time period when $\mathcal{A}$ remains idle. This relies on $\mathcal{A}$ being hotter than OPT during this whole period. Suppose $J_1, ..., J_{E_2}$ are not consecutive, then there are time steps in between where OPT is idle or schedules a job that is not too hot to be

scheduled by $\mathcal{A}$. It can be easily shown (by a trivial extension of Lemma 2.4 from the unit length case) that such time steps will reduce the difference in temperature between OPT and $\mathcal{A}$, thus will not help OPT schedule more OPT-only jobs. □

Lemma 3.3. *If $\mathcal{A}$ completes a job at time u and OPT does not start a job at $[u, u+1)$, then OPT can start at most E_1 OPT-only jobs on or after time $u+1$ but before $\mathcal{A}$ starts the next job.*

Proof. For $\mathcal{A}$ we must have $\tau_u \leq 1$ and $\tau_{u+1} \leq 1/R$. As in the previous lemma we assume these OPT-only jobs are started consecutively from time $u+1$. As $\mathcal{A}$ remains idle while OPT starts jobs, the temperature of $\mathcal{A}$ after i jobs have been run by OPT is given by

$$\tau_{u+ip+1} \leq \frac{1}{R^{ip+1}} \tag{3.6}$$

For each OPT-only job J_i it must be too hot to be admissible in $\mathcal{A}$, i.e.

$$h^*_{J_i} > R - \frac{\tau_{u+(i-1)p+1}}{R^{p-1}}$$

Therefore the temperature of OPT after it has run i jobs is given by

$$\tau'_{u+1} \geq 0, \quad \tau'_{u+ip+1} = \frac{\tau'_{u+(i-1)p+1}}{R^p} + \frac{h^*_{J_i}}{R}$$

This can be solved to

$$\tau'_{u+ip+1} \geq \frac{R^p - R^{p-ip}}{R^p - 1} - \frac{i}{R^{ip+1}} \tag{3.7}$$

As long as $\tau_{u+(i-1)p+1} > \tau'_{u+(i-1)p+1}$ it remains possible for OPT to schedule the i-th OPT-only job J_i. The maximum number of OPT-only jobs is thus the largest $i > 0$ such that $\tau_{u+(i-1)p+1} > \tau'_{u+(i-1)p+1}$, which using inequalities (3.6) and (3.7) is upper bounded by

$$\frac{1}{R^{(i-1)p+1}} > \frac{R^p - R^{p-(i-1)p}}{R^p - 1} - \frac{i-1}{R^{(i-1)p+1}}$$

This can be simplified to

$$(i+R)R^p > R^{ip+1} + i \tag{3.8}$$

which is the same as the formula that defines E_1. □

It turns out that E_2 is always close to E_1.

Lemma 3.4. *If E_1 and E_2 are defined from the same p and R, then $E_1 \leq E_2 \leq E_1 + 1$.*

Proof. For simplicity, define the constant $L = R^p$ and define $f(x) = L^x$, $f_1(x) = x\frac{L-1}{R} + L$ and $f_2(x) = x(L-1) + L$. Then E_1 and E_2 are the largest integers satisfying the inequalities $f(x) < f_1(x)$ and $f(x) < f_2(x)$ respectively. Let e_1^* and e_2^* be the unique positive roots of the corresponding equations, i.e. $f(e_1^*) = f_1(e_1^*)$ and $f(e_2^*) = f_2(e_2^*)$.

Observe that $f(x)$ is an exponential function, $f_1(x)$ and $f_2(x)$ are linear increasing functions with f_2 having a steeper slope, $f(0) = 1 < L = f_1(0) = f_2(0)$. Thus the intersection of $f(x)$ and $f_1(x)$ occurs at an earlier point of x than the intersection of $f(x)$ and $f_2(x)$, i.e. $e_1^* \leq e_2^*$.

Next we show $e_2^* \leq e_1^* + 1$. Note that $f(e_2^*) = f_2(e_2^*)$ and $f(e_1^*) \leq f_2(e_1^*)$ (as $e_1^* \leq e_2^*$). If $f(e_1^* + 1) \geq f_2(e_1^* + 1)$ this will establish the claim. This is equivalent to

$$\begin{aligned}
& L^{e_1^*+1} \geq (L-1)(e_1^*+1) + L \\
\iff \quad & L\left(\frac{L-1}{R}e_1^* + L\right) \geq (L-1)(e_1^*+1) + L \\
\iff \quad & L(L-1)e_1^* + L^2R \geq (L-1)Re_1^* + (L-1)R + LR \\
\iff \quad & (L-1)(L-R)e_1^* \geq (2L-1)R - L^2R \\
\iff \quad & (L-1)(L-R)e_1^* \geq -(L-1)^2R \\
\iff \quad & e_1^* \geq \frac{(L-1)R}{R-L}
\end{aligned}$$

which is true as $L = R^p > R$ so the right hand side of the last inequality is negative.

So we have $e_1^* \leq e_2^* \leq e_1^* + 1$. It follows from their definitions that $E_1 = \lceil e_1^* \rceil - 1$ and $E_2 = \lceil e_2^* \rceil - 1$. As $x \leq y$ implies $\lceil x \rceil \leq \lceil y \rceil$ for any x and y, this proves the lemma. □

We now prove the competitive ratio of Coolest First.

Theorem 3.5. *Coolest First is $(E_1 + 2)$-competitive for the non-preemptive model.*

Proof. The algorithm will be analysed by dividing the schedules into regions, and then associating with each region certain credits for jobs scheduled by OPT and $\mathcal{A}$. Each region is a time interval $[u, v)$, where u is the time when $\mathcal{A}$ starts a job, and v is the earliest time on or after $u + p$ when $\mathcal{A}$ starts another job. If $\mathcal{A}$ does not start another job after u, set $v = \infty$. Clearly, for

every two consecutive regions $[u, v)$ and $[u', v')$ it must be that $v = u'$, i.e. the regions form a non-overlapping partition of the schedules.

The first region starts when $\mathcal{A}$ first schedules a job. OPT cannot schedule any job before this time as $\mathcal{A}$ schedules a job as soon as one is released. Thus it is not possible for OPT to start a job before the start of the first region.

Each job completed by $\mathcal{A}$ or OPT will be given credits towards some region. We use $\Phi^{OPT}_{[u,v)}$ and $\Phi^{\mathcal{A}}_{[u,v)}$ to denote the credits associated to region $[u, v)$ of OPT and $\mathcal{A}$ respectively, and Φ^{OPT}_{∞} and $\Phi^{\mathcal{A}}_{\infty}$ to represent the total credits of OPT and $\mathcal{A}$ respectively after jobs have stopped being released and all jobs have expired or completed. We prove that for every region $[u, v)$ it must be that $\Phi^{OPT}_{[u,v)} \leq (E_1 + 2) \cdot \Phi^{\mathcal{A}}_{[u,v)}$. Summing over all regions this gives $\Phi^{OPT}_{\infty} \leq (E_1 + 2) \cdot \Phi^{\mathcal{A}}_{\infty}$. This shows that $\mathcal{A}$ is $(E_1 + 2)$-competitive.

Fix a region, and let J be the job that $\mathcal{A}$ schedules in this region. The starting time of J is u. We give $\mathcal{A}$ one credit for the job it schedules in each region, so $\Phi^{\mathcal{A}}_{[u,v)} = 1$ for every region. We also give one credit to $\Phi^{OPT}_{[u,v)}$ in every region, to account for the fact that at some point in the future OPT may schedule the job J.

We consider the following cases of OPT's action when J is started by $\mathcal{A}$ at time u:

Case 1: OPT starts a job K at u, and K has either been completed by $\mathcal{A}$ on or before u, or is the same job J. In either case we do not need to give any more credit to OPT for scheduling job K as it has already been accounted for.

Any other job started by OPT in this region must be started at a time

t while $\mathcal{A}$ is idle. Consider one such job X. If X has already been scheduled in $\mathcal{A}$ before u, then a credit will have already been given to a previous region to account for it, so we do not give any credit to this region. Otherwise, X is still pending in $\mathcal{A}$ at time t, and it must be too hot to be executed by $\mathcal{A}$ at time t because otherwise $\mathcal{A}$ would have started it and therefore started a new region. So X is an OPT-only job, and from Lemma 3.2 we know that OPT can start at most E_2 OPT-only jobs before $\mathcal{A}$ starts another job. We give $\Phi^{OPT}_{[u,v)}$ one credit for each such job. This means these OPT-only jobs account for at most E_2 credits. In addition to the credit already given for J, this make $\Phi^{OPT}_{[u,v)}$ at most $E_2 + 1$, while $\Phi^{\mathcal{A}}_{[u,v)}$ is equal to 1.

Case 2: OPT does not start a job at u. In this case OPT can either start a job in the interval $[u+1, u+p)$ or not. If OPT does not start a job in this interval then again we can use Lemma 3.2 to show that $\Phi^{OPT}_{[u,v)}$ will be given at most E_2 more credits as in Case 1. If OPT does start a job K in this interval then we give $\Phi^{OPT}_{[u,v)}$ another credit for K. By the same argument as in Case 1, any other jobs started by OPT have either been scheduled by $\mathcal{A}$ before and have therefore been accounted for, or are OPT-only. These jobs (excluding K) can only be started on or after time $u + p + 1$, one time step after $\mathcal{A}$ completes J. From Lemma 3.3 it follows that there are at most E_1 such OPT-only jobs. By Lemma 3.4 we have that $E_2 \leq E_1 + 1$, meaning the total credits in $\Phi^{OPT}_{[u,v)}$ is at most $E_1 + 2$.

Case 3: OPT starts a job $K \neq J$ at u, but K has not been completed by

$\mathcal{A}$ on or before u. It must be the case that $h_K \geq h_J$, since otherwise $\mathcal{A}$ would have started the cooler job K instead of J. We consider two sub-cases. If $\tau'_u \geq \tau_u$, then it must also be the case that $\tau'_{u+p} \geq \tau_{u+p}$ as K is hotter than or equal to J. Thus it is not possible to have any OPT-only jobs in this region. So $\Phi^{\mathcal{A}}_{[u,v)} = 1$ while $\Phi^{OPT}_{[u,v)} = 2$ (one from K and one from J). The other sub-case is $\tau'_u < \tau_u$. Here there must be a previous region before this one, as in the first region both OPT and $\mathcal{A}$ has the same starting temperature of 0. We claim that the previous region has at most $E_2 - 1$ or $E_1 - 1$ (depending on which case it belongs to) OPT-only jobs. This is because, if there are E_2 (or E_1) OPT-only jobs in the previous region, then following the arguments in Lemmas 3.2 and 3.3, by the end of the previous region (which is the beginning of this region, i.e. time u) OPT must be hotter than $\mathcal{A}$, otherwise there could be more OPT-only jobs, contradicting the maximality of E_2 (or E_1). Therefore, we can count the job K to the previous region instead of this one. As in Case 1 there can be at most E_2 OPT-only jobs in this region, so $\Phi^{OPT}_{[u,v)}$ is at most $E_2 + 1$.

From the three cases, it can be seen that $\Phi^{OPT}_{[u,v)} \leq \max\{E_1 + 2, E_2 + 1\} \cdot \Phi^{\mathcal{A}}_{[u,v)}$. Lemma 3.4 shows that $E_2 \leq E_1 + 1$. This completes the proof. □

3.3.3 Upper Bound: Non-idling Algorithms

The algorithm Coolest First is a reasonable algorithm. This is because Coolest First starts a job whenever one is admissible and, as it always starts the coolest job, will never start a job that is strictly dominated by another.

We now extend this notion further and define a class of *non-idling* algorithms, defined here for the single machine case.

Definition 3.6. *An online algorithm is called non-idling if at any time step when it is not already running a job it always starts a job if there is an admissible one available.*

Note that this is a weaker notion than that of reasonable algorithms, as reasonable algorithms are non-idling algorithms, but with stricter restrictions on which job must be scheduled. Also note that a non-idling algorithm will never preempt a job by this definition.

Using a region analysis similar to Theorem 3.5 it can be shown that non-idling algorithms have a competitive ratio at most $E_2 + 2$. Note that it follows from Lemma 3.4 that this bound is no worse than 1 greater than that of Coolest First.

Theorem 3.7. *Any non-idling algorithm is* (E_2+2)*-competitive for the non-preemptive model.*

Proof. We define regions in the same way as in Theorem 3.5. Again OPT cannot schedule any job before the start of the first region as $\mathcal{A}$ is non-idling and so will schedule a job as soon as one is released.

We prove that for every region $[u, v)$ it must be that $\Phi^{OPT}_{[u,v)} \leq (E_2 + 2) \cdot \Phi^{\mathcal{A}}_{[u,v)}$. For each region we will give one credit to $\Phi^{\mathcal{A}}_{[u,v)}$ for the one job J that is completed in that region. We also give one credit to $\Phi^{OPT}_{[u,v)}$ to account for the fact that job J may be completed by OPT at some point in the future. OPT may start a job ($\neq J$) in the interval $[u, u + p)$ that cannot be started

by $\mathcal{A}$ as it is already running J at this time, so we give another credit to $\Phi^{OPT}_{[u,v)}$ to account for this.

In each region OPT may schedule any amount of jobs that have already been scheduled by $\mathcal{A}$ but these do not give any extra credit to $\Phi^{OPT}_{[u,v)}$ as a credit has already been added in advance for every job started by $\mathcal{A}$.

By Lemma 3.2 we can show that at most E_2 OPT-only jobs can be scheduled by OPT in any region. We give $\Phi^{OPT}_{[u,v)}$ one credit for each such job. So in each region $\Phi^{\mathcal{A}}_{[u,v)} = 1$ and $\Phi^{OPT}_{[u,v)}$ is at most $E_2 + 2$, thus concluding the proof. □

3.4 The Preemptive Restart Model

We now consider the model where preemptive restarts are allowed. Using standard 3-field notation this problem can be described as $1|\textit{online-r}_i, h_i, p_i = p, \textit{pmtn-restart}|\sum U_i$.

In the preemptive restart and resume models many of the competitive ratios are given in terms of E_3, defined as the largest integer $e_3 \geq 0$ such that $e_3 R(R^p - 1) > R + R^{(e_3+1)p} - 2R^p$ holds. Table 3.4 shows the value of E_3 for some combinations of R and p.

E_3	$p = 2$	$p = 3$	$p = 4$
0	$R > \sqrt{2} \approx 1.414$	$R > \sqrt[3]{2} \approx 1.260$	$R > \sqrt[4]{2} \approx 1.189$
1	$1.414 \gtrsim R \gtrsim 1.109$	$1.260 \gtrsim R \gtrsim 1.081$	$1.189 \gtrsim R \gtrsim 1.062$
2	$1.109 \gtrsim R \gtrsim 1.050$	$1.081 \gtrsim R \gtrsim 1.039$	$1.062 \gtrsim R \gtrsim 1.031$
3	$1.050 \gtrsim R \gtrsim 1.030$	$1.039 \gtrsim R \gtrsim 1.023$	$1.031 \gtrsim R \gtrsim 1.019$
4	$1.030 \gtrsim R \gtrsim 1.019$	$1.023 \gtrsim R \gtrsim 1.015$	$1.019 \gtrsim R \gtrsim 1.012$

Table 3.2: Table of E_3 values for selected R and p values.

3.4.1 Lower Bounds

Two lower bounds will be proven in this section, which in a way complement each other. The first one applies to all values of R, but is relatively weak (gives a bound of 2) when $R > \sqrt[p]{2}$. The second one only applies to the case where $R^p > R + 1$, but gives a stronger bound of 3. The root of $R^p = R + 1$ is, for large p, just a bit larger than $\sqrt[p]{2}$, so the two theorems together show a lower bound of 3 for almost all values of R. Moreover, as the non-preemptive upper bound (Theorem 3.5) is 3 when $R > \sqrt[p+1]{2}$, this means that preemption with restarts does not help for $R^p > R + 1$, which is quite a wide range for large p (for example when $p = 2$ it is equivalent to $R > 1.618$ but when $p = 10$ it becomes $R > 1.076$).

Theorem 3.8. *No deterministic algorithm for the restart model can be better than* $(E_3 + 2)$*-competitive.*

Proof. Release a job J at time 0 with a large deadline $2D + (E_3 + 4)p$ and a total heat contribution $h^*_J = R - \epsilon$, for a very small $\epsilon > 0$, where D is the time it takes for the temperature to drop from 1 to $R^{p-1}\epsilon$, i.e. the temperature at which J can start. If $\mathcal{A}$ never starts the job then the competitive ratio is infinite. So $\mathcal{A}$ will start the job at some time u. At time $u + p - 1$, $\mathcal{A}$'s temperature (ignoring the small ϵ) is given by $\tau_{u+p-1} = \sum_{i=1}^{p-1} \frac{h_J}{R^i} = \frac{R^p - R}{R^p - 1}$. At time $u + p - 1$, release a tight job K with a heat contribution so that it is just too hot to be scheduled by $\mathcal{A}$, i.e. $h^*_K > R - \frac{\tau_{u+p-1}}{R^{p-1}} = R - \frac{R - R^{2-p}}{R^p - 1}$. As K is too hot to be executed anyway, $\mathcal{A}$ will not preempt J and will continue to finish it. Meanwhile OPT does not start J and starts K. Assume OPT has temperature 0 before it starts K. After K has been completed by OPT

we have

$$\tau_{u+2p-1} = \frac{1}{R^{p-1}}, \quad \tau'_{u+2p-1} = \frac{h^*_K}{R} > 1 - \frac{1-R^{1-p}}{R^p - 1}$$

At time $u + (i+1)p - 1$ for each $i = 1, 2, ...,$ as long as $\tau_{u+(i+1)p-1} > \tau'_{u+(i+1)p-1}$ we release a tight OPT-only job J_i with a heat contribution that is just too hot for $\mathcal{A}$ to schedule, i.e. $h^*_{J_i} > R - \frac{\tau_{u+(i+1)p-1}}{R^{p-1}}$. For $i \geq 1$, the temperatures $\tau_{u+(i+1)p-1}$ and $\tau'_{u+(i+1)p-1}$ are given by

$$\tau_{u+(i+1)p-1} = \frac{1}{R^{ip-1}}, \quad \tau'_{u+(i+2)p-1} = \frac{\tau'_{u+(i+1)p-1}}{R^p} + \frac{h^*_{J_i}}{R}$$

The latter can be solved to give

$$\tau'_{u+(i+1)p-1} = \frac{R^{1-ip} + R^p - 2R^{(1-i)p}}{R^p - 1} - \frac{i-1}{R^{ip-1}}$$

The maximum number of OPT-only jobs is the largest possible i such that $\tau_{u+(i+1)p-1} > \tau'_{u+(i+1)p-1}$, which is satisfied when $iR(R^p - 1) > R + R^{(i+1)p} - 2R^p$, i.e. the same formula that defines E_3.

OPT can schedule all the OPT-only jobs plus K and J: if $u < D+p$, then OPT will finish all $K, J_1, ..., J_{E_3}$ by time $u+(E_3+1)p-1+p < D+(E_3+3)p-1$, and can still meet the deadline of J after taking D time steps of cooling time. If $u \geq D + p$, OPT will schedule J at time 0 and will become cooler than $R^{p-1}\epsilon$ (i.e. temperature almost 0) at time u, and thus can schedule all OPT-only jobs as above. $\mathcal{A}$ can only complete J. This gives a lower bound of $E_3 + 2$. □

We now prove that when R is not too small the lower bound for the problem is 3. First we give a brief outline of the proof. The lower bound

proof is split into phases, and there are several steps in each phase. Different jobs are released in the phases depending on the decisions that $\mathcal{A}$ makes, however in every phase $\mathcal{A}$ will be able to schedule 1 job while OPT will schedule 3 jobs. It may also be that at the end of a phase $\mathcal{A}$ still has a job pending. In this case we start another phase, again with jobs released such that OPT can always schedule 3 jobs in the phase while $\mathcal{A}$ can only schedule 1. If no job is pending for $\mathcal{A}$ (or any pending job for $\mathcal{A}$ will never become admissible before its deadline) we stop releasing jobs.

This process keeps repeating until it either stops as described above giving a lower bound of 3, or until a large enough number n of phases have been completed. In this case one extra job may be completed by $\mathcal{A}$ in addition to the one job from each phase $\mathcal{A}$ has already completed and we get a lower bound of $3n/(n+1)$, which can be made arbitrarily close to 3 with large enough n.

We now give two lemmas that will be used in the proof. The first gives the temperature after not quite completing a job.

Lemma 3.9. *If a job J of heat contribution h_J^* is executed for $p-1$ steps only, then immediately after its execution the temperature is at least $\frac{h_J^*}{R+1}$.*

Proof. The temperature after executing J for $p-1$ steps is at least

$$\begin{aligned} h_J \sum_{i=1}^{p-1} \frac{1}{R^i} &= h_J \left(\frac{1 - \frac{1}{R^{p-1}}}{R-1} \right) \\ &= h_J^* \left(\frac{1 - \frac{1}{R^{p-1}}}{R-1} \right) \left(\frac{R^{p-1}(R-1)}{R^p - 1} \right) \\ &= h_J^* \left(\frac{R^{p-1} - 1}{R^p - 1} \right) \end{aligned}$$

This is an increasing function of p; when $p = 2$ this is equal to $\frac{h_J^*}{R+1}$ concluding the proof. □

The following lemma is used to establish the fact that we can always create a job with a certain heat contribution, given two jobs with certain heat contributions.

Lemma 3.10. *Let h_J^* and h_K^* be the heat contributions of some jobs, where h_K^* is just smaller than $h_J^*/R - \epsilon_0/R^p$ and ϵ_0 is some small non-negative constant that can be chosen to be as small as necessary. Then there exists a heat contribution h_L^* such that*

$$R^2\left(\frac{\epsilon_0}{R^{2p+1}} + \frac{h_K^*}{R^{p+1}} - \frac{h_J}{R^{2p+1}} - \frac{1-R^{1-p}}{R^p(R-1)}h_K\right) < h_L^* < R\left(\frac{h_J}{R^{2p}} + \frac{h_K^*}{R^p} - \frac{h_J^*}{R^{p+1}} - \frac{\epsilon_0}{R^{2p}}\right)$$

Proof. We first show that

$$R^2\left(\frac{\epsilon_0}{R^{2p+1}} + \frac{h_K^*}{R^{p+1}} - \frac{h_J}{R^{2p+1}} - \frac{1-R^{1-p}}{R^p(R-1)}h_K\right) < R\left(\frac{h_J}{R^{2p}} + \frac{h_K^*}{R^p} - \frac{h_J^*}{R^{p+1}} - \frac{\epsilon_0}{R^{2p}}\right)$$

which will show that such a h_L^* exists. Using (3.1) from page 59, this inequality can be shown to be equivalent to

$$h_K^* > \frac{R^p - 2R + 1}{R^{p+1} - R^2}h_J^* + \frac{2R}{R^{p+1} - R^2}\left(1 - \frac{1}{R^p}\right)\epsilon_0$$

The fraction $(R^p - 2R + 1)/(R^{p+1} - R^2)$ is strictly smaller than $1/R$ for $p \geq 2$. Thus this can be rewritten as

$$h_K^* > \left(\frac{1}{R} - \delta\right)h_J^* + C\epsilon_0$$

for some positive constants δ and C. This can in turn be rewritten as

$$h_K^* - \left(\frac{h_J^*}{R} - \frac{\epsilon_0}{R^p}\right) > -\delta h_J^* + \epsilon_0 \left(C + \frac{1}{R^p}\right)$$

As h_K^* is just smaller than $h_J^*/R - \epsilon_0/R^p$, the inequality can be made true by choosing a sufficiently small ϵ_0.

We also need to show that h_L^* is a valid heat contribution, i.e. between 0 and R. Consider the upper limit of the range of h_L^*, which is equal to

$$\begin{aligned} R\left(\frac{h_J}{R^{2p}} + \frac{h_K^*}{R^p} - \frac{h_J^*}{R^{p+1}} - \frac{\epsilon_0}{R^{2p}}\right) &= R\left(\frac{h_J^*(R-1)}{R^{2p+1} - R^{p+1}} + \frac{h_K^*}{R^p} - \frac{h_J^*}{R^{p+1}} - \frac{\epsilon_0}{R^{2p}}\right) \\ &< R\left(\frac{h_J^*(R-1)}{R^{2p+1} - R^{p+1}} - \frac{2\epsilon_0}{R^{2p}}\right) \end{aligned}$$

This is clearly positive, and is at most

$$\frac{R^2(R-1)}{R^{2p+1} - R^{p+1}} = \frac{R^2(R-1)}{R^{p+1}(R^p - 1)} < 1$$

so h_L^* can indeed take a valid heat contribution. □

We now formally define a phase. A phase i is started at the point in time that a job J_i is released, and ends at the time that a job J_{i+1} is released, if such a job is released. For the first phase the starting time will be time 0 when J_1 will be released, while the actual time that a J_i is released for the other phases (if at all) depends on the decisions of $\mathcal{A}$ within a phase. If the job J_i is completed by $\mathcal{A}$, then no future J-jobs will be released and phase i is the final phase. The same is also true if J_i is not completed by $\mathcal{A}$ in a phase but still cannot be completed by $\mathcal{A}$ before its deadline. Otherwise J_i

will become admissible in $\mathcal{A}$ at some time s_i. We define the start of the next phase $i+1$ to be the time $t_i = s_i - (2p+2)$. We denote the temperature of OPT at the start of a phase i (i.e. at time t_{i-1}) as ϵ_i for all $i \geq 2$. For the first phase we define ϵ_1 to be some very small positive number related to the deadline of J_1 and use $\epsilon_0 = 0$ in the calculations of the temperatures of OPT and $\mathcal{A}$ as a placeholder so that extending the analysis of phase 1 to other phases is more straightforward.

We now give a lemma that details the different cases for the first phase, and shows that in this phase OPT can always schedule 3 jobs while $\mathcal{A}$ will complete only 1.

Lemma 3.11. *When $R^p > R+1$, in phase 1 OPT will always complete 3 jobs, while the most $\mathcal{A}$ will complete is 1.*

Proof. At time 0 release a job J_1 with a total heat contribution of $h^*_{J_1} = R - \epsilon_1$ for some small $\epsilon_1 > 0$, and a deadline $D_1 + 4p$ where D_1 is the time it takes to cool down from a temperature of 1 to $R^{p-1}\epsilon_1$. We choose a small enough ϵ_1 so that D_1 is much larger than p. If J_1 is not scheduled by $\mathcal{A}$ then OPT will schedule it and the competitive ratio of $\mathcal{A}$ is infinite, so we assume that $\mathcal{A}$ starts J_1 at some time u_1.

In the following we assume $\mathcal{A}$ will not preempt a currently running job unless it is to start another job (i.e. it will not preempt a job only to stay idle). If such a situation does happen, OPT will then act as if the preempted job has never been started - it can only be disadvantageous to $\mathcal{A}$ for raising its temperature without completing the job. We now consider several cases, as illustrated in Figure 3.1.

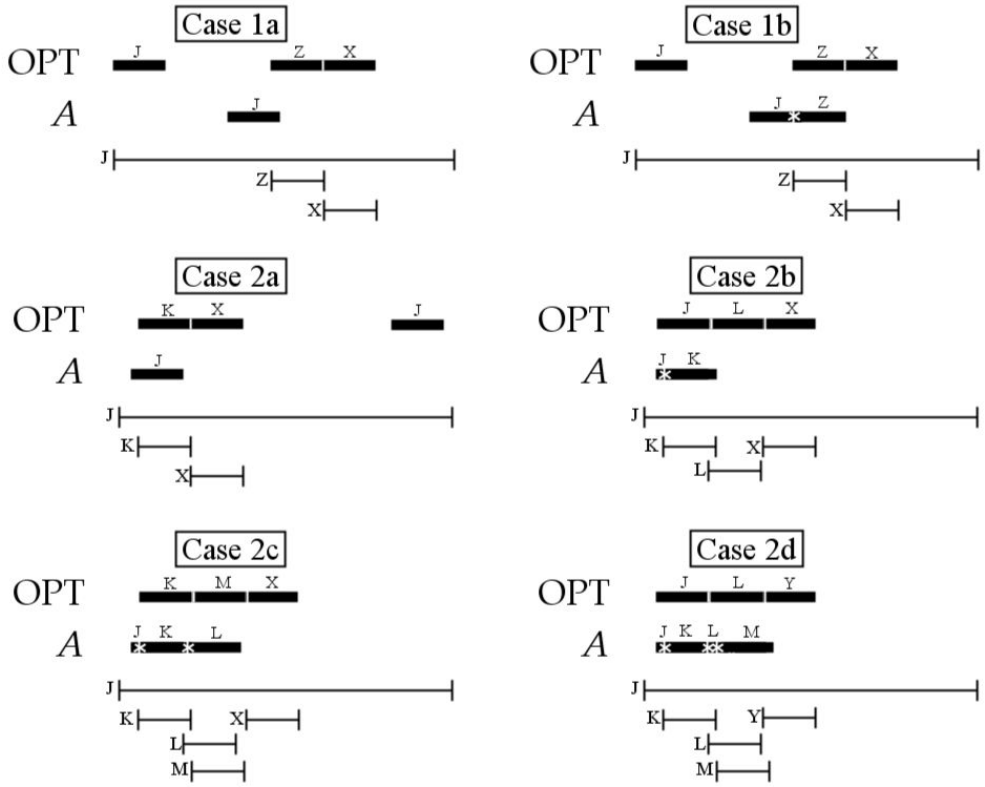

Figure 3.1: The different cases of Phase 1 in the lower bound construction.

Case 1: $u_1 \geq p$. At time $u_1 + p - 1$ we release a job Z_1 with a heat contribution of 0 and a tight deadline. Regardless of whether $\mathcal{A}$ preempts J_1 with Z_1 or not, $\mathcal{A}$'s temperature at time $u_1 + 2p - 1$ can be shown to be at least $\frac{h^*_{J_1}}{(R+1)R^p}$ by Lemma 3.9 because it executes at least $p - 1$ steps of J_1. OPT will start J_1 at time 0 and also runs Z_1, so by time $u_1 + 2p - 1$ its temperature is at most $\frac{h^*_{J_1}}{R^{2p}} + \frac{\epsilon_0}{R^{3p-1}}$. As $R^p > R + 1$, OPT is cooler than $\mathcal{A}$ for small enough ϵ_0, and we then release a job X_1 at time $u_1 + 2p - 1$ with a tight deadline and a heat contribution so that it is just too hot for $\mathcal{A}$ to schedule but OPT can schedule it. So OPT schedules 3 jobs and $\mathcal{A}$ schedules only one.

Case 1a: If $\mathcal{A}$ has not preempted J_1, then we are done.

Case 1b: If $\mathcal{A}$ has preempted J_1, it may restart the job later. This completes Phase 1 and we move to Phase 2.

Case 2: $u_1 \leq p-1$. At time u_1+1 the adversary releases a job K_1 with a tight deadline and a heat contribution $h^*_{K_1}$ that is just smaller than $\frac{h^*_{J_1}}{R} - \frac{\epsilon_0}{R^p}$.

Case 2a: Suppose $\mathcal{A}$ does not preempt J_1. Then it cannot schedule K_1. OPT will then schedule K_1, so at time u_1+p+1 its temperature is just below $\frac{\epsilon_0}{R^{u_1+p+1}} + \frac{h^*_{J_1}}{R^2} - \frac{\epsilon_0}{R^{p+1}} \leq \frac{h^*_{J_1}}{R^2}$, and is therefore cooler than $\mathcal{A}$, and so the adversary can release a job X_1 at time u_1+p+1 with a tight deadline and a heat contribution so that it is just too hot for $\mathcal{A}$ to schedule but OPT can schedule it. OPT waits D_1 steps after completing X_1 so that it is cool enough to schedule J_1. Since $(u_1+p+1)+p+D_1+p \leq D_1+4p$, OPT can complete J_1 before its deadline. OPT will have scheduled 3 jobs and $\mathcal{A}$ will have scheduled 1 and the lower bound is complete.

Case 2b: : Otherwise $\mathcal{A}$ must have preempted J_1 with K_1. In this case the adversary releases a job L_1 at time u_1+p with a tight deadline. Its heat contribution will be specified later. If $\mathcal{A}$ does not preempt K_1 then it cannot schedule L_1. In this case OPT will schedule J_1 at time u_1 and L_1 at u_1+p. We choose $h^*_{L_1}$ to be cool enough so that OPT will be cooler than $\mathcal{A}$ (which executed J_1 for

one time step and completed K_1) at $u_1 + 2p$, i.e.,

$$\frac{\epsilon_0}{R^{2p}} + \frac{h^*_{J_1}}{R^{p+1}} + \frac{h^*_{L_1}}{R} < \frac{h_{J_1}}{R^{2p}} + \frac{h^*_{K_1}}{R^p} \tag{3.9}$$

(It can be verified that it is indeed feasible for OPT to schedule both J_1 and L_1.) The adversary then releases a tight job X_1 with a heat contribution so that it is just too hot for $\mathcal{A}$ to schedule but OPT can schedule it. OPT will have scheduled 3 jobs (J_1, L_1 and X_1), while $\mathcal{A}$ will have completed 1 job, and may still be able to complete J_1 in the future. Move on to Phase 2.

If instead $\mathcal{A}$ preempted K_1 with L_1, we consider it in the next case.

Case 2c: In this case the adversary releases a job M_1 at time u_1+p+1 with a heat contribution of 0 and a tight deadline. If $\mathcal{A}$ does not preempt L_1 then it cannot schedule M_1. In this case OPT will schedule K_1 and M_1. L_1 has to be hot enough so that OPT is cooler than $\mathcal{A}$ (which has executed J_1 for one time step, then K_1 for $p-1$ time steps, then L_1) at time $u_1 + 2p + 1$, i.e.

$$\frac{\epsilon_0}{R^{2p+1}} + \frac{h^*_{K_1}}{R^{p+1}} < \frac{h_{J_1}}{R^{2p+1}} + \frac{h_{K_1}(1 - R^{1-p})}{R^p(R-1)} + \frac{h^*_{L_1}}{R^2} \tag{3.10}$$

Lemma 3.10 shows that an $h^*_{L_1}$ that satisfies both (3.9) and (3.10) does exist.

The adversary then releases a tight job X_1 at time $u_1 + 2p + 1$ with a heat contribution $h^*_{X_1}$ so that it is just too hot for $\mathcal{A}$ to

schedule but OPT can schedule it. In this case OPT will have scheduled 3 jobs and $\mathcal{A}$ will have completed 1 job, and still be able to complete J_1 in the future. Move on to Phase 2.

If instead $\mathcal{A}$ preempted L_1 with M_1, we consider it in the next case.

Case 2d: In this case the adversary releases a job Y_1 at time $u_1 + 2p$ with a heat contribution of 0 and a tight deadline. $\mathcal{A}$ can either preempt M_1 or not but either way only one of M_1 or Y_1 can be completed by $\mathcal{A}$. OPT will complete J_1, L_1 and Y_1. In this case OPT will have scheduled 3 jobs and $\mathcal{A}$ will have only completed 1 job, and may still be able to complete J_1 in the future. Again we will deal with this in Phase 2.

Note that in all of the cases, in this phase $\mathcal{A}$ has only completed 1 job while OPT has completed 3. □

We now give a lemma that establishes some properties that will be needed to complete the proof over all the necessary phases.

Lemma 3.12. *Consider cases 1b, 2b, 2c and 2d where J_1 may restart later. Let s_1 be the earliest time when $\mathcal{A}$ becomes cool enough to schedule J_1 again (i.e. temperature at most $R^{p-1}\epsilon_1$). Define a time $t_1 = s_1 - (2p + 2)$ and let $\epsilon_2 = \tau'_{t_1}$. Then*

(1) $s_1 \geq D_1 - 2$

(2) $\tau_{t_1} < \tau'_{t_1} \leq R^{3p+2}\tau_{t_1}$

(3) $\epsilon_1 < \epsilon_2 < R^{6p+3}\epsilon_1$.

Proof. (1) Consider cases 1b, 2b, 2c and 2d where J_1 may be restarted later by $\mathcal{A}$.

In case 1b, $\mathcal{A}$ executes $p-1$ steps of J_1, so its temperature when it preempts J_1 (i.e. at time u_1+p-1) is, by Lemma 3.9, at least $h^*_{J_1}/(R+1) > h^*_{J_1}/R^p$ (recall we have assumed $R^p > R+1$). The number of cooling steps required before it can run J_1 again is then at least

$$\left\lceil \log_R \left(\frac{h^*_{J_1}/R^p}{R^{p-1}\epsilon_1} \right) \right\rceil = \left\lceil \log_R \left(\frac{h^*_{J_1}}{R^{p-1}\epsilon_1} \right) \right\rceil - p = \left\lceil \log_R \left(\frac{R-\epsilon_1}{R^{p-1}\epsilon_1} \right) \right\rceil - p$$

which is at least $D_1 - p$, because by our definition of D_1, we have that $\lceil \log_R(R/R^{p-1}\epsilon_1) \rceil = D_1 + 1$. Thus $s_1 \geq (p-1) + (D_1 - p) = D_1 - 1$.

In cases 2c and 2d, $\mathcal{A}$ executes one step of J_1 and $p-1$ steps of K_1, so its temperature when it preempts K_1 (i.e. at time u_1+p) is, by Lemma 3.9, at least $h^*_{K_1}/(R+1) \approx h^*_{J_1}/R(R+1) > h^*_{J_1}/R^{p+1}$. The number of cooling steps required before it can run J_1 again is then at least

$$\left\lceil \log_R \left(\frac{h^*_{J_1}/R^{p+1}}{R^{p-1}\epsilon_1} \right) \right\rceil - 1 = \left\lceil \log_R \left(\frac{h^*_{J_1}}{R^{p-1}\epsilon_1} \right) \right\rceil - (p+1) - 1 \geq D_1 - p - 2$$

(Here we subtracted one time step to account for the fact that $h^*_{K_1}$ is slightly smaller than $h^*_{J_1}/R$ and thus it may require one fewer step to cool.) Thus $s_1 \geq p + (D_1 - p - 2) = D_1 - 2$.

In case 2b, $\mathcal{A}$ executes one step of J_1 and p steps of K_1, so its temperature when it completes K_1 (i.e. at time u_1+p+1) is at least $h^*_{K_1}/R$, and with similar calculations as cases 2c and 2d we can show that s_1 is again at least

$(p+1)+(D_1-p-2)=D_1-1$.

(2) Let t_{OPT} be the time when OPT finishes its last job in Phase 1. It can be verified that at time t_{OPT}, OPT must be hotter than $\mathcal{A}$. (In fact, if it is not the case, then we can simply release more just-too-hot jobs like X_1 that OPT can schedule but $\mathcal{A}$ cannot, making the competitive ratio even worse.) This remains true until time t_1 because until then both OPT and $\mathcal{A}$ are idle.

In the proof of (1) above, it was shown that when $\mathcal{A}$ preempts J_1 (case 1b) or completes K_1 (case 2b) or preempts K_1 (cases 2c, 2d), its temperature is always at least $h^*_{J_1}/R^{p+1}$ (i.e. just under $1/R^p$). At time t_{OPT} , OPT's temperature is at most 1. So, comparing these two points of time, OPT is hotter by at most a factor that is just larger than R^p; for convenience take the factor to be R^{p+1}. Also, in all these cases, there are at most $2p+1$ time steps between t_{OPT} and the time $\mathcal{A}$ finishes/preempts J_1/K_1, so $\mathcal{A}$ has at most $2p+1$ extra idle steps to cool down, so at any time after t_{OPT} , OPT's temperature is hotter than $\mathcal{A}$ (at the same time instant) by at most a factor of R^{3p+2}.

(3) Observe that $R^{p-2}\epsilon_1 < \tau_{s_1} \leq R^{p-1}\epsilon_1$ as s_1 is the earliest time that is cool enough. The first inequality is true because $\epsilon_2 = \tau'_{t_1} > \tau_{t_1} = R^{2p+2}\tau_{s_1} > R^{2p+2}R^{p-2}\epsilon_1 > \epsilon_1$. The second inequality is true because $\epsilon_2 = \tau'_{t_1} < R^{3p+2}\tau_{t_1}$ (from (2)) $= R^{3p+2}R^{2p+2}\tau_{s_1} \leq R^{5p+4}R^{p-1}\epsilon_1 = R^{6p+3}\epsilon_1$. □

Using these properties we now construct the full proof over all phases to prove the lower bound.

Theorem 3.13. *Every deterministic algorithm for the preemptive restart model is at least 3-competitive when $R^p > R+1$.*

Proof. Lemma 3.11 shows that in phase 1 OPT always completes 3 jobs while $\mathcal{A}$ completes 1. If J_1 has not been completed by $\mathcal{A}$ during phase 1 then we start the next phase.

Phase 2. If $s_1 > d_{J_1} - p$ then $\mathcal{A}$ cannot schedule J_1 so we have achieved a ratio of 3 and we stop here. Otherwise Phase 2 begins at time t_1. Let ϵ_2 be the temperature of OPT at this point. We release a job J_2 at time $t_1 = s_1 - (2p + 2)$, with deadline $t_1 + D_2 + 4p$ and a heat contribution of $h^*_{J_2} = R - \epsilon_2$, where D_2 is the time it takes to cool down from a temperature of 1 to $R^{p-1}\epsilon_2$. Note that OPT is cool enough to start J_2 at this point if it so wishes. Note also that t_1 is after the deadlines of all the phase 1 jobs except J_1, as D_1 is much larger than p. Therefore, scheduling of Phase 1 jobs (except J_1) does not interfere with the scheduling of Phase 2 jobs.

$\mathcal{A}$ cannot schedule both J_1 and J_2: if $\mathcal{A}$ schedules J_1 at some time $\geq s_1$, then J_2 cannot start until at least $D_2 - 1$ steps after $s_1 + p$, i.e. time $s_1 + p + D_2 - 1$, but the deadline of J_2 is $t_1 + D_2 + 4p = s_1 + D_2 + 2p - 2$, so J_2 cannot finish before its deadline. If on the other hand J_2 is scheduled first, on or after t_1, then J_1 must wait at least $D_1 - 1$ steps after J_2 finishes, i.e. J_1 starts earliest at $t_1 + p + (D_1 - 1) = (s_1 - 2p - 2) + D_1 + p - 1 = s_1 + D_1 - 3 - p \geq 2D_1 - 5 - p$ (from Lemma 3.12 (1)) but its deadline is $D_1 + 4p$. It can therefore be assumed without loss of generality that $\mathcal{A}$ will not schedule J_1, as J_2 is cooler and has a later deadline.

$\mathcal{A}$ must schedule J_2 at some time u_2; then we proceed as in Phase 1 and

Lemma 3.11, following cases 1a - 2d as necessary (replacing J_1, Z_1, K_1, L_1, M_1, X_1, Y_1 with J_2, Z_2, K_2, L_2, M_2, X_2, Y_2; ϵ_0 with ϵ_2; and change all job release times to be relative to time t_1). In this phase OPT can complete 3 jobs while $\mathcal{A}$ can complete at most one, and possibly have J_2 still pending at the end of the phase, in which case we move to Phase 3.

Phase $i \geq 2$**.** In general, after Phase $(i-1)$, if J_{i-1} is not pending then the construction ends. Otherwise, let s_{i-1} be the earliest time that $\mathcal{A}$ is cool enough to restart J_{i-1}. If $s_{i-1} > d_{J_{i-1}} - p$ then again the construction ends. Otherwise define a time $t_{i-1} = s_{i-1} - (2p+2)$ and Phase i begins at time t_{i-1}. Let ϵ_i be OPT's temperature at this point. We release J_i with a deadline of $t_{i-1} + D_i + 4p$ and a heat contribution of $h^*_{J_i} = R - \epsilon_i$, where D_i is the time it takes to cool down from a temperature of 1 to $R^{p-1}\epsilon_i$. Similar to the discussion in Phase 2, we can assume $\mathcal{A}$ will not schedule J_{i-1}, and cases 1a - 2d will be followed.

This process continues until either $\mathcal{A}$ does not have some J_i pending that will become admissible before $d_{J_i} - p$, or Phase n is reached for a large enough n in which case we stop. In the latter case the competitive ratio is $3n/(n+1)$ as OPT schedules 3 jobs in every phase whereas $\mathcal{A}$ schedules one job in every phase except Phase n where it schedules 2.

Similar to Lemma 3.12 (3) we can prove that $\epsilon_i < \epsilon_{i+1} \leq R^{6p+3}\epsilon_i$ for all i. Thus by choosing a small enough ϵ_1 we can ensure that all other ϵ_i are also small enough and the D_i are large enough for the construction to work for any large enough number of phases n. This concludes the proof as the ratio

can be made arbitrarily close to 3 for large n. □

3.5 The Preemptive Resume Model

We now consider the model where preemptive resumes are allowed. Using standard 3-field notation this problem is defined as $1|\textit{online-}r_i, h_i, p_i = p, \textit{pmtn}|\sum U_i$.

Potentially, the preemptive resume model can allow for a much hotter job to be executed than the non-preemptive or restart models: even if the heat contribution per step (h_J) of a job is almost R, it can still be scheduled by taking a long enough idle time in-between any two scheduled steps of the job. However, it can be shown that such a model leads to bad results.

Theorem 3.14. *When the maximum heat contribution of a job per time step is $R - \epsilon$ for some sufficiently small $\epsilon > 0$, no deterministic algorithm for the resume model is better than pE_2-competitive, where E_2 is as defined in Lemma 3.2.*

Proof. Fix some deterministic algorithm $\mathcal{A}$. Release a job J at time 0 with $h_J = R - \epsilon$ for some small $\epsilon > 0$ and a very large deadline $p + pD$; D is the number of time steps it takes for a system to cool from $1 - \epsilon/R$ to ϵ. We choose a small enough ϵ so that D is much larger than E_2. Because it is so hot, J can only be scheduled by splitting it into p unit-sized pieces, each separated by at least D time steps in the schedule.

If $\mathcal{A}$ does not schedule J at all then OPT does and the competitive ratio will be infinite. Therefore $\mathcal{A}$ starts the first piece of J at some time u_1. OPT

remains idle until $u_1 + 1$ and then we release E_2 OPT-only jobs in the same way as in the proof of Lemma 3.2. For a sufficiently small ϵ, these jobs can all be scheduled by OPT but cannot be scheduled by $\mathcal{A}$.

Let D' be the number of cooling time steps needed for OPT to cool down after processing the last of the E_2 jobs to start the same sequence of E_2 OPT-only jobs. The value of D' depends only on the heat contributions of the E_2 OPT-only jobs. A smaller ϵ means $\mathcal{A}$ will be hotter and therefore these OPT-jobs can be less hot, implying that D' will be slightly smaller. At the same time, a smaller ϵ implies a larger D. Thus it is always possible to choose a small enough ϵ so that $D > D' + E_2$, i.e. $\mathcal{A}$ must remain idle while OPT processes these E_2 jobs and cools down.

After $\mathcal{A}$ becomes cool enough, it must schedule the second piece of J at some time u_2 or else again its competitive ratio is infinite. The process is then repeated ($\mathcal{A}$ is slightly hotter this time because of a non-zero starting temperature, but we can still use the same set of E_2 jobs for OPT). In the end, $\mathcal{A}$ will only be able to schedule J to completion while OPT will schedule E_2 jobs after every piece of J scheduled by $\mathcal{A}$ thus giving a competitive ratio of pE_2. □

Therefore, in the following we assume the maximum permissible heat contribution of a job is the same as the non-preemptive/restart case, i.e. $h_J^* \leq R$. This also allows a fairer comparison between different models of preemption.

We now prove a lower bound that is precisely one smaller than the corresponding lower bound for the restart model.

Theorem 3.15. *No deterministic algorithm for the resume model can have a competitive ratio better than $E_3 + 1$.*

Proof. The proof is very similar to that of Theorem 3.8. Fix a deterministic algorithm $\mathcal{A}$. At time 0 release a job J with a total heat contribution $h_J^* = R$, and a tight deadline of p. We assume that $\mathcal{A}$ schedules J for p consecutive time slots in $[0, p)$ as otherwise it schedules no job at all and gives an infinite competitive ratio. After $p-1$ steps the temperature of $\mathcal{A}$ is given by $\tau_{p-1} = \sum_{i=1}^{p-1} \frac{h_J}{R^i} = \frac{R^p - R}{R^p - 1}$. We then release a tight job K with a heat contribution that is just too hot for $\mathcal{A}$ to schedule, i.e $h_K^* > R - \frac{\tau_{p-1}}{R^{p-1}} = R - \frac{R - R^{2-p}}{R^p - 1}$. Therefore $\mathcal{A}$ will continue with and complete J. Meanwhile OPT will not schedule J and so can schedule K. The rest of the proof is the same as Theorem 3.8 i.e. for $i \geq 1$, as long as $\tau_{(i+1)p-1} > \tau'_{(i+1)p-1}$ we release an OPT-only job J_i at time $(i+1)p - 1$. The maximum number of OPT-only jobs is thus given by the same formula that defines E_3. We then have a lower bound of $E_3 + 1$ as OPT can complete the E_3 OPT-only jobs plus K while $\mathcal{A}$ can only complete J. □

Note that for $R > \sqrt[p]{2}$, the above theorem gives a trivial lower bound of 1. We can give a different lower bound (that works for any $R > 1$):

Theorem 3.16. *No deterministic algorithm for the resume model can have a competitive ratio better than 2.*

Proof. Fix a deterministic algorithm $\mathcal{A}$. Release a job J at time 0 with a total heat contribution $h_J^* = R - \epsilon$ for some very small $\epsilon > 0$. This gives $h_J = \frac{(R-1)(R-\epsilon)R^{p-1}}{R^p - 1}$. Job J has a deadline of $2D + 3p$, where D is the number of time steps it takes for the machine to cool down from a temperature of 1

to $R^{p-1}\epsilon$. Again we can assume $\mathcal{A}$ starts the job at some time u; if it does not start J at all then the competitive ratio would be infinite.

If $u \geq D + p$ then OPT will schedule J in the p consecutive time steps starting at time 0. This means that OPT will have cooled down to a temperature of at most $R^{p-1}\epsilon$ by time $u + 1$. At time $u + 1$ the temperature of $\mathcal{A}$ is equal to (ignoring the tiny value of ϵ) $\tau_{u+1} = \frac{h_J}{R} = \frac{(R-1)R^{p-1}}{R^p-1}$. At time $u+1$ the adversary releases a job K with a tight deadline and a heat contribution that makes it just too hot to be scheduled by $\mathcal{A}$ but can be scheduled by OPT. This requires the heat contribution of K to be $h^*_K > R - \frac{\tau_{u+1}}{R^{p-1}} = R - \frac{R-1}{R^p-1}$. As this job has a tight deadline $\mathcal{A}$ cannot ever schedule it even if it preempts J, so $\mathcal{A}$ will only be able to schedule J. Meanwhile OPT can schedule both K and J. This gives a competitive ratio of 2 for this case.

Otherwise $u < D + p$. In this case OPT will remain idle until $u + 1$ when K will be released as in the previous case. OPT will complete K and then wait at most D time steps and schedule J. OPT will therefore finish J at the latest by time $(u + 1) + p + D + p \leq (D + p) + p + D + p = 2D + 3p$, meeting its deadline. Again this means that OPT has scheduled J and K while $\mathcal{A}$ can only schedule J, giving a competitive ratio of 2 and concluding the proof. □

3.6 Summary

In this chapter we considered maximising the throughput of equal length, unweighted jobs on a single machine. We show that an algorithm is optimal for all cooling factors in the non-preemptive case, and that the class of reason-

able algorithms have a competitive ratio of at most 1 larger than the optimal competitive ratio. These competitive ratios increase when R decreases, and also as the job length p decreases.

We then analyse the extra power that preemption can provide. We show that for a large range of cooling factors allowing preemptive restarts cannot reduce the competitive ratio over the non-preemptive case. We also show that allowing preemptive resumes can only reduce the competitive ratio by at most 1 over the preemptive restarts model, for all cooling factors.

Chapter 4

Variable Length Throughput

We now extend the equal length job problem to consider the case where jobs may have any integer length. We consider two different models for this problem, the first is the unweighted model in which an algorithm gets the same value for completing each job regardless of the job's characteristics. The second model is the proportionally weighted model, where the value an algorithm gets for completing each job is the same as the length of the job i.e. $\forall J : w_J = p_J$.

We will denote the longest job length as L throughout the chapter. All of the results in this chapter are upper and lower bounds that are linear in terms of L. The results given are for the one machine case for all $R > 1$.

4.1 Motivation

Considering jobs of arbitrary lengths is a natural extension of the equal length job problem. Until now we have been considering jobs that are modelled as

equal length slices of jobs, while in reality the complete jobs are likely to be of differing lengths. Therefore to more accurately study real life scenarios, it is useful to investigate the effect of allowing a variety of lengths, and discover how this affects the quality of a schedule.

As with the non-unit length, equal length jobs scheduling of variable length jobs may involve preemption. Analysing the potential power of preemption will again be useful, as there are various applications where preemption is either not possible, or very costly.

4.2 Previous Results

There are no previous results for either the unweighted or proportionally weighted models with temperature. Without temperature, in the unweighted case with no preemption it is well known that no algorithm can have a competitive ratio of better than L_R, where L_R is the ratio of the longest to shortest job length. The unweighted case with preemptive restarts has been studied in [32]. They give an algorithm with an upper bound of 2-competitive, and a matching lower bound.

With the unweighted case with preemptive resumes it has been shown that every deterministic algorithm has a non constant competitive ratio, implied from the proof to be $\Omega(\frac{\log L_R}{\log \log L_R})$ [8]. It can be shown that for this model the well known algorithm SRPT is $\Theta(\log L_R)$-competitive [36]. The offline version of the problem is solvable in polynomial time, using a dynamic programming algorithm [41]. In [8] they also give several special instances of the problem (e.g. 2-competitive tight bounds for the case of monotonic dead-

lines) and corresponding deterministic algorithms with constant competitive ratios. A randomised algorithm for the problem is given in [36] which has a very large, but constant competitive ratio.

For the proportionally weighted model without temperature, if preemptive resumes are allowed there is a 4-competitive deterministic algorithm [9], and this is also shown to be optimal. For the non-preemptive version of this problem an algorithm has been shown with a competitive ratio of $\frac{2C_{min}+C_{max}+2}{C_{min}+2}$ [24] where C_{max} and C_{min} are the largest and smallest job lengths. Several results for various restrictions on the general problem are also given in [24].

4.3 Unweighted Jobs

We first give some results for the unweighted job version of the problem. Using standard 3-field notation this problem is defined as $1|\mathit{online}\text{-}r_i, h_i, p_i|\sum U_i$. In this section we give an upper bound on non-idling algorithms, and then show lower bounds that are almost matching for both the non-preemptive and preemptive cases.

4.3.1 Upper Bound: Non-idling Algorithms

We now analyse non-idling algorithms, as defined in Section 3.3.3. In this section we give bounds in terms of U as defined in Chapter 2. Recall the definition of U as the largest $u \geq 1$ such that $R^u < (u+1)R - u$.

We first give a lemma that proves that U is the maximum number of OPT-only jobs that OPT can schedule while $\mathcal{A}$ remains idle.

Lemma 4.1. *Consider a time step when $\mathcal{A}$ is idle. U is the maximum number of OPT-only jobs that OPT can schedule from this idle time step until $\mathcal{A}$ starts a job.*

Proof. It can easily be established from examining the analysis in Lemma 2.6 and Theorem 2.7 that U is the maximum number of OPT-only jobs that can be scheduled before $\mathcal{A}$ starts a job, if all of them are of unit length. We show that the same holds if some of these jobs are longer.

Consider an OPT-only job J that is scheduled by OPT at some time t and has a length $p_J \geq 2$. Consider an alternative schedule OPT' that is identical to OPT up to time t, and that the time interval $[t, t+p_J)$ is replaced by a time interval $[t, t+1)$ that schedules a unit length OPT-only job J'. The rest of the OPT' schedule is identical to the rest of OPT moved forward, i.e. OPT' in $[t+1, \infty)$ is the same as OPT in $[t+p_J, \infty)$.

Let τ''_t be the temperature of OPT' at time t. By definition of OPT-only jobs it must be that $h_{J'} > R - \tau_t$ and $h^*_J > R - \tau_t/R^{p_J-1}$. We can assume all OPT-only jobs take their minimum permissible heat contributions as this maximises the number of OPT-only jobs. Therefore, if we ignore the slight extra heat contribution of each job, we have

$$\tau'_{t+p_J} = \tau'_t/R^{p_J} + 1 - \tau_t/R^{p_J}; \quad \tau''_{t+1} = \tau'_t/R + 1 - \tau_t/R.$$

We show that $\tau'_{t+p_J} \geq \tau''_{t+1}$. This is true if and only if $\tau'_t/R^{p_J} + 1 - \tau_t/R^{p_J} \geq \tau'_t/R + 1 - \tau_t/R$ which is equivalent to $(R^{p_J} - R)(\tau_t - \tau'_t) \geq 0$. This must be true because (1) $R > 1$ and $p_J \geq 2$; and (2) for an OPT-only job to exist at t it must be that $\tau'_t < \tau_t$. Therefore $\tau'_{t+p_J} \geq \tau''_{t+1}$, thus OPT' is still a feasible

schedule i.e. the jobs in OPT after $t + p_j$ can still be scheduled in OPT' after $t + 1$. As for $\mathcal{A}$, it remains idle in both cases so it is unaffected. This means that $\tau_{t+p_J} = \tau_t / R^{p_J}$ and $\tau_{t+1} = \tau_t / R$, so $\tau_{t+1} > \tau_{t+p_J}$. Therefore the other OPT-only jobs after J' in OPT' remain OPT-only as the temperature at the corresponding time in $\mathcal{A}$ is higher.

We can similarly replace all other non-unit length OPT-only jobs in OPT to create a schedule OPT' so that all OPT-only jobs in OPT' are of unit length. OPT' is a feasible schedule containing the same number of OPT-only jobs as OPT but all of them are of unit length. As we already know that the number of OPT-only jobs in OPT' is at most U, the number of OPT-only jobs in OPT is also at most U. □

Now we show an upper bound on all non-idling algorithms.

Theorem 4.2. *All non-idling algorithms are* $(L + U + 1)$*-competitive.*

Proof. We analyse the algorithms using a charging scheme, where every job completed by OPT generates a charge to a job completed by $\mathcal{A}$. We then show that every job completed by $\mathcal{A}$ can receive at most $L + U + 1$ charges thus proving the competitiveness of $\mathcal{A}$.

Every job J completed by $\mathcal{A}$ receives a charge from each of the jobs that is started by OPT while $\mathcal{A}$ is running J. We use 3 types of charge.

Type-A Charge: As non-idling algorithms do not preempt any jobs we know that J will always complete and so will always be able to receive the charges from these jobs. As J will have a length at most L, J can receive at most L charges of this type.

Type-B Charge: The next type of charge will be generated by jobs that have yet to be completed by $\mathcal{A}$ and are completed by OPT after $\mathcal{A}$ has completed job J, but before $\mathcal{A}$ starts another job. As $\mathcal{A}$ is non-idling, any job that is admissible would be started immediately and as the jobs that generate these charges are not started by $\mathcal{A}$ (by definition of the charge) they must be too hot to be admissible for $\mathcal{A}$, and therefore OPT-only. The maximum number of these jobs that can be completed by OPT while $\mathcal{A}$ is idle is shown by Lemma 4.1 to be U.

Type-C Charge: The final type of charge that J can receive is from a copy of itself, if it is completed by OPT later. It is clear that each job can only receive one of these charges.

These 3 charge types cover all possible jobs that could be scheduled by OPT and so every job completed by OPT generates a charge while every job completed by $\mathcal{A}$ can receive at most $L + U + 1$ charges, thus concluding the proof. □

4.3.2 Lower Bounds

We now show some lower bounds for deterministic algorithms for the unweighted job problem. First we give a bound for the model without preemption that is only 1 below the upper bound for any R and L values.

Without Preemption

Theorem 4.3. *No non-preemptive deterministic algorithm is better than* $(L + U)$*-competitive.*

Proof. Fix some deterministic algorithm $\mathcal{A}$ that does not use preemption. Release a job J with a total heat contribution $h_J^* = R - \epsilon$ for some small $\epsilon > 0$, a processing time of L, and a deadline of $2D + U + 3L$ for some large D. D must be large enough so that a system can cool to ϵ after a starting temperature of 1 and D idle time steps. If $\mathcal{A}$ never schedules J then OPT does and $\mathcal{A}$ has an unbounded competitive ratio. Otherwise $\mathcal{A}$ schedules J at some time u.

Case 1: If $u \leq D + L$ then OPT does not start J yet. At each time step $u + i$ for $i = 1, 2, ..., L - 1$ we release a job K_i with a heat contribution of 0, a processing time of 1 and a tight deadline of $u + i + 1$. OPT will schedule all of these jobs while $\mathcal{A}$ will not be able to as it has already scheduled J and is non-preemptive. At each time step $u + L + i - 1$ for $i = 1, 2, ..., U$ we will release an OPT-only job with a processing time of 1. This is done in the same way as the OPT-only jobs are released in Theorem 2.8, with the proof of that theorem being sufficient to show that we will indeed be able to release a maximum number U of these jobs. OPT will then remain idle for D time steps and then schedule J (for L time steps). This will always be possible before J's deadline as $u + U + D + 2L \leq 2D + U + 3L$. In this case OPT will have completed $L - 1$ of the K jobs, U OPT-only jobs and J, while $\mathcal{A}$ will only complete J, thus giving a competitive ratio of $L + U$.

Case 2: Otherwise $u > D + L$ and OPT starts J as soon as it is released. As $u > D + L$ once $\mathcal{A}$ starts J the temperature of OPT will be at most ϵ which can be made arbitrarily close to 0. As before we release the

$L-1$ K jobs followed by U OPT-only jobs. OPT will schedule all of these jobs while $\mathcal{A}$ can still only schedule J, thus concluding the proof.

□

With Preemption

Now we give lower bounds for the model where preemption is allowed. The bounds hold for both types of preemption. The bounds for this case are linear in L, as are the bounds without preemption. This shows that given a fixed R value, preemption can only give at most a constant factor improvement in competitive ratio.

The following term is used in the bound:

$$\log_R\left(\frac{R}{R-1}\right)$$

This number is between 0 and 1 whenever $R \geq 2$ and gets larger as R decreases, becoming infinity when $R = 1$. Table 4.1 gives some values for this term with corresponding R values.

$\log_R\left(\frac{R}{R-1}\right)$	R
≤ 1	$R \geq 2$
≤ 2	$R \geq 1.618$
≤ 3	$R \geq 1.466$
≤ 4	$R \geq 1.380$
≤ 5	$R \geq 1.325$
≤ 10	$R \geq 1.197$
≤ 100	$R \geq 1.035$

Table 4.1: Table of $\log_R\left(\frac{R}{R-1}\right)$ values.

Theorem 4.4. *No deterministic algorithm with preemption is better than* $\Omega\left(L - \log_R\left(\frac{R}{R-1}\right)\right)$*-competitive, for any* $L > \log_R(\frac{R}{R-1})$.

Proof. Fix some deterministic algorithm $\mathcal{A}$ that uses preemption. Release a job J at time 0 with $h_J^* = R$, $p_J = L$ and $d_J = L$. If $\mathcal{A}$ does not start J as soon as it is released then it cannot ever start J, and in this case OPT will complete J and the competitive ratio of $\mathcal{A}$ will be unbounded. So $\mathcal{A}$ starts J as soon as it is released and OPT stays idle at time 0.

At time 1 we release a job K_1 with $p_{K_1} = 1$, a tight deadline of 2, and a heat contribution that makes it just too hot for $\mathcal{A}$ to schedule K_1 even if $\mathcal{A}$ were to preempt J. $\mathcal{A}$ therefore cannot schedule K_1. OPT will schedule K_1 then remain idle until it is cooler than $\mathcal{A}$. As soon as OPT is cooler than $\mathcal{A}$ we release another OPT-only job K_2 with a processing time of 1 and a tight deadline that is just slightly too hot to be scheduled by $\mathcal{A}$. We continue this process, idling until OPT is cooler than $\mathcal{A}$ then releasing an OPT-only job K_i with a processing time of 1 and a tight deadline. As each job K_i is only slightly too hot to be scheduled by $\mathcal{A}$, and OPT is cooler than $\mathcal{A}$, it is always possible to choose a heat contribution for K_i such that OPT can schedule these jobs.

We now show that after $\log_R(\frac{R}{R-1})$ time steps, exactly 1 K-job can be scheduled by OPT every 2 time steps. The temperature of OPT after scheduling each K-job will always be at most 1, and therefore after an idle step following a K-job the temperature of OPT will be at most $1/R$. We now calculate the temperature of $\mathcal{A}$ after completing x time steps of J. As $h_J^* = R$ and $p_J = L$ this gives $h_J = \frac{(R-1)R^L}{R^L-1} > R - 1$ meaning that at time x the temperature of $\mathcal{A}$ must be at least $1 - 1/R^x$. Therefore it is always

possible to release an OPT-only K-job every 2 time steps after some time x as long as $1/R < 1 - 1/R^x$, which is true whenever $x > \log_R(\frac{R}{R-1})$. Therefore the total number of jobs that OPT can schedule is at least $\frac{L - \log_R(\frac{R}{R-1})}{2}$, while $\mathcal{A}$ can only ever complete J. □

Note that as R decreases, so does the value of $L - \log_R\left(\frac{R}{R-1}\right)$. On the other hand when R decreases the value of $U + 1$ increases. This means that for the case where $U + 1 > L - \log_R\left(\frac{R}{R-1}\right)$ we can use the lower bound for unit length jobs as in Theorem 2.8 to show that no deterministic algorithm can be better than $(U+1)$-competitive. In this bound all jobs are unit length meaning that no preemption will be possible, therefore this bound also holds for all preemption models.

We can combine Theorem 4.4 and 2.8 to show a lower bound of

$$\max\left\{\Omega\left(L - \log_R\left(\frac{R}{R-1}\right)\right), \Omega(U)\right\}$$

$$= \Omega\left(L - \log_R\left(\frac{R}{R-1}\right) + U\right).$$

It follows from Theorem 4.4, that for a fixed R, the lower bound is only dependent on L.

Corollary 4.5. *For a fixed R, no deterministic algorithm can be better than $\Omega(L)$-competitive.*

As given in Theorem 4.2, the upper bound without preemption is $L + U + 1$. If we assume a constant R value this becomes $O(L)$, therefore we have shown that for a fixed R allowing preemption can only give a constant factor improvement over the non-preemptive case.

4.4 Proportionally Weighted Jobs

We now give some results for the proportionally weighted job version of the problem. Using standard 3-field notation this problem is defined as $1|\textit{online-}r_i, h_i, p_i \geq 1|\sum p_i U_i$. The bounds in this proof are expressed in terms of a value $V = \lceil \log_R 2 \rceil - 2$ that increases as R decreases. We give a lower bound of $L + V + 1$, that works for the cases with and without preemption. We also show a non-preemptive upper bound that matches when $V \geq 1$, and when $V = 0$ is almost matching. This shows that preemption cannot significantly reduce the competitive ratio of algorithms in this model.

4.4.1 Lower Bound

We now provide a lower bound for deterministic algorithms. Note that this lower bound can apply to the models both with and without preemption.

Theorem 4.6. *No deterministic algorithm is better than* $(1+V+L)$*-competitive.*

Proof. Fix some deterministic algorithm $\mathcal{A}$. At time 0 release a job J with $h_J^* = R - \epsilon$ for some small $\epsilon > 0$, $p_J = 1$ and $d_J = V + L + 2D + 2$ where D is the number of time steps it takes a system to cool from 1 to ϵ. $\mathcal{A}$ will start J at some time u. We consider two cases.

Case 1: $\mathcal{A}$ starts J at some $u \leq D$. OPT will remain idle at and before u. At time $u + 1$ we release a job K that has the smallest possible heat contribution such that $h_K^* > R - \frac{R-\epsilon}{R^V}$, $p_K = V$ and a tight deadline. (If $V = 0$ we omit this job.) $\mathcal{A}$ will be just too hot to schedule K, making it OPT-only, and so will idle for the next V time steps, while OPT

will schedule K. After these V time steps we will have $\tau_{u+1+V} = \frac{R-\epsilon}{R^{V+1}}$ and $\tau'_{u+1+V} > 1 - \frac{R-\epsilon}{R^{V+1}}$. This means that we have $\tau_{u+1+V} > \tau'_{u+1+V}$ if $V < \log_R(2 - 2\epsilon/R)$, which is true as ϵ can be made arbitrarily small and $V = \lceil \log_R 2 \rceil - 1$. We then release another OPT-only job X at $u + 1 + V$ with $h^*_X = R - \frac{\tau'_{u+1+V}}{R^{L-1}}$, $p_X = L$ and a tight deadline. As an OPT-only job, X will be too hot to be scheduled by $\mathcal{A}$ meanwhile OPT will schedule it. OPT will then idle for D time steps until it cools down to a temperature of ϵ, and then schedule J. J will still be pending as $u \leq D$, therefore $u+1+V+L+D \leq V+L+2D+1 < d_J$. For this case $\mathcal{A}$ will only have scheduled J and so have a profit of 1 while OPT will schedule J, K and X receiving a profit of $1 + V + L$.

Case 2: : Otherwise $\mathcal{A}$ starts J at some $u > D$. In this case OPT will start J in the first time slot and so be ϵ or cooler by time u. We then proceed as before releasing jobs K and X. Again we will get a competitive ratio of $1 + V + L$.

□

4.4.2 Upper Bound: Longest First

In this proportionally weighted model the value of a job depends on its length, with a longer job giving more value than a short job. A natural algorithm to consider for creating a schedule with this objective is therefore the simple Longest First algorithm. Longest First will run a job whenever there is a job admissible, meaning it belongs to the class of non-idling algorithms. If there are several admissible jobs it runs the one with the largest processing

time. If there are several jobs with this processing time, then the job with the smallest heat contribution is chosen. Longest First does not preempt any jobs. We now provide an upper bound on the competitive ratio of the algorithm Longest First which we will denote as $\mathcal{A}$ throughout the proof.

We analyse $\mathcal{A}$ by splitting the schedule produced by $\mathcal{A}$ into regions $[u, v)$ as in Theorem 3.5. To recall, each region is defined as starting at a time u when a job J starts in $\mathcal{A}$ and ending at the time v that $\mathcal{A}$ starts another job J'. This means that for two consecutive regions $[u, v)$ and $[u', v')$ it must be that $v = u'$. Each job started by $\mathcal{A}$ and OPT will give credits to some region $[u, v)$. We use $\Phi^{\mathcal{A}}_{[u,v)}$ and $\Phi^{OPT}_{[u,v)}$ to denote the credits associated to a region $[u, v)$ of $\mathcal{A}$ and OPT respectively. We then show that for every region $[u, v)$ it must be that when $V = 0$ we have $\Phi^{OPT}_{[u,v)} \leq (\frac{3}{2} + L)\Phi^{\mathcal{A}}_{[u,v)}$ and when $V \geq 1$ we have $\Phi^{OPT}_{[u,v)} \leq (1 + V + L)\Phi^{\mathcal{A}}_{[u,v)}$. Summing over all regions shows that $\mathcal{A}$ is $(\frac{3}{2} + L)$-competitive when $V = 0$ and $(1 + V + L)$-competitive when $V \geq 1$.

In every region $\mathcal{A}$ will get credit for the job J that it schedules in the region so $\Phi^{\mathcal{A}}_{[u,v)} = p_J$. We also give $\Phi^{OPT}_{[u,v)}$ the same p_J credits for that region to account for the fact that J may be scheduled by OPT at some point in the future. This means that we now only consider jobs started by OPT in a region $[u, v)$ that have not already been scheduled by $\mathcal{A}$.

We use the following lemmas to bound the credit that can be generated by OPT-only jobs in any given region.

Lemma 4.7. *If an OPT-only job J is started by OPT at a time t and J is not the last OPT-only job in that region, $p_J \leq V$.*

Proof. As we know that $\tau_t \leq 1$, it must be that $h^*_J > R - 1/R^{p_J - 1}$ for J

to be OPT-only. It must also be that $\tau'_t \geq 0$, therefore $\tau'_{t+p_J} > 1 - 1/R^{p_J}$. By definition of the region and OPT-only jobs it must be that $\mathcal{A}$ stays idle during the region $[t, t + p_J)$ (otherwise a new region will be started and so J was the last job that OPT could start in that region), which will give $\tau_{t+p_J} \leq 1/R^{p_J}$. For there to be another OPT-only job in the region we must have $\tau_{t+p_J} > \tau'_{t+p_J}$. This implies $1/R^{p_J} > 1 - 1/R^{p_J}$, which is equivalent to $p_J < \log_R 2$. As p_J must be an integer it means that for this to be possible $p_J \leq \lceil \log_R 2 \rceil - 1 = V$. □

Lemma 4.8. *The maximum credits that OPT can generate from OPT-only jobs in a region $[u, v)$ is $V + L$.*

Proof. First we note that a job X that starts at some time t with $h_X = x + \eta$ where $x = \frac{(R-1)(R^{p_X} - \tau_t)}{R^{p_X} - 1}$ is OPT-only whenever $\eta > 0$. Using this definition we will refer to x as being the minimum total heat contribution a job X needs to be OPT-only.

Consider the heat contributions of two OPT-only jobs J and K such that $p_J > p_K$, that start at some time t. Let

$$j = \frac{(R-1)(R^{p_J} - \tau_t)}{R^{p_J} - 1}; \quad k = \frac{(R-1)(R^{p_K} - \tau_t)}{R^{p_K} - 1}.$$

We have that $j < k$ iff

$$(R-1)(R^{p_J} - 1)(R^{p_K} - 1)(\tau_t - 1)(R^{p_J} - R^{p_K}) < 0$$

Because we have $R > 1$, $p_J > p_K \geq 1$ and $\tau_t \leq 1$ the above statement must be true. It follows that the minimum possible heat contribution per time

step of a longer OPT-only job will be no greater than that of a shorter job if both started at the same time t.

Now consider two consecutive OPT-only jobs J_1 and J_2 with no other jobs or idle time in between. Consider the heat contributions and temperatures if they were replaced with a single OPT-only job K, of length $p_{J_1} + p_{J_2}$, starting at the same time J_1 was started. We can assume all J_1, J_2, K have the minimum heat contributions for them to be OPT-only. The previous paragraph shows that K has a smaller per-step heat contribution than J_1. The same holds for J_2; the fact that J_2 starts later only makes the minimum heat contribution, in order for it to be OPT-only, higher. Therefore such a replacement results in a lower temperature of OPT by the time J_2/K finishes. We can apply a similar procedure so that all except the last OPT-only job are merged into one OPT-only job, with the temperature of OPT only being reduced. By Lemma 2.4 we still know that no other jobs or idle time space in between the OPT-only jobs can increase the total number of OPT-only jobs.

Lemma 4.7 shows that it is not possible for an OPT-only job of length greater than V to be followed by another OPT-only job. The original set of OPT-only jobs, before merging, will result in an even higher temperature and therefore also will also not allow another OPT-only job to appear after them. Therefore the total length of all except the last OPT-only jobs must be at most V.

Finally we can bound the total credit of all the OPT-only jobs. We have already bounded the total length of all but the last OPT-only jobs to be at most V, while it is clear that the last OPT-only job can have a length at

most L. Therefore the maximum total length of OPT-only jobs is $V + L$; it follows that this is the total number of credits that OPT can generate in a region. □

Now we use these lemmas to prove the competitiveness of Longest First.

Theorem 4.9. *When $V \geq 1$ Longest First is $(1 + V + L)$-competitive, otherwise when $V = 0$ Longest First is $(\frac{3}{2} + L)$-competitive.*

Proof. We will refer to the job scheduled by $\mathcal{A}$ in a given region as job J. Therefore for each region $[u, v)$ and job J scheduled by $\mathcal{A}$ in the region, we have $\Phi^{\mathcal{A}}_{[u,v)} = p_J$. By Lemma 4.8 we bound the credit for OPT generated by OPT-only jobs in every region to $V + L$. OPT also gets extra credit of p_J for the possibility that it may run J in the future. The only jobs that haven't been accounted for now are the jobs that are started by OPT while $\mathcal{A}$ is running J. We bound the processing time of these jobs in the following cases:

Case 1: At u OPT doesn't start a job. Note that this case applies if OPT is running a job it has already started, but the credit for this job will be counted towards the previous region. If $p_J = 1$ then no other jobs can be started while $\mathcal{A}$ is running J, so we have that $\Phi^{OPT}_{[u,v)} \leq 1+V+L$ and $\Phi^{\mathcal{A}}_{[u,v)} = 1$. This gives a competitive ratio of $1 + V + L$.

Otherwise $p_J > 1$. In this case OPT may start some jobs while J is running that $\mathcal{A}$ cannot start as $\mathcal{A}$ will not preempt J. The last of these jobs gives a credit of at most L while the total length of all the other preceding jobs is at most $p_J - 2$. This gives $\Phi^{OPT}_{[u,v)} \leq 2p_J - 2 +$

$V + 2L$. This gives a competitive ratio for this case of $\frac{2p_J - 2 + V + 2L}{p_J} = 2 + \frac{2L+V-2}{p_J} \leq 2 + \frac{2L+V-2}{2} = L + \frac{V}{2} + 1 \leq 1 + V + L$.

Case 2: At u OPT starts some job K with $p_K = p_J$. If $h_K \leq h_J$ then we know that either $K = J$ or K must have already been scheduled by $\mathcal{A}$ at some time before u, as otherwise $\mathcal{A}$ would have started K instead of J. In either case we do not need to give OPT any extra credit for K. Hence $\Phi^{OPT}_{[u,v)} \leq p_J + V + L$, and as $p_J \geq 1$ this case gives a competitive ratio of at most $1 + V + L$.

Otherwise $h_K > h_J$. We now consider two sub-cases.

Case 2a: $\tau_u \leq \tau'_u$. In this case it must also be that $\tau_{u+p_J} \leq \tau'_{u+p_J}$. Therefore OPT cannot schedule any OPT-only job that has not already been completed by $\mathcal{A}$ at some time earlier than u, after J in this region. This is because $\mathcal{A}$ will always start a job if one is admissible, and if a job was pending in $\mathcal{A}$ and admissible for OPT it would also be admissible for $\mathcal{A}$. This means that $\Phi^{OPT}_{[u,v)} \leq 2p_J$, as OPT can only get credit for running K and possibly J in the future. This gives a ratio of at most 2 for this sub-case.

Case 2b: $\tau_u > \tau'_u$. Similar to the proof of Lemma 4.7 it can be shown that if $\tau_u > \tau'_u$ then in the previous region OPT can have scheduled OPT-only jobs for at most V time steps. Therefore the previous region will be at least L credits below its maximum and as $p_K \leq L$ we can charge K to the previous region without the previous region's credits exceeding the bound, in a similar way to Theorem 3.5. This gives $\Phi^{OPT}_{[u,v)} \leq p_J + V + L$, giving a ratio of at

most $1 + V + L$.

Case 3: At u OPT starts some job K with $p_K > p_J$. If $\tau_u > \tau'_u$ then we can again charge K to the previous region. This gives $\Phi^{OPT}_{[u,v)} \leq p_J + V + L$ but no more as OPT starts K at u and $p_K > p_J$ so OPT cannot start any other jobs while $\mathcal{A}$ is running J.

Otherwise $\tau_u \leq \tau'_u$. In this case we know that if K was pending for $\mathcal{A}$ then it would have also been admissible for $\mathcal{A}$ and as $p_K > p_J$ $\mathcal{A}$ would have started K. This means that K must have already been scheduled by $\mathcal{A}$ and so we do not need to give OPT any extra credit for it.

In either instance it must be that $\Phi^{OPT}_{[u,v)} \leq p_J + V + L$, giving a competitive ratio of at most $1 + V + L$ for this case.

Case 4: At u OPT starts some job K with $p_K < p_J$. In this case it must be that $p_J \geq 2$. OPT may start some jobs before J has completed in $\mathcal{A}$ in this case. Similar to Case 1, the last of these jobs has length at most L and the total length of the other jobs preceding it is at most $p_J - 1$, so the maximum number of credits that can be generated by these jobs is $p_J - 1 + L$. This gives $\Phi^{OPT}_{[u,v)} \leq 2p_J - 1 + V + 2L$. This results in a competitive ratio of $\frac{2p_J - 1 + V + 2L}{p_J} = 2 + \frac{2L + V - 1}{p_J} \leq 2 + L + \frac{V-1}{2} = L + \frac{V+3}{2}$. When $V = 0$ this gives a competitive ratio of at most $\frac{3}{2} + L$, while for $V \geq 1$ the competitive ratio is at most $L + \frac{V+3}{2} \leq 1 + V + L$.

□

Note that all of the analysis in this section can be extended to the case where $w_J = p_J^\alpha$ for any $\alpha \geq 1$. The analysis of the number of jobs remains

the same, but the different credit for the jobs has to be taken into account i.e. Lemma 4.1 gives a new limit of $V^\alpha + L^\alpha$ on the amount of credit than can be generated by OPT-only jobs in a region. This gives a lower bound of $(1 + V^\alpha + L^\alpha)$ and an upper bound of $1 + V^\alpha + L^\alpha$ when $V \geq 1$ and $L^\alpha + 3/2$ when $V = 0$.

4.5 Summary

In this chapter we considered maximising the throughput of variable length jobs on a single machine. First we show that when jobs are unweighted and preemption is not allowed, any non-idling algorithm is nearly optimal. We also show that when both types of preemption are allowed, for any fixed cooling factor, these algorithms are still optimal within constant factors.

We then consider the case where jobs have weights that are proportional to their length. We show that for small cooling factors the algorithm Longest First is optimal, and for the remaining cooling factors the algorithm is almost optimal. This applies to the non-preemptive case, and with both types of preemption.

Chapter 5

Flow Time

In this chapter we consider the objective of minimising the flow time of a schedule. We will only consider the most basic single machine problem with unit length jobs i.e. $\forall J : p_J = 1$. We only consider the unweighted version of this problem. We will be considering the case of minimising total/average flow time of the schedule produced by the algorithm, apart from in Section 5.7 where we consider some results for the case of minimising the maximum flow time of a job in a schedule.

5.1 Motivation

Until this chapter we have only considered problems with the objective of maximising throughput, however this is only one way of assessing the quality of a scheduling algorithm. In some real life problems, maximising throughput might not be the best way to decide which algorithm is best. For example, consider the situation where customers have jobs that they would like

completing as soon as possible. In this case just completing the maximum number of jobs is not an appropriate measure of the quality of the schedule produced, while measuring the average flow time will give us a way to assess how content most customers are with how quickly their job was completed.

If we want to make sure that each customer is as happy with the schedule as every other customer, then it might make sense to measure the maximum flow time of a job in the schedule. It is necessary to make a distinction between the two different measures of flow time because it is possible that an algorithm that is good at minimising average flow time performs very badly for the objective of minimising maximum flow time.

5.2 Previous Results

Without temperature, it is well known that SRPT is optimal for minimising the total flow time in the case where jobs have arbitrary lengths and preemption is allowed. It is also well known that FIFO is optimal for minimising the maximum flow time in the non-temperature case. For the offline case without preemption with arbitrary job lengths it is known that unless $P = NP$, no polynomial time approximation algorithm for minimum flow time can have an approximation ratio of less than $O(\sqrt{n})$ [37]. They also gave an approximation algorithm that achieved the $O(\sqrt{n})$-approximation ratio.

The NP-hardness result from [19] is originally for the model of maximising throughput but also implies that it is NP-hard to minimise total flow time. In [6] they consider the makespan objective with multiple machines

where all jobs have the same release time (hence this is an offline problem). They showed that it is not possible to have a polynomial time approximation algorithm with a better approximation ratio than 4/3 for minimising the makespan of a schedule, unless $P = NP$. They also show a polynomial time algorithm with an approximation ratio of $7/3 - 1/3m$ for minimising the makespan. This algorithm and analysis therefore apply to the objective of minimising the maximum flow time in the restricted case where all release times are 0.

5.3 Preliminaries

We now outline some preliminaries that will be used throughout this chapter. We will describe jobs that are pending for an algorithm as being stored in a queue. For simplicity we will denote the queues of $\mathcal{A}$ and OPT at some time t as Q_t and Q'_t respectively, for the time instant when all jobs that are released at t have arrived, but before any jobs have been scheduled for that time step. We will denote the size of a queue Q at this time instant t as $|Q_t|$. Where there is no chance of confusion we will also denote the flow time of a schedule of $\mathcal{A}$ as $|\mathcal{A}|$, of a job J as $|J|$ and a set of jobs A as $|A|$. We will denote the maximum heat of any job as h_{max} throughout this chapter.

Job naming conventions. We now describe the naming conventions that we use for jobs in this chapter. In several of the proofs we release a large number of jobs with heat contributions of exactly $R - 1$, and other jobs with heat contributions of 0. Where it is appropriate, for simplicity, we

shall refer to these jobs as B-jobs and Z-jobs respectively. It should be noted that whenever a B-job is scheduled by an algorithm at some time t, the temperature of that algorithm at $t+1$ will be hotter than at t. In several proofs we also release what we call B-blocks. A B-block of size b will be a block of B-jobs released one per time step for b time steps. After each of the jobs in a B-block of size b has been scheduled the temperature of the algorithm that schedules it will be at least $1 - 1/R^b$. Using this construction the temperature of an algorithm can be made arbitrarily close to 1, by increasing the size of the B-block until the algorithm's temperature is as close to 1 as required, assuming the algorithm has to schedule all of the jobs in the block. In a similar way we also release Z-blocks of size z, that can reduce the temperature of the algorithm that schedules them completely to at most $1/R^z$.

Incremental flow times. It is well known that, in a discrete time model, the flow time of a schedule is equal to the sum of the number of pending jobs (i.e. the queue size) at each time step, over all time steps. This is because the flow time of each job will increase by 1 for each time step until they are completed. In many of the proofs we use this approach and bound the number of jobs in the queues in each time step. Sometimes we call this the *incremental flow time* of this time step. Moreover we can generalise this and refer to the incremental flow time of a time interval $[u, v)$ as the sum of incremental flow times (i.e. queue sizes) of the time steps $u, u+1, ...v-1$.

Repeating phases. Several of the proofs in this chapter use the fact that total flow time is the sum of an algorithm's queue size to show that an algorithm has an $\Omega(n)$-competitive ratio. We generally proceed by describing a phase that is repeated many times. We formalise our approach in the following lemma.

Lemma 5.1. *Suppose the schedules of OPT and $\mathcal{A}$ can be partitioned into phases such that (i) all phases are identical, i.e. they contain the same constant number of time steps $t = \Theta(1)$, the jobs released and OPT and $\mathcal{A}$'s schedules are repeated over all phases; (ii) if started from an empty queue, the incremental flow time of a phase for OPT and $\mathcal{A}$ are both $\Theta(1)$; (iii) at the end of the phase the queue size of $\mathcal{A}$ will be one larger than before the phase starts but OPT's will remain the same. Then after c phases the competitive ratio of $\mathcal{A}$ is at least $\Omega(c)$.*

Proof. Suppose there are c phases each containing a constant number of time steps $t = \Theta(1)$. Just before the i-th phase $\mathcal{A}$'s queue contains $i-1$ jobs, so the incremental flow time for the i-th phase for $\mathcal{A}$ will be $\Theta(1) + (i-1)t = \Theta(i)$ (as each additional pending job contributes 1 to the incremental flow time in each of the t steps). Meanwhile the incremental flow time for OPT is still $\Theta(1)$. After the i-th phase $\mathcal{A}$ will now have i jobs in its queue. The total flow time for $\mathcal{A}$ after c phases is therefore $\Theta(c^2)$ and for OPT it is $\Theta(c)$, therefore the competitive ratio of $\mathcal{A}$ is then $\Omega(c)$. □

5.4 The Offline Case

In this section we give three results about the offline version of the problem: first we generalise the proof in [19] and prove that the problem remains NP-hard even if all jobs have heat contribution arbitrarily close to $R-1$ (below which the problem becomes trivial). For simplicity we only consider the case when $R = 2$, although the proof can be extended to all R values. Next we show that the problem is not approximable within a factor of $O(\sqrt{n})$, unless $P = NP$ (this however uses hot jobs). Finally we show an algorithm that achieves a 2.618-approximation ratio for the special case where all jobs have the same release time.

5.4.1 NP-hardness

The result given in this section is for the problem formally denoted as $1|r_i, h_i, p_i = 1|\sum F_i$.

Theorem 5.2. *When $R = 2$ and h_{max} is at most $R - 1 + \delta$ for any $\delta > 0$, the offline problem is NP-hard.*

Proof. As in [19] this reduction is from NUMERICAL-3D-MATCHING (N3DM). In this problem, there are 3 sets A_1, A_2 and A_3 of k non-negative integers each and a positive integer β. A 3-dimensional numerical matching is a set of k triples $(a_1, a_2, a_3) \in A_1 \times A_2 \times A_3$ such that each number is matched exactly once and all triples satisfy $a_1+a_2+a_3 = \beta$. NUMERICAL-3D-MATCHING is known to be NP-complete even when the values of all numbers are bounded by a polynomial in k.

We construct a job for each integer in A_1, A_2, A_3 as follows. First we define a function $f(x) = \frac{\delta x}{(y-1)\beta}$ for some $y > 0$ to be described later and small $\delta > 0$. We also fix some α_1, α_2 and α_3 to be chosen later. For every integer $a_1 \in A_1$ there is an A_1-job with a heat contribution such that $h_{A_1} = \alpha_1 + 8f(a_1)$. For every integer $a_2 \in A_2$ there is an A_2-job with a heat contribution such that $h_{A_2} = \alpha_2 + 4f(a_2)$. For every integer $a_3 \in A_3$ there is an A_3-job with a heat contribution such that $h_{A_3} = \alpha_3 + 2f(a_3)$. We will refer to these jobs collectively as A-jobs. These heat contributions will be described in more detail later but we first claim they must satisfy the following:

$$\frac{1+\alpha_1}{8} + \frac{\alpha_2}{4} + \frac{\alpha_3}{2} = 1 - \frac{y\delta}{y-1} \tag{5.1}$$

$$\alpha_1 > 1 - \frac{79\delta}{105}; \quad \alpha_2 > 1 - \frac{1829\delta}{1575}; \quad \alpha_3 > 1 - \frac{9\delta}{7}$$

$$1 \geq h_{A_1} > h_{A_2} > h_{A_3}$$

It can be easily shown that it is possible to find values that satisfy all of these conditions: first, if we choose $\alpha_1, \alpha_2, \alpha_3$ that are as close as possible to the minimum values stated above then we have $1 > \alpha_1 > \alpha_2 > \alpha_3$ and moreover $\alpha_1 + 2\alpha_2 + 4\alpha_3 < 7 - 8\delta$. Therefore the α_i's can be chosen such that $\alpha_1 + 2\alpha_2 + 4\alpha_3$ is arbitrarily close to $7 - 8\delta$. Equation (5.1) is equivalent to $y = \frac{7-(\alpha_1+2\alpha_2+4\alpha_3)}{7-8\delta-(\alpha_1+2\alpha_2+4\alpha_3)}$, therefore we can make y as large as necessary. It follows that the $f(.)$ terms are arbitrarily small and so $1 > h_{A_1} > h_{A_2} > h_{A_3}$.

The job scheduling instance consists of a B-block of size x released at time 0, where x is a constant large enough to bring the temperature to close enough to 1 if all jobs in the B-block are scheduled consecutively, and the

$3k$ A-jobs described above are released at time x. In addition we have k "gadget" jobs that have heat contributions of $1 + \delta$, also released at time x. Note that all non-gadget jobs have heat contributions of at most 1 and so are always admissible at any time step.

The idea of the proof is as in [19] but will be described here for completeness. We show there is a solution to the NUMERICAL-3D-MATCHING instance if and only if there is a schedule that completes all jobs by time $x + 4k$, i.e. with no idle time. The gadget jobs are hot enough so that in order to be scheduled without idle slots they need to be scheduled every 4th time slot, splitting the schedule into k blocks. The other jobs have heat contributions that depend on two parts. The first part is the constant part (α_1, α_2, α_3). This part is large enough so that in each block there must be exactly a single A_1, A_2 and A_3 job, and also ensure that they must be scheduled in that order. The second part is variable and depends on the instance of the matching problem. This defines the partition of triplets into the form $(a_1, a_2, a_3) \in A_1 \times A_2 \times A_3$. The gadget jobs are then hot enough to force every triple to satisfy each $a_1 + a_2 + a_3 \leq \beta$. We now formalise the argument.

($\Rightarrow$) Suppose there is a solution to the instance of NUMERICAL-3D-MATCHING. We construct a schedule that contains no idle time. We start by scheduling all of the jobs in the B-block, which will bring the temperature to close to 1. We then schedule the gadget jobs in every 4th time slot, which splits the schedule into k blocks. We associate the ith triple (a_1, a_2, a_3) to an ith block containing the corresponding A_1, A_2, A_3 jobs scheduled in that order.

We can show that the temperature of the schedule never exceeds 1. The

non-gadget jobs must all be cooler than 1 so by definition after they have completed the temperature must be at most 1, therefore we now show that after each gadget job has executed the temperature must be no greater than 1 by induction. Note that after scheduling the B-block the temperature must be close to, but below, 1.

We now work out the temperature of the schedule just before the ith gadget job is started. By definition of $f(x)$ and the fact that $a_1+a_2+a_3 = \beta$, we have that:

$$\frac{8f(a_1)}{8} + \frac{4f(a_2)}{4} + \frac{2f(a_3)}{2} = \frac{\delta}{y-1}$$

Combined with the definition of α_1, α_2 and α_3, the temperature just before the gadget job is scheduled is at most

$$\frac{1+h_{A_1}}{8} + \frac{h_{A_2}}{4} + \frac{h_{A_3}}{2} = 1 - \frac{y\delta}{y-1} + \frac{\delta}{y-1} = 1-\delta$$

Therefore after the gadget job the temperature is at most $(1-\delta+1+\delta)/2 = 1$.

($\Leftarrow$) Suppose there is a schedule with no idle time. It must again be that all of the jobs in the B-block are scheduled immediately in the interval $[0, x)$, bringing the temperature to close to 1. We now show that a gadget job must be scheduled every 4th time slot. First we note that every job heat must be hotter than α_3, and because the temperature is arbitrarily close to 1 at time x, it follows that the temperature must be hotter than α_3 at any time after x. We now show by contradiction that a gadget job cannot be scheduled two time slots after the last gadget job was scheduled.

$$\frac{\alpha_3+1+\delta}{8} + \frac{h_{A_3}}{4} + \frac{h_{A_3}}{2} > \frac{\alpha_3+1+\delta}{8} + \frac{\alpha_3}{4} + \frac{\alpha_3}{2} > 1-\delta$$

This must be true as long as $\alpha_3 > 1 - 9\delta/7$.

We can use this restriction to work out the minimum temperature straight after a gadget job has been scheduled.

$$\frac{\tau + \alpha_3}{16} + \frac{\alpha_3}{8} + \frac{\alpha_3}{4} + \frac{1+\delta}{2} = \tau$$

Therefore $\tau > 1 - \delta/15$.

We can use this to show that an A_1 job must be in the first time slot of a block. We show this by contradiction, assume it is not scheduled in the first time slot then:

$$\frac{\tau + \alpha_3}{8} + \frac{h_{A_1}}{4} + \frac{\alpha_3}{2} > \frac{\tau + \alpha_3}{8} + \frac{\alpha_1}{4} + \frac{\alpha_3}{2} \geq 1 - \delta$$

This must be true as long as $\alpha_1 > 1 - 79\delta/105$.

We can then use this information to work out what this will make the new minimum temperature after a gadget job:

$$\frac{\tau + \alpha_1}{16} + \frac{\alpha_3}{8} + \frac{\alpha_3}{4} + \frac{1+\delta}{2} = \tau$$

Therefore $\tau > 1 - 7\delta/225$.

We now use this to show that an A_2 job must be scheduled in the second time slot.

$$\frac{\tau + h_{A_1}}{8} + \frac{\alpha_3}{4} + \frac{h_{A_2}}{2} > \frac{\tau + \alpha_1}{8} + \frac{\alpha_3}{4} + \frac{\alpha_2}{2} \geq 1 - \delta$$

This must be true as long as $\alpha_2 > 1 - 1829\delta/1575$.

We now show that each triple must satisfy $a_1 + a_2 + a_3 = \beta$. If this isn't the case then there must exist a triple (a_1, a_2, a_3) such that $a_1 + a_2 + a_3 > \beta$ which implies

$$\frac{8f(a_1)}{8} + \frac{4f(a_2)}{4} + \frac{2f(a_3)}{2} > \frac{\delta}{y-1}$$

This means that:

$$\frac{1+h_{A_1}}{8} + \frac{h_{A_2}}{4} + \frac{h_{A_3}}{2} = 1 - \frac{y\delta}{y-1} + \frac{8f(a_1)}{8} + \frac{4f(a_2)}{4} + \frac{2f(a_3)}{2} > 1 - \delta$$

and the gadget job that follows will not be admissible, contradicting the feasibility of the schedule. □

5.4.2 Inapproximability

In this section we extend the inapproximability result from [37] to show that in our model for any $R > 1$ no polynomial time approximation algorithm can have a worst case performance guarantee of $O(n^{1/2-\epsilon})$ for any $\epsilon > 0$. The proof is again a reduction from the N3DM problem. For consistency with [37] we will redefine the notation used to describe the N3DM problem. The problem consists of positive integers a_i, b_i and c_i, $1 \leq i \leq k$, with $\sum_{i=1}^{k}(a_i + b_i + c_i) = kD$. The problem is finding if there exist permutations π, ψ such that $a_i + b_{\pi(i)} + c_{\psi(i)} = D$.

For every N3DM instance and for any given $\epsilon > 0$, we construct a corresponding scheduling instance. We first define some numbers:

$$r = \lceil \mu k^{2(1/\epsilon - 1)} D^{1/\epsilon} \rceil; \quad g = rk^2$$

where μ is a large constant.

For every number a_i in the N3DM instance, we introduce a corresponding job with a heat contribution of $R - 1/R^{(2r+a_i)rg-1}$, for every b_i a job with a heat contribution of $R - 1/R^{(4r+b_i)rg-1}$, and for every c_i a job with a heat contribution of $R-1/R^{(8r+c_i)rg-1}$. These $3k$ jobs are called *big* jobs and they are all released at time rg.

We will also have a number of *tiny* jobs. Tiny jobs occur in groups denoted by $G(t;l)$ where t and l are positive integers. A group $G(t;l)$ consists of l tiny jobs, released at times $t+i$ for $i = 0, ..., l-1$, all with heat contributions of $R-1$. Note that it is always possible to process all jobs in $G(t;l)$ during the time interval $[t, t+l]$ with a total flow time of l (and a final temperature below 1). We introduce the following groups of tiny jobs:

(T_1) For $0 \leq i \leq k$, we introduce the group $G(i(14r + D + 1)rg; rg)$.

(T_2) For $1 \leq i \leq g$, we introduce the group $G((k(14r + D + 1) + ir + 1)rg - 2g; 2g)$.

This completes the construction of the scheduling instance. The idea is that the $k + 1$ groups of type (T_1) occur regularly starting at time 0. Each takes rg time steps and leaves k 'holes' each of length $(14r + D)rg$ between them. Similarly, the groups of type (T_2) occur regularly after time $k(14r + D + 1)rg + rg$. They leave holes of size $(r^2 - 2)g$.

Each group has at least g jobs in it. The number g must be chosen (by choosing a large enough μ) such that after scheduling consecutively g jobs with heat contribution $R-1$, the temperature of the system is brought 'very close' to 1. If this temperature were exactly 1 then a job with a heat

contribution of $R - 1/R^x$, for some positive integer x, would require exactly x consecutive idle time steps before the job becomes admissible, and then after its execution the temperature is again 1. In fact, when we say 'very close', all we need is that this temperature is strictly greater than $1/R$; a job with heat contribution $R - 1/R^x$ would still require x idle time steps. It can be easily shown that after scheduling g consecutive jobs of heat contribution $R - 1$, the temperature is at least $1 - 1/R^g$, so for any constant R there is a corresponding constant g that makes this temperature greater than $1/R$. Moreover, right after scheduling a big job the temperature is clearly also greater than $1/R$.

We will denote the total flow time obtained by the optimal algorithm as F^* throughout this proof.

Lemma 5.3. *If the N3DM instance has a solution then for the constructed scheduling instance $F^* \leq 52rg^2$ holds.*

Proof. Consider the following feasible schedule. All tiny jobs are processed immediately at their release times. Hence, their total flow time equals $(k + 1)rg + 2g^2$. For every triple $(a_i + b_{\pi(i)} + c_{\psi(i)})$ with sum D in the solution we put the corresponding three jobs together into one of the holes of length $(14r + D)rg$ that are left free by the groups of type (T_1). The job corresponding to a_i is scheduled first, followed by the job corresponding to $b_{\pi(i)}$, and finally the job corresponding to $c_{\psi(i)}$. We now show that such a schedule is always feasible. Consider the time t_i after the $(i - 1)$-th type (T_1) group has completed. We have $\tau_{t_i} \leq 1$ and we know that a job of temperature $R - 1/R^x$ will be admissible after x consecutive idle time steps. The heat contribution of the

a_i job is $R - 1/R^{(2r+a_i)rg-1}$, and so the job can be completed in $(2r + a_i)rg$ time steps ($(2r+a_i)rg-1$ idle time steps followed by one time step of actual execution). After executing this job the temperature is at most 1 again. The same holds for the other two jobs. As there is a matching we know that $a_i + b_{\pi(i)} + c_{\psi(i)} = D$. This means that all three jobs can be completed in $(14r+D)rg$ time steps, precisely the size of the hole between the (T_1) groups $i-1$ and i.

It is easy to see that this schedule yields a total flow time of

$$((k+1)rg + 2g^2)+$$
$$\left(3\sum_{i=1}^{k}(a_i+2r) + 2\sum_{i=1}^{k}(b_i+4r) + \sum_{i=1}^{k}(c_i+8r) + \frac{3}{2}(14r+D+1)k(k-1)\right)rg$$

Since $\sum_{i=1}^{k}(a_i+b_i+c_i) = kD$ and since $D \leq r$, this gives an upper bound of

$$((k+1)rg+2g^2)+(25rk+24rk(k-1))rg < (rg^2+2g^2)+(49rk^2)rg < 52rg^2.$$

□

Next we prove that if the N3DM instance has no solution that the flow time of the corresponding job instance must be large. Before that we need several lemmas on the structure of the optimal schedule regarding how the tiny jobs are scheduled. We first state this observation (see e.g. [19]): given two jobs, scheduling the hotter job first followed by the cooler job will result in a lower final temperature than scheduling them the other way round, as long as the hotter job is admissible at the earlier time step. In particular

this means we can assume, without loss of generality, that an idle slot will not precede a job if the job is admissible at the idle slot.

Define a *tiny interval* to be the time interval during which a group of tiny jobs is released.

Lemma 5.4. *In any optimum schedule, and in any subinterval of a (T_1) tiny interval where no big jobs are scheduled, we can without loss of generality assume that the schedule is in the form of a number (possibly zero) of consecutive tiny jobs followed by a number (possibly zero) of consecutive idle steps.*

Proof. If there are some idle slots in a tiny interval, the only reason for delaying the processing of those tiny jobs is to get a lower temperature by the end of this tiny interval so that the big job that follows can execute earlier. If there is an idle slot followed by a scheduled tiny job, we can change this to a tiny job followed by an idle slot, and this (i) makes the temperature lower, and (ii) reduces the flow time of this group of tiny jobs. □

Lemma 5.5. *If a schedule S^* has flow time less than $r^2g^2/4$, then in each (T_1) tiny interval, the temperature must be strictly greater than $1/R$ at some point.*

Proof. Let the tiny interval be $[t, t + rg - 1)$. If a big job is scheduled somewhere inside then obviously the temperature is strictly greater than $1/R$ after this big job. Suppose no big job is scheduled in this interval, and suppose y of those tiny jobs are scheduled after time $t+rg-1$. By Lemma 5.4, we can assume the other $rg-y$ tiny jobs are scheduled consecutively in $[t, t+rg-y)$. Even if all these y delayed jobs are scheduled in $[t + rg, t + rg + y)$, the flow

time of this tiny group is at least $(rg - y) + y(y + 1) = rg + y^2$. Thus, if $y > rg/2$, the flow time of this group alone is already at least $(rg)^2/4$, contradicting the condition of the lemma. Hence at least $rg/2 > g$ tiny jobs are scheduled consecutively starting at t, bringing the temperature to strictly larger than $1/R$. □

Lemma 5.6. *Consider a schedule S^* with flow time less than $r^2g^2/4$ and consider a (T_1) group. Let y_1 be the number of idle slots after the point of highest temperature in the tiny interval. Then $y_1 < g$.*

Proof. Let t be the ending time of the tiny interval. By Lemmas 5.4 and 5.5, the hottest point in the tiny interval must have temperature greater than $1/R$. After this point all slots must be idle (since either a big job or a tiny job can only raise the temperature). So there are (at least) y_1 tiny jobs delayed. Clearly, the only reason to delay scheduling these y_1 jobs is to schedule a big job in front of them. As any big job has heat at least $R - 1/R^{2r^2g-1}$, it takes at least $2r^2g - 1$ idle steps from a temperature of larger than $1/R$ before a big job is admissible. Hence a big job can finish earliest at time $t - y_1 + 2r^2g$. Therefore, similar to the proof of Lemma 5.5, the total flow time of this tiny group is at least $(rg - y_1) + y_1(2r^2g)$. If $y_1 \geq g$ then the flow time of this group is already at least $r^2g^2/4$, contradicting the condition of the lemma. Hence $y_1 < g$. □

Lemma 5.7. *Consider a schedule S^* with flow time less than $r^2g^2/4$ and consider a (T_1) tiny group where a big job is scheduled. Let y_2 be the number of tiny/idle slots in this tiny interval before the big job. Then $y_2 < rg/2 + g$.*

Proof. By Lemma 5.4, the tiny interval before the big job must consists of consecutive tiny jobs followed by idle slots. There must be less than g consecutive tiny jobs; any more would result in a temperature of greater than $1/R$ and the big job cannot be scheduled in this tiny interval. From the proof of Lemma 5.5, there cannot be more than $rg/2$ idle slots in a tiny group. Hence $y_2 < rg/2 + g$. □

Lemma 5.8. *In a schedule S^* with flow time less than $r^2g^2/4$, a big job cannot be scheduled after the end of the last (T_1) group.*

Proof. First we show that in S^* none of the big jobs are processed during the time interval that starts with the release of the last group of type (T_1) and ends with the release of the last group of type (T_2). By Lemma 5.6, the temperature is larger than $1/R$ less than g steps before the end of this last (T_1) group. Hence a big job can only be run at least $2r^2g - g$ steps after the end of the last (T_1) group. Since the first (T_2) group arrives after less than r^2g time steps, and the size of the (T_2) group is $2g$, this big job must be executed at least $(2r^2g - g) - r^2g - 2g = r^2g - 3g$ steps after the end of the (T_2) interval. Some of these (T_2) jobs may be executed during this (T_2) interval or after it. From Lemma 5.4 we can assume w.l.o.g. that any (T_2) jobs not delayed are executed as early as possible in this interval, and those delayed are scheduled after the big job. There are two possibilities: (1) If less than g tiny jobs are scheduled in the tiny interval, i.e. at least g are delayed to after the big job, then the total flow time of these tiny jobs is at least $(r^2g - 3g)g \geq r^2g^2/4$, contradicting the condition of the lemma. (2) If at least g tiny jobs are scheduled in the tiny interval, then the temperature

becomes larger than $1/R$, so the big job will need another $2r^2g$ idle steps, i.e. it can only be executed completely after the next (T_2) group. We can then apply the same argument to the next group.

Therefore, the only remaining possibility is to execute the big job after the last (T_2) group, which is completed at time $(k(14r+D+1)+gr+1)rg$. Hence, the big job has a completion time of at least $r^2g^2 + rg$, and since its release time is rg, its flow time will therefore be at least $r^2g^2+(r-1)g > (r^2g^2)/4$ for suitably large r and g, again contradicting the condition of the lemma. □

Lemma 5.9. *If the N3DM instance does not have a solution, then for the constructed scheduling instance $F^* \geq r^2g^2/4$ holds.*

Proof. Consider an optimum schedule S^* and suppose that its total flow time is strictly less than $r^2g^2/4$. From Lemma 5.8 we know that the big jobs can only be scheduled before the end of the last group of (T_1) jobs. If all (T_1) tiny jobs are scheduled as soon as they are released, then the big jobs can only be scheduled in the holes between (T_1) intervals. The temperature at the beginning of the hole is larger than $1/R$, and each hole is of size $(14r + D)rg$. However, not all tiny jobs may be scheduled in their intervals. From Lemma 5.6 we know that a temperature of larger than $1/R$ will appear at most g steps before the start of the hole. If a big job is scheduled in a tiny interval earlier than the final g steps of the interval, we count it towards the hole before it; Lemma 5.7 tells us that it must be scheduled at most $rg/2 + g$ steps into a tiny interval. Thus, the 'effective' size of a hole (counting from the point before the hole when the temperature is larger than $1/R$, to the last point when a big job can be scheduled) is at

most $(14r + D)rg + rg/2 + 2g \leq 15r^2g$.

Suppose two jobs corresponding to the numbers c_i and c_j are scheduled into one of these holes. Before each big job can start there must be at least $8r^2g - 1$ idle time steps (from a point when the temperature is larger than $1/R$, either because of the tiny group or another big job). Adding the two time steps to actually execute the jobs, there must be at least $16r^2g$ time steps in a hole to schedule two jobs corresponding to the numbers c_i and c_j, therefore this is impossible. As there are k holes and k c-jobs, having no c-job in one hole would imply two or more c-jobs in another hole, so it follows that each hole must contain exactly one job corresponding to some c_i.

By similar arguments we can show that every hole must contain exactly one job corresponding to some a_i, b_j and c_h respectively. Again by relating the heat contribution of the jobs to the number of idle steps required, this implies that for the corresponding three numbers $(2r + a_i)rg + (4r + b_j)rg + (8r + c_h)rg \leq (14r + D)rg + rg/2 + 2g$. Moreover, as a_i, b_i and c_i are integers, the quantity on the left is a multiple of rg and so the extra $rg/2+2g$ is not useful and the inequality is equivalent to $(2r+a_i)+(4r+b_j)+(8r+c_h) \leq 14r+D$, i.e. $a_i + b_j + c_h \leq D$. This in turn means that $a_i + b_j + c_h = D$ since the sum of these $3k$ numbers is kD. This is a contradiction as this implies the N3DM instance has a solution, and therefore our claim is proved. □

We have introduced $3k$ big jobs, $k + 1$ groups of tiny jobs of size rg and g groups of tiny jobs of size $2g$, so the total number of jobs n is equal to $3k + (k + 1)rg + 2g^2 = 3k + (k + 1)r^2k^2 + 2r^2k^4$.

Lemma 5.10. *For any constant γ, there exists sufficiently large k and D*

such that $\gamma n^{1/2-\epsilon} < r/208$.

Proof. We have $n = 3k + r^2k^2(k+1) + 2r^2k^4 < 4r^2k^4$. Hence

$$n^{1/2-\epsilon} < 4^{1/2-\epsilon} r^{1-2\epsilon} k^{2-4\epsilon} < 2r(\mu k^{2(1/\epsilon-1)} D^{1/\epsilon})^{-2\epsilon} k^{2-4\epsilon} = (2\mu^{-2\epsilon} k^{-2} D^{-2})r.$$

Hence, no matter how large γ is, there are large enough k and D so that $\gamma n^{1/2-\epsilon}$ is smaller than $r/208$. □

Theorem 5.11. *For any $\epsilon > 0$, there is no polynomial time approximation algorithm for minimising total flow time with worst-case approximation ratio $O(n^{1/2-\epsilon})$, unless $P = NP$.*

Proof. Suppose an approximation algorithm $\mathcal{A}$ with approximation ratio $O(n^{1/2-\epsilon})$ exists. The approximation ratio of $\mathcal{A}$ is smaller than $r/208$ by Lemma 5.10. Take an instance of N3DM, and perform the above construction. Both r and g are polynomial in k and D, thus the number of jobs is also polynomial. Moreover the heat contributions of a big job $R - 1/R^{(2r+a_i)rg-1}$ (more precisely the $-1/R^{(2r+a_i)rg-1}$ part) can be encoded in $O((r + a_i)rg)$ bits, again polynomial in k and D. Thus, as long as the instance of N3DM is encoded in unary (N3DM remains NP-complete when encoded in unary [37]), the size of the constructed instance is polynomial in the size of the N3DM instance and this can be done in polynomial time. Apply algorithm $\mathcal{A}$ to the resulting instance. From Lemma 5.3, if the N3DM instance has a solution then the flow time of the scheduling instance returned by $\mathcal{A}$ is smaller than $52rg^2 \cdot r/208 = r^2g^2/4$. On the other hand, if the N3DM instance has no solution then from Lemma 5.9 the flow time returned by $\mathcal{A}$ is at least $r^2g^2/4$.

Hence with the approximation algorithm $\mathcal{A}$ we can distinguish the two cases and solve the N3DM problem in polynomial time, hence $P = NP$. □

Note that this result also implies that no online polynomial-time algorithm can have a competitive ratio better than $O(\sqrt{n})$, unless $P = NP$.

5.4.3 Approximation: Identical Release Times

We consider the single machine case with n jobs, all having the same release time, w.l.o.g. we can assume it is 0. More formally this problem is denoted as $1|r_i = 0, h_i, p_i = 1| \sum F_i$.

Note that if more than 1 job with a heat contribution of R is released then no algorithm will ever be able to schedule more than the first such job. Therefore we will restrict our attention to the set of inputs with at most 1 job with a heat contribution of R, as these are the only inputs that allow a feasible schedule to be constructed.

For each job J_i with a heat contribution $h_{J_i} > R - 1$, we define a value k_i that is the largest k such that $h_{J_i} > \frac{R^k - 1}{R^{k-1}}$.

We now show an upper bound for this case. The upper bound proof makes use of two propositions from [6] that will be restated here for completeness.

Proposition 5.12. *[6] Any schedule in which every job J_i is executed after at least k_i idle slots is feasible.*

Proposition 5.13. *[6]*

When $R \geq 2$, in an optimal schedule, between the execution on the same processor of jobs J_j and J_i (where J_j is before J_i), of heat contributions

$h_{J_j}, h_{J_i} > R-1$ *there are at least* $k_i - 1$ *slots, which are either idle or execute jobs of heat contribution at most* $R-1$.

First the algorithm orders all of the jobs in non-decreasing order by heat contribution i.e. $h_{J_1} \leq h_{J_2} \leq ... \leq h_{J_n}$. Next we split the jobs into 2 sets depending on their heat contribution. We refer to the number of jobs with a heat contribution of not hotter than $R-1$ as c and use this to split the jobs. We define set C as all jobs with heat contributions of not hotter than $R-1$, $C = \{J_1, J_2, ..., J_c\}$ and the set H as all the other jobs, $H = \{J_{c+1}, J_{c+2}, ..., J_n\}$. For simplicity we will refer to a job J_i in set C (where $i \leq c$) as C_i and a job J_{c+i} in set H (where $i \leq n-c$) as H_i.

The algorithm first assigns the hottest job, H_{n-c} if $H \neq \emptyset$, to the first time slot. For time slots 2 to $c+1$ the algorithm then assigns all of the C jobs in descending order. These jobs will always be admissible as they all have heat contributions of not greater than $R-1$. All of the remaining jobs $J_i \in H - \{H_{n-c}\}$ are then scheduled in the coolest first order, where each job of heat contribution h_{J_i} is preceded by k_i idle time slots. Proposition 5.12 ensures that the schedule is feasible.

This is only a small modification of Coolest First, but it is necessary to obtain the 2-approximation ratio. To illustrate this necessity consider the following example, we have two jobs J_1 and J_2 such that $h_{J_1} = R-1$ and $h_{J_2} = R - 1/R^k$ for some large k. Coolest First would schedule J_1 followed by J_2 and produce a schedule with the total flow time of $1 + \Omega(k)$ while the modification given here will schedule them in the opposite order, producing a schedule with the flow time of 2.

Theorem 5.14. *When $R \geq 2$, the above algorithm $\mathcal{A}$ achieves a* 2.618*-approximation ratio for minimising total flow time.*

Proof. If $c \geq n-1$ then $\mathcal{A}$ will schedule 1 job every time step without idling and so must be optimal, therefore we need to only consider the case where $c < n-1$. It is clear that $|\mathcal{A}| = |C| + |H| = |C| + |H - \{H_{n-c}\}| + 1$. We can use the following to calculate the flow time of each of the jobs C_i for $1 \leq i \leq c$ as $|C_i| = i+1$. This can be used to get the flow time for the set C.

$$|C| = \sum_{i=1}^{c} |C_i| = \frac{c^2 + 3c}{2}$$

Next we calculate the flow time of each job H_i for $1 \leq i < n-c$: $|H_1| = 1 + c + k_{c+1} + 1$ and $|H_i| = |H_{i-1}| + k_{c+i} + 1$, solving this recursion gives $|H_i| = c + i + 1 + \sum_{j=1}^{i} k_{c+j}$. We can then use this to get the flow time for the set $H - \{H_{n-c}\}$.

$$|H - \{H_{n-c}\}| = \sum_{i=1}^{n-c-1} |H_i| = \frac{n^2 + n - c^2 - 3c - 2}{2} + \sum_{i=1}^{n-c-1} k_{c+i}((n-c-1)-(i-1))$$

These can be combined to give the total flow time of $\mathcal{A}$:

$$|\mathcal{A}| = 1 + \frac{c^2 + 3c}{2} + \frac{n^2 + n - c^2 - 3c - 2}{2} + \sum_{i=1}^{n-c-1} k_{c+i}((n-c-1)-(i-1)) \quad (5.2)$$

We now analyse the flow time of the optimal schedule by analysing the flow time of a virtual schedule OPT' that must have a flow time of no more than that of OPT. OPT' will schedule the hottest job H_{n-c} at the first time step. OPT' will then schedule each job in H according to the Shortest

Processing Time First rule, but using k_i as the processing time for each job J_i (instead of $k_i + 1$ as in $\mathcal{A}$). OPT' will then assign the jobs from C into the earliest possible idle slots in between each of the jobs from H. The virtual schedule of OPT' may not be feasible but by Proposition 5.13 and the optimality of the Shortest Processing Time First rule when temperature is not considered, it must be that $|OPT| \geq |OPT'|$. We will denote the flow time, in the schedule of OPT', of a job J as $|J^*|$ and a set A as $|A^*|$.

We now analyse this virtual schedule. It must be that $|OPT'| = |C^*| + |H^*| = |C^*| + |H^*_{n-c}| + |(H - \{H_{n-c}\})^*|$. It must be that $|C^*| + |H^*_{n-c}| \geq |C| + |H_{n-c}| = 1 + (c^2 + 3c)/2$. We can work out the flow time of each job H_i for $1 \leq i < n - c$: $|H^*_1| = 1 + k_{c+1}$ and $|H^*_i| = |H^*_{i-1}| + k_{c+i}$, solving the recursion gives $|H^*_i| = 1 + \sum_{j=1}^{i} k_{c+j}$. This can be used to get the total flow time for the set $H - \{H_{c-n}\}$.

$$|(H - \{H_{n-c}\})^*| = \sum_{i=1}^{n-c-1} |H^*_i| = (n - c - 1) + \sum_{i=1}^{n-c-1} k_{c+i}((n - c - 1) - (i - 1))$$

These can then be combined to bound the flow time for OPT:

$$|OPT| \geq |OPT'| = 1 + \frac{c^2 + 3c}{2} + (n - c - 1) + \sum_{i=1}^{n-c-1} k_{c+i}((n - c - 1) - (i - 1)) \tag{5.3}$$

Combining Equations (5.2) and (5.3) gives us the competitive ratio:

$$\frac{|\mathcal{A}|}{|OPT|} \leq \frac{n^2 + n + 2\sum_{i=1}^{n-c-1} k_{c+i}((n - c - 1) - (i - 1))}{c^2 + c + 2n + 2\sum_{i=1}^{n-c-1} k_{c+i}((n - c - 1) - (i - 1))}$$

As every k_i value for a $J_i \in H$ must be at least one, we can work out the

minimum value for $\sum_{i=1}^{n-c-1} k_{c+i}((n-c-1)-(i-1))$ to be $(c^2-2cn+c+n^2-n)/2$. This gives a ratio of at most:

$$\frac{n^2+n+c^2-2cn+c+n^2-n}{c^2+c+2n+c^2-2cn+c+n^2-n} = \frac{c^2-(2n-1)c+2n^2}{2c^2-(2n-2)c+n^2+n} < \frac{c^2-2nc+2n^2}{2c^2-2nc+n^2}$$

Let $x = c/n$, then this ratio is equal to $\frac{x^2-2x+2}{2x^2-2x+1}$. For $0 \leq x \leq 1$, the maximum value of this ratio is equal to $(3+\sqrt{5})/2$, attained at $x = (3-\sqrt{5})/2$. □

5.5 Bounded Maximum Job Heat

Now we show some lower bounds for online algorithms to minimise the total flow time of a schedule when h_{max} is at most $R-\epsilon$ for some $0 < \epsilon < 1$. We consider this range of ϵ values because this is the only ϵ range that gives non-trivial results. No algorithm can give a bounded competitive ratio when h_{max} is allowed to be exactly R. This is because after scheduling any job with a non-zero heat contribution, any algorithm will have a positive temperature which means that the algorithm will never be able to schedule a job with heat R, and so any such jobs will end up with an infinite flow time.

If, on the other hand, h_{max} is restricted to be no hotter than $R-1$ then any job will be able to be scheduled at any time. This is because the maximum temperature of an algorithm is 1 and if the maximum heat of a job is $R-1$ then after running any job the temperature of any algorithm will be no more than $(1+R-1)/R = 1$, which means that the temperature threshold can never be violated. As the flow time of an algorithm's schedule increases with the queue size of the algorithm, the only way that an optimal

algorithm can be better than the online algorithm is if at some point both algorithms have the same queue size, but the online algorithm idles while the optimal algorithm schedules a job. However if the maximum heat of a job is limited to $R-1$, all jobs in the online algorithm's queue are always admissible. It is therefore trivial to show that any non-idling algorithm is 1-competitive.

Using standard 3-field notation this problem is denoted as $1|\textit{online-}r_i, h_i \leq R-\epsilon, p_i = 1|\sum F_i$.

5.5.1 Lower Bounds

We now show that no online algorithm is better than $\Omega(\log(1/\epsilon))$-competitive. In this proof we frequently use the fact that starting a hot job before a cool job gives a cooler final temperature than if the jobs are scheduled the other way around. Note that this still applies when swapping a hot job with an idle time slot.

Theorem 5.15. *For any integer $k \geq 2$, if $h_{max} = R-\epsilon$ where $\epsilon \leq (R-1)/R^k$ then any deterministic algorithm is at least k-competitive.*

Proof. Fix a deterministic algorithm $\mathcal{A}$. At time 0 release a job J_1 with $h_{J_1} = R-1$ and a job J_2 with $h_{J_2} = (R-1)/R^{k-1}$. $\mathcal{A}$ will start a job at some time t. We analyse the two cases.

Case 1: If $\mathcal{A}$ starts J_1 first then at time $t+1$ we release another job J_3 with $h_{J_3} = R-(R-1)/R^k$. $\mathcal{A}$ can either schedule J_2 or J_3 as its next job.

Case 1a: Suppose $\mathcal{A}$ starts the job J_2 first, then starts J_3 at some time u. We now show that $u \geq t+k+1$. If J_2 is started by $\mathcal{A}$ on or

after $t+k$ then by definition of the case J_3 cannot start before J_2 and so we must have that $u \geq t+k+1$. Otherwise J_2 is started earlier than $t+k$. In this case the temperature of $\mathcal{A}$ at time $t+k$ must be at least $\frac{R-1}{R^k} + \frac{(R-1)/R^{k-1}}{R^{k-1}} > (R-1)/R^k$, therefore J_3 will not be admissible until at least $t+k+1$. The temperature of $\mathcal{A}$ at $u+1$ (i.e. directly after J_3 has been scheduled) must be hotter than $h_{J_3}/R = 1-(R-1)/R^{k+1}$.

We then release a job J_4 with a heat contribution of $R-1/R^{k-1}$, starting at time $u+1$ and repeating every k time steps. As $1/R^{k-1} > (R-1)/R^k \geq \epsilon$ this job temperature is below $R-\epsilon$. We now show that $\mathcal{A}$ needs at least $k-1$ idle time steps to cool down before being able to schedule each of these J_4-jobs. For a J_4-job to be admissible the temperature must be no more than $1/R^{k-1}$. As $\tau_{u+1} > 1-(R-1)/R^{k+1}$, after $k-2$ idle slots from time $u+1$ the temperature of $\mathcal{A}$ becomes $(1-(R-1)/R^{k+1})/R^{k-2}$. This last term is greater than $1/R^{k-1}$ if and only if $(R-1)R^{1-2k}(R^k-1) > 0$ which is true as $R > 1$ and $k \geq 2$. So after $k-2$ idle slots $\mathcal{A}$ will still be too hot to schedule the first J_4-job.

We now show that it will not be possible for $\mathcal{A}$ to schedule the rest of the J_4-jobs without at least $k-1$ idle steps before each J_4-job that is scheduled. We assume that each J_4-job starts as soon as it is admissible in $\mathcal{A}$. Recall that swapping a hot job with an idle slot will increase the final temperature. As all of the J_4-jobs have the same heat contributions postponing a J_4-job when it is admissible cannot increase the number of J_4-jobs that $\mathcal{A}$ can

schedule in a given time interval, therefore the assumption is safe. We also know that each job must be admissible after at most $k-1$ idle time steps following the previous J_4-job because after $k-1$ idle time steps the temperature of $\mathcal{A}$ is at most $1/R^{k-1}$.

Consider the temperature of $\mathcal{A}$ at time $u+k+1$ in case 1a i.e. after the first J_4-job has been scheduled by $\mathcal{A}$

$$\tau_{u+k+1} > \left(1 - \frac{(R-1)/R^k}{R}\right)/R^k + \frac{R - 1/R^{k-1}}{R}.$$

After $k-2$ idle time steps we can then calculate the temperature of $\mathcal{A}$

$$\tau_{u+2k-1} > \frac{\left(1 - \frac{(R-1)/R^k}{R}\right)/R^k + \frac{R-1/R^{k-1}}{R}}{R^{k-2}}.$$

This is not only hotter than $1/R^{k-1}$ but also hotter than $(1 - \frac{(R-1)/R^k}{R})/R^{k-2}$ (i.e. the minimum value of τ_{u+k-1}). This means that the temperature of $\mathcal{A}$ just before it has scheduled this second J_4-job will be hotter than before it scheduled the first. This in turn means that after each J_4-job has been executed it will be hotter than after executing the last, and therefore between each J_4-job there must be $k-1$ idle time steps.

Case 1b: Alternatively $\mathcal{A}$ can start J_3 first at some time u. At $t+1$ the temperature of $\mathcal{A}$ is $(R-1)/R$. In order to start J_3 the temperature of $\mathcal{A}$ must be no hotter than $(R-1)/R^k$ and therefore we have that $u \geq t+k$.

If $u \geq t+k+1$ then we release a job J_4 with a heat contribution

of $R - 1/R^{k-1}$ every k time steps starting at $u + 1$ as in case 1a. Because the temperature at $u+1$ is the same as in case 1a, we can use the same argument as case 1a to show that $\mathcal{A}$ can only start each J_4-job after at least $k - 1$ idle time steps. $\mathcal{A}$ will also still have J_2 pending but scheduling J_2 either before the first J_4-job or between any J_4 jobs can only increase the temperature of $\mathcal{A}$, and will therefore not reduce the number of idle time steps $\mathcal{A}$ needs before scheduling the next J_4-job.

Otherwise it must be that $u = t + k$, and therefore $\tau_{u+1} = 1$. In this case we release a job J_4 with a heat contribution of $R-1/R^{k-1}$ every k time steps starting at $u + 2$. Again we can safely assume that these J_4-jobs are started as soon as they become admissible, and they will become admissible in $\mathcal{A}$ after at most $k - 1$ idle steps. If J_2 is scheduled before the first J_4-job then at $u + k$ the temperature of $\mathcal{A}$ will be at least $\frac{1}{R^{k-1}} + \frac{(R-1)/R^{k-1}}{R^{k-1}} > 1/R^{k-1}$, meaning that J_4 will not be admissible until $u+k+1$. In a similar way to case 1a we can calculate the temperature after scheduling the first J_4-job

$$\tau_{u+k+2} \geq \frac{1/R + (R-1)/R^k}{R^k} + \frac{R - 1/R^{k-1}}{R} = 1 - \frac{(R-1)(R^k - R)}{R^{1+2k}}.$$

After $k-2$ idle time steps we can again calculate the temperature of $\mathcal{A}$ for this case

$$\tau_{u+2k} \geq \frac{1 - \frac{(R-1)(R^k-R)}{R^{1+2k}}}{R^{k-2}}.$$

This last term is not only greater than $1/R^{k-1}$ but hotter than the minimum possible value of τ_{u+k} iff $(R-1)(R^k-1)(R^k-R) > 0$, which is true as $R > 1$ and $k \geq 2$. Therefore it must be that $\mathcal{A}$ needs more than $k-2$ idle slots before being able to schedule the next J_4-job. This also means that the temperature of $\mathcal{A}$ after it has scheduled this second J_4-job is again hotter than after it scheduled the first, and as this will only increase with each job it must be that between each J_4-job there must be $k-1$ idle time steps in $\mathcal{A}$.

Otherwise J_2 is not started before the first J_4-job. In this case $\tau_{u+k} = 1/R^{k-1}$, meaning J_4 becomes admissible by $u+k$ at the earliest, and so will be started at this time giving $\tau_{u+k+1} = 1$. As $\tau_{u+k+1} = \tau_{u+1}$ we can actually infer that if J_2 remains unscheduled by $\mathcal{A}$ for the time interval $[u, u+ik)$, that for any $i \geq 1$, $\tau_{u+ik} = 1/R^{k-1}$, and so the next J_4-job will be scheduled at $u+ik$ giving $\tau_{u+ik+1} = 1$. When the job J_2 is scheduled in some interval $[u+ik+1, u+(i+1)k+1)$, we can use similar calculations to the case where J_2 is scheduled before the first J_4-job to show that the next J_4-job won't become admissible until at least $u+(i+1)k+1$. It then also follows that from this point there must again be $k-1$ idle time steps in between each J_4-job in $\mathcal{A}$. We will show that J_2 cannot be indefinitely postponed.

In either of these cases OPT schedules jobs J_2, J_3 and J_1 at time t, $t+1$ and $t+2$ respectively, this gives $\tau'_{t+3} = 1$. OPT will then idle until

J_4 is released. As the release time of the first J_4-job is always at least $t + k + 2$ the temperature of OPT will be at most $1/R^{k-1}$ when the first J_4-job is released and so can schedule it immediately. As there are then $k - 1$ idle steps before the next J_4-job is released OPT is clearly able to schedule each of the J_4-jobs as soon as they are released.

Consider the case when in total x copies of J_4 are released. Denote the flow time that OPT incurs from the jobs J_1, J_2 and J_3 as μ. OPT will get a flow time of 1 for each of the x J_4-jobs and so for case 1 will get a total flow time of $\mu + x$. Meanwhile for case 1a and the case 1b where J_2 is started before the first J_4-job the flow time of $\mathcal{A}$ for each of the x J_4-jobs will be kx giving a total flow time of greater than kx (as we are ignoring the flow time of J_1, J_2 and J_3). For case 1b where J_2 is not started before the first i J_4 jobs for some $0 \leq i \leq x$ the flow time of the J_4-jobs and J_2 gained from time $u + 1$ will be at least $(k-1)i+ki+(x-i)k \geq kx$. As x can be made arbitrarily large, we can ignore the constant value of μ giving a competitive ratio of arbitrarily close to k. If J_2 gets postponed after x J_4-jobs then the flow time of J_2 in $\mathcal{A}$ alone is already at least xk, so no more J_4-jobs need to be released and the argument still works.

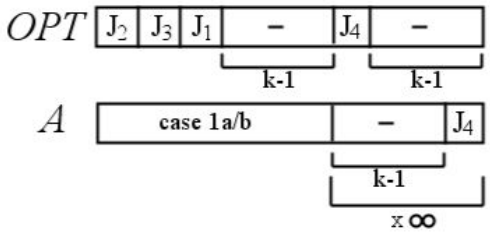

Figure 5.1: Theorem 5.15 Case 1 overall

Case 2: Otherwise $\mathcal{A}$ starts job J_2 at time t. Suppose J_1 is started by $\mathcal{A}$ at some time $v \geq t+1$. At time $v+1$ we release a job J_3 with $h_{J_3} = R-(R-1)/R^k$ that will be started by $\mathcal{A}$ at some time u. After $k-1$ idle slots the temperature of $\mathcal{A}$ at $v+k$ is $\frac{(R-1)+(R-1)/R^k}{R^k} > (R-1)/R^k$. This means that we must have $u \geq v+k+1$.

Meanwhile OPT will start J_2, J_3 and J_1 at time t, $v+1$ and $v+2$. Note that this will be possible because $v+1 \geq t+2$ and $\tau'_{t+2} = (R-1)/R^{k+1}$, so J_3 will be admissible from this time, and J_1 is always admissible.

Next we release a job J_4 with a heat contribution of $R-1/R^{k-1}$, starting at time $u+1$ and repeating every k time steps. Because the minimum temperature of $\mathcal{A}$ at $u+1$ in this case is the same as the minimum at $u+1$ in case 1a we can use the same argument as in case 1a to show that OPT is able to schedule each of these jobs as soon as they are released, while $\mathcal{A}$ will have a delay of $k-1$ time steps per job before they can be scheduled. Again this can be repeated a large amount of times to give a competitive ratio of k, thus concluding the proof.

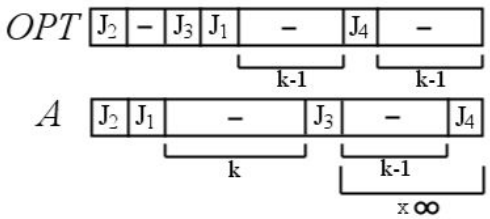

Figure 5.2: Theorem 5.15 Case 2

□

The above theorem only gives a non-trivial result when $\epsilon < (R-1)/R^2$.

The next theorem gives a bound that is not as strong but holds for any $\epsilon < 1$.

Theorem 5.16. *Any deterministic algorithm is at least* 2*-competitive when* $h_{max} = R - 1 + \delta$ *for any* $\delta > 0$.

Proof. Fix a deterministic algorithm $\mathcal{A}$. At time 0 we release a B-block of size x, for some large enough x to be explained later. Initially we assume $\mathcal{A}$ will always schedule a job when it has one that is admissible, we will return to this assumption later. This means that $\mathcal{A}$ will schedule all x of these B-jobs and have temperature of $1 - \alpha$ at time x where $\alpha = 1/R^x$. At this time step we then release a B-job, B_1, and a job J_1 with $h_{J_1} = R - 1 - R\delta + \alpha$.

If $\mathcal{A}$ schedules B_1 at time x then immediately afterwards its temperature is $1 - \alpha/R$. At time $x + 1$ we then release a job J_2 with a heat contribution of $R - 1 + \delta$. $\mathcal{A}$ will be too hot to schedule job J_2 as long as $\alpha/R < \delta$. We can make α arbitrarily small by choosing a large enough x, so this is always possible. $\mathcal{A}$ will therefore start J_1 instead. This gives $\mathcal{A}$ a temperature of $1 - \delta + \alpha/R - \alpha/R^2$ and so $\mathcal{A}$ is still too hot to schedule the job J_2 and must idle at $x + 2$ (as it has no other jobs in its queue). Meanwhile OPT will schedule J_1, J_2 and B_1 at times x, $x+1$ and $x+2$ respectively. We then release a B-job in each of the next y time steps. During these time steps $|Q_t|$ is at least 2 while $|Q'_t| = 1$. The incremental flow time in these time steps is therefore at least $2y$ for $\mathcal{A}$ and y for OPT, and thus for large y the competitive ratio is arbitrarily close to 2.

Otherwise $\mathcal{A}$ schedules J_1 at time x. At time $x + 1$ no new jobs are released. $\mathcal{A}$ will schedule B_1 as it is admissible, giving a temperature of $1 - \delta/R$. Meanwhile OPT will schedule B_1 at time x and J_1 at $x + 1$ and

end up with a temperature of $1 - \delta + \alpha/R - \alpha/R^2$. At time $x + 2$ we then release a job J_3 with $h_{J_3} = R - 1 + \delta - \alpha/R + \alpha/R^2$. As we already have that $\alpha/R < \delta$, we have that while OPT can schedule this job $\mathcal{A}$ cannot and so must remain idle. We then continue to release y B-jobs and as before get a competitive ratio arbitrarily close to 2.

We now return to the assumption that $\mathcal{A}$ will schedule a job whenever it has one that is admissible. Suppose $\mathcal{A}$ idles at some time t when it has an admissible job in its queue. We can easily see from the above cases that it must be that $|Q_t| \geq |Q'_t|$ for any possible t. It is also easy to see that at any time in any of the above cases, OPT can construct a schedule with the jobs remaining in its queue that has no idle time. We then stop releasing any further jobs for the next $|Q'_t|$ time steps. At each time $t + i$ for every $0 \leq i < |Q'_t|$, OPT will schedule a job in its queue. Starting at time $t + |Q'_t|$ we then release each of the y B-jobs at each time step as before. Again $|Q'_u| = 1$ during each of these time steps u while $|Q_u| \geq 2$ as $\mathcal{A}$ idled at least in time step t. Again y can be made arbitrarily large so we get a competitive ratio of arbitrarily close to 2. □

5.5.2 Non-idling algorithms

We now give lower and upper bounds on the performance of non-idling algorithms. The analysis is tight within constant factors, for any fixed R and ϵ.

We first show a lower bound on the competitive ratio for this algorithm class. This lower bound holds where the maximum heat of a job is bounded to

$R-1+\delta$ for any small $\delta > 0$ which means that this class of algorithms cannot ever achieve a constant competitive ratio for any non-trivial maximum job heat.

Theorem 5.17. *When $h_{max} = R - 1 + \delta$ for any $\delta > 0$, any non-idling algorithm is at least $\Omega(n)$-competitive for any fixed R and δ.*

Proof. Fix a non-idling algorithm $\mathcal{A}$. The proof uses many jobs that are organised as a large number of phases, each in turn contains a large number of sub-phases. We begin with a description of sub-phases and phases. The sub-phases are illustrated in Figure 5.3, while the overall proof construction is illustrated in Figure 5.4. The proof uses a large number of K-jobs, all with heat contribution $R - 1 + \delta$. We also fix a large enough integer x that is a constant depending on R and δ.

We now describe a sub-phase. Each sub-phase contains either $x + 4$ or $x + 5$ time steps. We maintain the following invariants: at the start and end of each sub-phase, $\mathcal{A}$'s temperature is at least $1 - 1/R^x$. Moreover, at the beginning of a sub-phase, OPT only has B-jobs in its queue while $\mathcal{A}$ only has K-jobs in its queue. At the end of the sub-phase, OPT will have one more B-job and $\mathcal{A}$ will have one more K-job in their queues. Finally, we make sure that K-jobs are not admissible at any time during a sub-phase i.e. the temperature is hotter than $1 - \delta$ throughout. This is clearly true at the beginning of a sub-phase.

Let time t be the starting time of a sub-phase. At time t we release a job J_1 with $h_{J_1} = R - 1 - \alpha$ for some small $0 < \alpha < R\delta/2$ and another job J_2 with $h_{J_2} = R - 1 - R\delta + \alpha$. Both J_1 and J_2 are admissible for $\mathcal{A}$ at t while

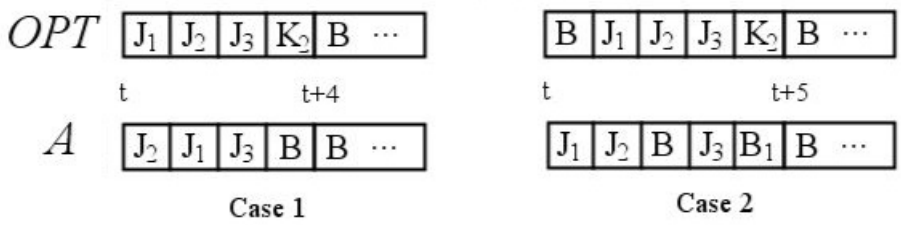

Figure 5.3: Theorem 5.17 sub-phases

any K-jobs are not, and so $\mathcal{A}$ must schedule either J_1 or J_2.

Case 1: $\mathcal{A}$ schedules J_2 at time t. We release the following additional jobs: at $t+2$ we release a job J_3 with $h_{J_3} = R-1-(R-1)\delta-\alpha/R+\alpha/R^2$, and at time $t+3$ we then release one B-job and one K-job. $\mathcal{A}$ must schedule J_1, J_3 and the B-job at time $t+1, t+2, t+3$ respectively: we show that any K-jobs are not admissible at any of these time steps. Note that $\tau_t = 1 - 1/R^x$ can be made arbitrarily close to 1; for simplicity we first assume $\tau_t = 1$ here. Then we have:

$$\tau_{t+1} = 1 - \delta + \alpha/R; \quad \tau_{t+2} = 1 - \delta/R - \alpha/R + \alpha/R^2;$$

$$\tau_{t+3} = 1 - \frac{\delta(R^2 - R + 1)}{R^2} - \frac{2\alpha(R-1)}{R^3}$$

It is clear that at $t+1$ and $t+2$, $\mathcal{A}$ is hotter than $1-\delta$ as $R > 1$ and $0 < \alpha < R\delta/2$. We also have that $\tau_{t+3} > 1 - \delta$ as long as $(R-1)(\delta R - 2\alpha) > 0$ which is true by choice of δ. Therefore the K-jobs are not admissible at these times. The fact that τ_t is not 1 but $1 - 1/R^x$ means the temperatures are actually slightly lower than indicated above, but we can choose a large enough x to ensure the difference is small enough so that any K-job remains not admissible.

Meanwhile OPT will schedule J_1, J_2, J_3 and the K-job at time t, $t+1$, $t+2$ and $t+3$. As both J_1 and J_2 are cooler than $R-1$, they are admissible at any time step. It can also be verified that $\tau'_{t+3} \leq 1-\delta$ and so the K-job is admissible at $t+3$ as well, so this schedule is indeed feasible.

At $t+4$ we then release another B-block of size x. Both $\mathcal{A}$ and OPT will schedule B-jobs in the next x steps. (We know that $\tau_{t+4} > \tau_{t+3} > 1-\delta$ as $\mathcal{A}$ scheduled a B-job at $t+3$, so $\mathcal{A}$ is again too hot to schedule K-jobs at any of these steps.) At time $t+4+x$ the temperature of $\mathcal{A}$ therefore returns to at least $1-1/R^x$ as at the start of the case. At $t+4$, OPT has one more B-jobs in its queue comparing to t, while $\mathcal{A}$ has one more K-jobs.

Case 2: Otherwise $\mathcal{A}$ schedules J_1 at time t. At $t+2$ and $t+3$ we release a B-job and J_3 respectively, and at time $t+4$ we then release a B-job and a K-job. $\mathcal{A}$ must schedule J_2, a B-job, J_3 and the other B-job at time $t+1$ to $t+4$, in this order: we show that $\mathcal{A}$ will be too hot to schedule any K-job in any of these steps. As before we can assume $\tau_t = 1$.

$$\tau_{t+1} = 1-\alpha/R; \quad \tau_{t+2} = 1-\delta+\alpha/R-\alpha/R^2;$$

$$\tau_{t+3} = 1-\delta/R+\alpha/R^2-\alpha/R^3; \quad \tau_{t+4} = 1-\frac{\delta(R^2-R+1)}{R^2}-\frac{\alpha(R-1)^2}{R^4}$$

It is clear that at $t+1$, $t+2$ and $t+3$ that $\mathcal{A}$ is hotter than $1-\delta$ as $R>1$ and $0<\alpha<R\delta/2$. We also have that $\tau_{t+4} > 1-\delta$ as long as $(R-1)(\delta R^2+\alpha-\alpha R) > 0$ and as $(R\delta)/2 > \alpha > 0$ and $R>1$ this

must also be true.

Meanwhile OPT schedules a B-job (from the already released B-block), J_1, J_2, J_3 and a K-job at time t to $t+4$, in this order. It can be verified that $\tau'_{t+4} \leq 1-\delta$, so a K-job is indeed admissible at $t+4$. At $t+5$ we then release another B-block of jobs of size x. Again we know that $\tau_{t+5} > \tau_{t+4} > 1-\delta$ as $\mathcal{A}$ scheduled a B-job at $t+4$, so both $\mathcal{A}$ and OPT will schedule B-jobs in the next x steps, and that the temperature of $\mathcal{A}$ returns to close to at least $1-1/R^x$ as at the start of the case. At this point OPT will again have one more B-job in its queue while $\mathcal{A}$ will have one more K-job, comparing to time t.

This completes the description of a sub-phase. We now define a phase. We maintain the invariants that whenever a phase begins, $\mathcal{A}$ has at least one K-job in its queue while OPT has exactly two B-jobs in its queue. Moreover $\mathcal{A}$'s temperature is at least $1-1/R^x$ at the beginning and end of the phase. Finally, at the end of a phase, OPT again has two B-jobs in its queue whereas $\mathcal{A}$ will have one more K-job comparing to the beginning of the phase.

For a phase that begins at time t, it consists of y sub-phases of jobs, the first one at time t and each subsequent one immediately following the end of the previous one, for some $y \geq 1$ to be defined later. From the properties of a sub-phase it follows that, at the end of the y-th sub-phase, OPT has $y+2$ B-jobs in its queue while $\mathcal{A}$ has y new K-jobs (in addition to whatever number of K-jobs it has at the start of the phase). Let u be the time when the y-th sub-phase ends. No jobs are released for the next y time steps. In each of these time steps OPT will schedule one of the B-jobs in its queue, so

in the end it returns to having two B-jobs in its queue. $\mathcal{A}$ meanwhile will be too hot to schedule any K-jobs at time u and so will idle for one step, then schedule $y-1$ K-jobs that it still has pending. y needs to be chosen such that exactly $y-1$ K-jobs scheduled consecutively by $\mathcal{A}$ starting at time $u+1$ brings its temperature to larger than $1-\delta$ i.e. $\mathcal{A}$ cannot schedule the yth job in its queue with that heat contribution. Finally at $u+y$ we release a B-block of size x, which both $\mathcal{A}$ and OPT schedule, bringing $\mathcal{A}$'s temperature to at least $1-1/R^x$. This completes the description of one phase.

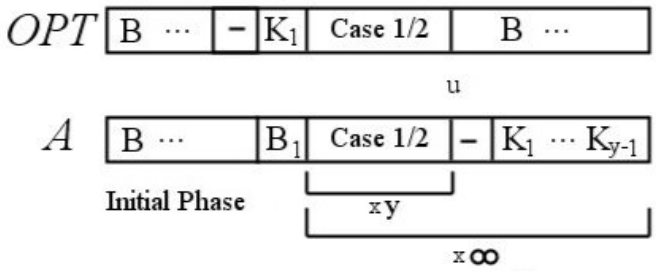

Figure 5.4: Theorem 5.17 Overall Construction

We can now describe the complete construction. At time 0 we start an initial phase: we release a B-block of size x. $\mathcal{A}$ will schedule all of these jobs. At time x we release another B-job, and a K-job. x is large enough so that $\mathcal{A}$ is too hot to schedule the K-job, so $\mathcal{A}$ will have to schedule B_1. Meanwhile OPT will schedule the first $x-1$ B-jobs from the B-block then idle for one time step and schedule the K-job when it is released. At this time $\mathcal{A}$ will have only a K-job in its queue while OPT will have two B-jobs. $\mathcal{A}$ will have a temperature of $1-1/R^x$. Thus, we have established the starting conditions required for a phase.

We then release a large number of phases c, each immediately following

the preceding one. Each phase involves the release of $\Theta(y)$ sub-phases, while each sub-phase involves the release of $\Theta(x)$ jobs. The value of y will be the same for each phase, because this value depends on the temperature of $\mathcal{A}$ at time u (the time straight after the yth sub-phase ends). However, as the last x time steps of each sub-phase are a B-block, this will bring the temperature of $\mathcal{A}$ at u to very close to the same temperature after each sub-phase in every phase. An x value can always be chosen so that the temperature at u in each phase is arbitrarily close to the same temperature, meaning that the y value will be the same for each phase. Each sub-phase involves $x+4$ or $x+5$ jobs, therefore each phase involves the release of $\Theta(yx)$ jobs, and at the end of each phase the queue size of $\mathcal{A}$ will be one larger than before the phase starts, but OPT's remains the same. Although these conditions are not exactly as in Lemma 5.1, as there is only very small variation in phase lengths and y the analysis of the Lemma still holds, giving a competitive ratio of $\Omega(c)$. For any fixed w, x and y values (which depend on fixed R and ϵ values) $n = \Theta(cy(x+w)) = \Theta(c)$, therefore the competitive ratio of $\mathcal{A}$ will be $\Omega(n)$. □

We now show an upper bound on the performance of non-idling algorithms.

Theorem 5.18. *For any integer $k \geq 1$, if the maximum heat of a job is $R-\epsilon$ where $\epsilon \geq (R-1)/R^k$, any non-idling algorithm is $O(kn)$-competitive.*

Proof. First we note that any job with the maximum heat contribution of $R-\epsilon$ can be scheduled after k idle time steps. OPT can schedule no more than 1 job per time step and so for an instance with n jobs we know that

$|OPT| \geq n$. $\mathcal{A}$ meanwhile will schedule at least 1 job every k time steps, with the first job being completed immediately. We now calculate the flow time for the ith job J_i that is scheduled by $\mathcal{A}$.

$$|J_1| = 1; \quad |J_i| \leq (i-1)k + i;$$

We can now use this to upper bound the total flow time of the schedule produced by $\mathcal{A}$

$$|\mathcal{A}| = \sum_{i=1}^{n} |J_i| \leq 1 + (kn^2 - kn + n^2 + n - 2)/2 \leq O(kn^2)$$

Therefore we have that $|\mathcal{A}|/|OPT| \leq O(kn)$. □

Comparing this to the lower bound $\Omega(k)$ of the problem (Theorem 5.15) this shows that ignoring constant factors this upper bound is a factor of $O(n)$ larger than the lower bound. For a fixed R value the upper bound becomes $O(n)$, and so this analysis is tight within constant factors.

5.6 Increased Thermal Threshold

In this section we look at the case where the temperature threshold of the online algorithm is increased. We redefine ϵ so that the online algorithm has a thermal threshold of $1 + \epsilon$ for some $0 < \epsilon < \frac{1}{R-1}$ but the temperature threshold of OPT remains at 1. (Note that the maximum heat contribution of a job remains at R.) We limit the value of ϵ to less than $\frac{1}{R-1}$ because if a larger value is allowed then, in a similar way to the case where we bound

job heats to a maximum of $R-1$ (Section 5.5), any algorithm can always schedule any job at any time step and therefore any non-idling algorithm is trivially 1-competitive. Using standard 3-field notation the problem is defined as $1|\textit{online-}r_i, h_i, p_i = 1|\sum F_i$.

This model is a form of resource augmentation as the power of the online algorithm is being increased, relative to the adversary. This increased thermal threshold could be achieved by improving the quality of the components in the microprocessor, for example.

5.6.1 Lower Bounds

Theorem 5.19. *For any integer $k \geq 1$, if $\epsilon < (R-1)/R^{k+1}$ then no deterministic algorithm is better than k-competitive.*

Proof. At time 0 release a job J_1 with $h_{J_1} = R-1$. $\mathcal{A}$ will start J_1 at some time t. At $t+1$ we release a job J_2 with $h_{J_2} = R$ which will be started by $\mathcal{A}$ at some time u. It must be that $u \geq t+k$ because otherwise the temperature of $\mathcal{A}$ would be at least $(R-1)/R^{k+1} + 1 > 1 + \epsilon$ after scheduling J_2 at any time earlier than $t+k$. Meanwhile OPT schedules J_2 and J_1 at time $t+1$ and $t+2$ respectively.

Starting at time $u+1$, i.e. immediately after $\mathcal{A}$ has completed J_2, we release a job J_3 with $h_{J_3} = R - 1/R^{k-1}$ every k time steps. We can safely assume that each J_3-job starts as soon as it is admissible in $\mathcal{A}$ because all J_3-jobs have the same heat contribution and so postponing a J_3-job cannot increase the number of J_3-jobs that can be scheduled in any given interval. Consider the first such job at time $u+1$. $\mathcal{A}$ has a temperature of at least 1 at

this time step (because it has just complete J_2), and so would not be able to schedule J_3 for the next $k-1$ time steps as long as $1/R^{k-1}+(R-1/R^{k-1})/R > 1+\epsilon$, i.e. $\epsilon < \frac{1}{R^{k-1}} - \frac{1}{R^k}$. Note that $\frac{R-1}{R^{k+1}} < \frac{1}{R^{k-1}} - \frac{1}{R^k}$ for all $R > 1$, therefore the requirement $\epsilon < \frac{1}{R^{k-1}} - \frac{1}{R^k}$ is satisfied as $\epsilon < \frac{R-1}{R^{k+1}}$. Meanwhile at $u+1$, OPT has a temperature of at most $1/R^{k-1}$ (as OPT has idled between $t+2$ and $u+1 \geq t+k+1$), so is able to start J_3 immediately. OPT will then idle for $k-1$ steps to cool to a temperature of at most $1/R^{k-1}$.

The same holds for subsequent J_3 jobs: OPT will always have been idle for $k-1$ time steps before each job is released and so will be able to schedule each J_3-job immediately, while as long as $\epsilon < \frac{1}{R^{k-1}} - \frac{1}{R^k}$ $\mathcal{A}$ will only be able to schedule each job after it has already been released for $k-1$ time steps. After scheduling a J_3-job the temperature of $\mathcal{A}$ will always be greater than 1. Therefore in between each J_3-job there must be at least $k-1$ idle steps in $\mathcal{A}$ by the same argument as why $\mathcal{A}$ has to idle for $k-1$ steps before starting the first of the J_3-jobs.

If in total x such J_3 jobs are released then the flow time of $\mathcal{A}$ is at least xk while OPT will have a flow time of x for these jobs. As x can be made arbitrarily large we can ignore the flow time generated by J_1 and J_2, giving a competitive ratio arbitrarily close to k. □

The following is a direct consequence of the above theorem.

Corollary 5.20. *No algorithm can be 1-competitive when* $\epsilon < (R-1)/R^2$.

This contrasts with the upper bound to be given in Section 5.6.3 which shows that Hottest First is 1-competitive whenever $\epsilon \geq \frac{R^2+R+1}{(R-1)(R+1)^2}$.

5.6.2 Some Bad Algorithms

We now examine the limitations of some algorithms for this model. We first show a counter example for all non-idling algorithms that holds for a limited range of ϵ. We then show a lower bound for Coolest First, which is a non-idling algorithm, which holds for every possible ϵ.

We now show a lower bound that gives a competitive ratio of $\Omega(n)$ but for the restricted group of non-idling algorithms. This lower bound holds as long as $\epsilon < 1/R^2$.

Theorem 5.21. *If $\epsilon < 1/R^2$ then any non-idling algorithm is at least $\Omega(n)$-competitive.*

Proof. Fix a non-idling algorithm $\mathcal{A}$. Similar to Theorem 5.17, the proof uses many jobs that are organised as a large number of phases, each in turn contains a large number of sub-phases. We begin with a description of sub-phases and phases. We fix some values x and y such that both are large enough, to be described later. These values depend on R and ϵ. This proof uses a large number of K-jobs, all with heat contributions of $R - 1/R^y$.

We now describe a sub-phase. Each sub-phase contains $x + 1$ time steps. We maintain the following invariants: at the start and end of a sub-phase the temperature of $\mathcal{A}$ is at least $1 - 1/R^x$ and at no time step during a sub-phase is the temperature of $\mathcal{A}$ any less than $1/R$. At the start of each sub-phase OPT will have only B and Z-jobs in its queue, with a total of $y + 1$ jobs and at least one B-job. At the end of each sub-phase the queue of OPT will contain one less B-job and one more Z-job than at the start of the sub-phase. The queue of $\mathcal{A}$ will contain the only K-jobs at the start and

end of a sub-phase, with the number of K-jobs being the same at both the start and end of each sub-phase. The temperature constraint on $\mathcal{A}$ ensures that no K-jobs are admissible during the sub-phase.

Let t be the starting time of a sub-phase. At t we release a Z-job. As $\mathcal{A}$ only has K-jobs already in its queue and these are not admissible at t, $\mathcal{A}$ must schedule the Z-job at this time. At $t+1$ we release a B-block of size x. As $1-1/R^x$ can be made arbitrarily close to 1, for simplicity we will first assume that $\tau_t = 1$ here. This means that $\tau_{t+1} = 1/R$. We will now show that x and y values can always be chosen so that K is not admissible at this time step. K is not admissible as long as $1/R + R - 1/R^y > R(1+\epsilon)$. This is true as long as $\epsilon < 1/R^2 - 1/R^{y+1}$, therefore we can always choose a large enough y to ensure that any K-job remains not admissible as long as $\epsilon < 1/R^2$. The fact that τ_t is not 1 but at least $1 - 1/R^{x+1}$ means the temperatures are actually slightly lower than indicated above but we can choose a large enough x to ensure the difference is small enough so that any K-job remains not admissible. Therefore for each time step $t+i$ for $1 \leq i \leq x$ $\mathcal{A}$ will start a B-job from the B-block, as scheduling each B-job will only raise the temperature of $\mathcal{A}$. This means that the invariants for $\mathcal{A}$ are maintained.

Meanwhile OPT will start one of the B-jobs in its queue at time t, and then run each job from the B-block when it is released. It is clear that this will also maintain the invariants for OPT.

This completes the description of a sub-phase. We now define a phase. We maintain the invariants that whenever a phase begins and ends the temperature of $\mathcal{A}$ is at least $1-1/R^x$. At the start and end of a phase the queue of OPT will contain exactly $y+1$ B-jobs. Meanwhile the queue of $\mathcal{A}$ at the

start of each phase will contain at least one K-job, and at the end of each phase will contain one extra K-job than at the start of the phase.

For a phase that begins at time t we release y consecutive sub-phases, the first starting at time t and then each subsequent sub-phase starting immediately after the end of the previous one. From the properties of a sub-phase it follows that after these y sub-phases OPT will have only y Z-jobs and one B-job in its queue, while the queue of $\mathcal{A}$ will contain only the same number of K-jobs as at the start of the phase.

Let u be the time the yth sub-phase ends. At u we release a B-block of size y which $\mathcal{A}$ must schedule entirely as any of the K-jobs in its queue will still not be admissible at any of these time steps. Meanwhile OPT schedules all y Z-jobs in its queue consecutively bringing its temperature to at most $1/R^y$. At time $u+y$ after the B-block has finished being released, we release a K-job. $\mathcal{A}$ will still be too hot to schedule it and so remains idle, while OPT will schedule the new K-job. We then release another B-block of size x which will be scheduled by both $\mathcal{A}$ and OPT (as the K-jobs are still too hot to be admissible in $\mathcal{A}$) bringing the temperature of $\mathcal{A}$ to at least $1-1/R^x$. This clearly maintains the invariants for both OPT and $\mathcal{A}$.

We now describe the complete construction. At time 0 we start an initial phase. At this time we release a B-block of size $x+y+1$ and at time $x+y$ we release a K-job. $\mathcal{A}$ will schedule all of the jobs from the B-block, and not the K-job because by the time the K-job is released $\mathcal{A}$ will already be too hot to schedule it, and so will schedule the final B-job. Meanwhile OPT will schedule the first x B-jobs, then idle for y time steps, and schedule the K-job as soon as it is released. After this time $\mathcal{A}$ will be hotter than $1-1/R^x$ and

its queue will contain only a K-job, while OPT's queue will contain $y+1$ B-jobs. Thus, we have established the conditions for the start of a phase.

We then release a large number of phases c, each immediately following the preceding one. It is clear that each sub-phase involves the release of $x+1$ jobs, and so each phase involves the release of $\Theta(yx)$ jobs. At the end of each phase the queue size of $\mathcal{A}$ will have increased, and so the conditions of Lemma 5.1 are satisfied giving a competitive ratio of $\Omega(c)$. For any fixed x and y values (which depend on fixed R and ϵ values) $n = \Theta(cyx) = \Theta(c)$, therefore the competitive ratio of $\mathcal{A}$ will be $\Omega(n)$. □

We now consider the algorithm Coolest First. This is the algorithm that schedules a job whenever one is admissible, and if several jobs are admissible it schedules the coolest one. Coolest First is a non-idling algorithm so Theorem 5.21 applies, but we now give a stronger bound that shows Coolest First has a competitive ratio of $\Omega(n)$ for all non-trivial values of ϵ.

Theorem 5.22. *If $\epsilon < 1/(R-1)$ then Coolest First has a competitive ratio of at least of $\Omega(n)$.*

Proof. Again we describe the adversary using sub-phases and phases. We begin with a description of these sub-phases and phases. We fix some values x, w and y such that all are large enough, to be described later. These values depend on the values of R and ϵ. This proof uses a large number of K-jobs, all with heat contributions of $R - 1/R^w$.

We now describe a sub-phase. Each sub-phase lasts $x+w+1$ time steps. We maintain the following invariants: at the start of each sub-phase the queue of $\mathcal{A}$ will either be empty or contain only K-jobs, and at the end of

each sub-phase the queue of $\mathcal{A}$ will contain one extra K-job compared to at the start of the sub-phase. At the start of each sub-phase the queue of OPT will either be empty or contain only Z-jobs, and at the end of each sub-phase the queue of OPT will contain one extra Z-job compared to at the start of the sub-phase.

Let t be the starting time of a sub-phase. At t we release a B-block of size x and at time $t+x$ we then release a Z-block of size w. At $t+x+w$ we release 2 jobs, a Z-job and a K-job. As B- and Z-jobs are all cooler than the K-jobs that are already in $\mathcal{A}$'s queue, $\mathcal{A}$ will schedule the complete B-block, followed by the Z-block and at $t+x+w$ the newly released Z-job. This means the K-job will be in the queue of $\mathcal{A}$ at this time which is the end of the sub-phase, maintaining the invariant.

Meanwhile OPT will also schedule the whole of the B- and Z-blocks. OPT will then schedule the K-job as soon as it is released at $t+x+w$. As OPT will be no hotter than $1/R^w$ at time $t+x+w$, the K-job will be admissible. This means the Z-job will be in the queue of OPT at this time which is the end of the phase, maintaining the invariant.

This completes the description of a sub-phase. We now define a phase. We maintain the invariants that whenever a phase ends the queue of OPT will be empty, while the queue of $\mathcal{A}$ will contain an extra K-job compared to the start of the phase.

For a phase that begins at time t we release y consecutive sub-phases, the first starting at time t, and then each subsequent sub-phase starting immediately after the end of the previous one. From the properties of a sub-phase it follows that after these y sub-phases OPT will have y Z-jobs in its

queue and $\mathcal{A}$ will have at least y K-jobs.

Let u be the time the yth sub-phase ends. At time u we stop releasing jobs for y time steps. OPT will schedule the y Z-jobs in its queue and at time $u+y$ have an empty queue. Meanwhile $\mathcal{A}$ can schedule at most $y-1$ of the K-jobs in its queue consecutively and be too hot to schedule a K-job in the time step $u+y-1$, we now show this must be the case. At time u the temperature of $\mathcal{A}$ must be close to $1/R^{w+1}$, and as the temperature of $\mathcal{A}$ can be made arbitrarily close to this value by increasing x, we will assume for simplicity that $\tau_u = 1/R^{w+1}$. We now show that if the yth K-job is scheduled then the temperature of $\mathcal{A}$ will be hotter than $1+\epsilon$. The temperature of $\mathcal{A}$ after it scheduled the yth K-job can be calculated as

$$\frac{1}{R^{w+1+y}} + \sum_{k=1}^{y} \frac{R - 1/R^w}{R^k}.$$

This will be larger than $1+\epsilon$ as long as

$$R^{-w-y-1} + \frac{R^{-w-y}}{R-1} - \frac{R^{-w}}{R-1} - \frac{R^{1-y}}{R-1} + R/(R-1) - 1 > \epsilon$$

As both w and y can be made arbitrarily large then we can make $R^{-w-y-1} + \frac{R^{-w-y}}{R-1} - \frac{R^{-w}}{R-1} - \frac{R^{1-y}}{R-1}$ arbitrarily small which means that it is not possible to schedule the yth K-job as long as $\epsilon < R/(R-1) - 1 = 1/(R-1)$. This is the end of the phase, and so the invariants have been maintained.

For the complete construction we release a large number of phases c, each immediately following the preceding one. It is clear that each phase involves the release of $\Theta(x+w)$ jobs, and so each phase involves the release of

$\Theta(y(x+w))$ jobs. After each phase the queue size of $\mathcal{A}$ is increased, while the queue size of OPT will stay the same, so each phase satisfies the conditions of Lemma 5.1 giving a competitive ratio of $\Omega(c)$. For any fixed w, x and y values (which depend on fixed R and ϵ values) $n = \Theta(cy(x+w)) = \Theta(c)$, therefore the competitive ratio of $\mathcal{A}$ will be $\Omega(n)$. □

5.6.3 A 1-Competitive Algorithm

We now show an upper bound for the Hottest First algorithm, that is 1-competitive when $\epsilon \geq \frac{R^2+R+1}{(R-1)(R+1)^2}$. We do this by showing that for every time step t, the number of idle steps by OPT before t is less than the number of idle time steps by $\mathcal{A}$ before t. This shows that the queue of OPT must be at least the size of the queue of $\mathcal{A}$ and therefore the flow time of OPT must be at least that of $\mathcal{A}$.

First we split all jobs into two classes, any job J with $R^2/(R+1) < h_J (\leq R)$ is called an H-job, and every other job is called a C-job. We now show three lemmas regarding the properties of H and C-jobs.

Lemma 5.23. *OPT can never schedule two H-jobs consecutively.*

Proof. If two H-jobs J_1 and J_2 run consecutively, the temperature of OPT immediately after running the second job is given by

$$\frac{h_{J_1}}{R^2} + \frac{h_{J_2}}{R} > \frac{R^2/(R+1)}{R^2} + \frac{R^2/(R+1)}{R} = 1$$

i.e. it exceeds the temperature threshold. The inequality is due to the minimum heat contribution of H-jobs. □

Lemma 5.24. *When $\epsilon \geq \frac{R^2+R+1}{(R-1)(R+1)^2}$, $\mathcal{A}$ can always schedule a H-job directly after a C-job if some H-job is pending for $\mathcal{A}$.*

Proof. In order for a C-job J_1 and H-job J_2 to always be able to be scheduled consecutively by $\mathcal{A}$ it must be that $(1+\epsilon+h_{J_1})/R^2 + h_{J_2}/R \leq 1+\epsilon$, and as $h_{J_1} \leq R^2/(R+1)$ and $h_{J_2} \leq R$ this is true if

$$\frac{1+\epsilon+R^2/(R+1)}{R^2} + \frac{R}{R} \leq 1+\epsilon$$

which is equivalent to $\epsilon \geq \frac{R^2+R+1}{(R-1)(R+1)^2}$ thus concluding the proof. □

Lemma 5.25. *When $\epsilon \geq 1/(R^2-1)$, a pending C-job will always be admissible to $\mathcal{A}$.*

Proof. In order for a C-job J to always be admissible for $\mathcal{A}$ it must be that $(1+\epsilon+h_J)/R \leq 1+\epsilon$, and as we have that $h_J \leq R^2/(R+1)$ this is true if

$$\frac{1+\epsilon+R^2/(R+1)}{R} \leq 1+\epsilon$$

which is equivalent to $\epsilon \geq 1/(R^2-1)$. □

We now introduce some notation that will be used in the rest of the proof. We refer to the number of C-jobs scheduled by $\mathcal{A}$ and OPT in $[0,t)$ (i.e. time steps 0, ..., $t-1$) as c_t and c'_t respectively. The number of H-jobs scheduled by $\mathcal{A}$ and OPT in $[0,t)$ will similarly be referred to as h_t and h'_t.

Lemma 5.26. *When $\epsilon \geq 1/(R^2-1)$, if there exists some time t where $\mathcal{A}$ is idle and $|Q_t| = |Q'_t|$, it must be that $h'_t \geq h_t$.*

Proof. Consider such a time t. $\mathcal{A}$ will always schedule an admissible job and by Lemma 5.25 all C-jobs are always admissible. Hence, as $\mathcal{A}$ idles at t, it must have scheduled all of the C-jobs released so far, so $c'_t \leq c_t$. As $|Q_t| = |Q'_t|$ we have that $c_t + h_t = c'_t + h'_t$. Therefore $h'_t \geq h_t$. □

Lemma 5.27. *When $\epsilon \geq \frac{R^2+R+1}{(R-1)(R+1)^2}$, at every time t it must be that $h'_t \leq h_t$.*

Proof. We prove this claim by induction on t. First we show two trivial base cases. Before time 0 neither algorithm will have scheduled any job and so it must be that $h'_0 = h_0 = 0$. If OPT has scheduled a hot job J at time 0, then this job must also be admissible for $\mathcal{A}$ and as $\mathcal{A}$ will always schedule the hottest job possible then either J, or a hotter job will be scheduled by $\mathcal{A}$ and so $h'_1 \leq h_1$.

For general $t \geq 1$, we use the induction hypothesis $h'_{t-1} \leq h_{t-1}$ and $h'_t \leq h_t$ to show that $h'_{t+1} \leq h_{t+1}$. Consider the following cases. If OPT schedules a C-job at t then $h'_{t+1} = h'_t$ and as $h'_t \leq h_t$ it must also be that $h'_{t+1} \leq h_{t+1}$. In the same way we can show $h'_{t+1} \leq h_{t+1}$ if at t both OPT and $\mathcal{A}$ schedule a H-job. This means we only need to consider the case where OPT schedules a H-job at t but $\mathcal{A}$ does not. In this case we know by induction that we have that $h'_{t-1} \leq h_{t-1}$. By Lemma 5.23 we know that OPT cannot schedule a H-job at $t-1$, so $h'_{t+1} = h'_{t-1} + 1$. Hence if $h'_{t-1} \leq h_{t-1} - 1$ then it must be that $h'_{t+1} = h'_{t-1} + 1 \leq h_{t-1} \leq h_{t+1}$.

Otherwise $h'_{t-1} = h_{t-1}$. If $\mathcal{A}$ schedules a H-job at $t-1$ then $h_{t+1} = h_{t-1}+1$ and we know that $h'_{t+1} = h'_{t-1} + 1$, therefore it follows that $h'_{t+1} = h_{t+1}$. Otherwise $\mathcal{A}$ did not schedule a H-job at $t-1$. However we can show that an H-job must be pending at t: as $h'_{t-1} = h_{t-1}$ and OPT schedules a H-

job at t, therefore at least $h_{t-1}+1$ H-jobs have been released up to and including time t, and as $\mathcal{A}$ does not schedule a H-job at $t-1$, at least one H-job is pending at t. $\mathcal{A}$ will always schedule a job when one is admissible though, and we know that $\mathcal{A}$ scheduled either a C-job or no job at $t-1$ so by Lemma 5.24 this H-job or any hotter admissible job will be scheduled by $\mathcal{A}$ at t, contradicting the assumption that it does not. This concludes the proof. □

Theorem 5.28. *When* $\epsilon \geq \frac{R^2+R+1}{(R-1)(R+1)^2}$, *Hottest First is 1-competitive.*

Proof. Note that $\frac{R^2+R+1}{(R-1)(R+1)^2} > \frac{1}{R^2-1}$ whenever $R > 1$. Thus for any $\epsilon > \frac{R^2+R+1}{(R-1)(R+1)^2}$ Lemmas 5.26 and 5.27 both hold.

For OPT to have a total flow time less than $\mathcal{A}$ there must exist some time s such that $|Q_s| > |Q'_s|$. Moreover there must also exist some $t < s$ such that $|Q_t| = |Q'_t|$, $\mathcal{A}$ idles at t and OPT does not idle at t. It follows from Lemmas 5.26 and 5.27 that if such a time t were to exist then $h'_t = h_t$. We now show that such a t cannot exist, specifically by showing that if OPT does not idle at t then $\mathcal{A}$ would not idle either. We consider two cases.

Case 1: OPT schedules a C-job at t. As $h'_t = h_t$ and by definition of time t we have $|Q_t| = |Q'_t|$, then $c'_t = c_t$ (by the same argument as Lemma 5.26). This means that if a C-job is pending for OPT at t, that one must also be pending for $\mathcal{A}$ at t. By Lemma 5.25 we know that this C-job must be admissible for $\mathcal{A}$ and $\mathcal{A}$ always schedules a job if one is admissible, contradicting that $\mathcal{A}$ is idle at t.

Case 2: Otherwise OPT schedules a H-job at t. This means $h'_{t+1} = h_{t+1}+1$ contradicting Lemma 5.27.

□

We have proven that when $\epsilon \geq \frac{R^2+R+1}{(R-1)(R+1)^2}$ Hottest First is 1-competitive. When $R = 2$ then this makes Hottest First 1-competitive when $\epsilon \geq \frac{7}{9}$, while from Corollary 5.20 we know that no algorithm can be 1-competitive when $\epsilon < \frac{1}{4}$.

By Theorem 2.9 we know that in the unit length job case with the objective of maximising throughput Hottest First performs worse than Coolest First. It is therefore interesting to note that with the objective of minimising total flow time with resource augmentation, Hottest First performs the best out of the two algorithms.

5.7 Maximum Flow Time

We now show some bounds for algorithms that have the objective of minimising the maximum flow time. This objective can be described using standard 3-field notation as $1|\textit{online-}r_i, h_i, p_i = 1|F_{\max}$

First we note that the k-competitive lower bounds given for the objective of minimising total flow time can be easily modified to give $\Omega(k)$-competitive lower bounds with regard to the maximum flow time, in both the bounded heat and increased threshold models.

It is trivial to show that the algorithms Hottest First and Coolest First have competitive ratios of at least $\Omega(n)$. This is for both the case where the thermal threshold of the online algorithm is increased and when job heats are bounded, as the proofs do not even require a job being not admissible for the online algorithm. The proofs take advantage of how the algorithms choose

which job to schedule, and use this to prevent another job ever being chosen. For Coolest First if we release two jobs B and K in the first time step with B being an B-job and $h_K > h_B$. Coolest First will schedule B as its cooler and then in each time step $i > 1$ we release another B-job. These will always be cooler than K and so Coolest First will schedule an B-job at each time step and K will never get scheduled. Meanwhile OPT will schedule K followed by B and then at each of the next i time steps the B-job that was released at $i-1$, giving OPT a maximum flow time of 2. To show that Hottest First has a competitive ratio of at least $\Omega(n)$ we release the same jobs in the same way except that $h_K < h_B$.

We now consider the algorithm FIFO. We define FIFO to be an algorithm that schedules a job whenever one is admissible, and chooses the job with the earliest release time if there are several admissible jobs. If several jobs have the same release time then any of the jobs is chosen. This makes FIFO a non-idling algorithm.

In the case without temperature FIFO is known to be optimal for the objective of minimising the maximum flow time. FIFO is a non-idling algorithm meaning that we can use Theorem 5.17 to show FIFO must have a competitive ratio of at least $\Omega(n)$ for the bounded maximum job heat model for any $\delta > 0$. Even though Theorem 5.17 is for the objective of minimising total flow time, the proof works by increasing the size of the queue of the online algorithm to a large amount while keeping the queue size of the optimal algorithm constant. The proof still holds because it is clear that each time the queue size of the online algorithm increases by one, then the maximum flow time out of the jobs in that queue also increases by one.

We can also show some results for FIFO in the increased thermal threshold case. The result given in Theorem 5.21 still holds for the maximum flow time case for the same reason as in the bounded heat case, however we now give a lower bound for FIFO that holds for all non-trivial values of ϵ.

Theorem 5.29. *If the temperature threshold of the online algorithm is* $1+\epsilon$ *where* $\epsilon < 1/(R-1)$ *then FIFO has a competitive ratio of at least* $\Omega(n)$.

Proof. This proof starts in a similar way to that of Theorem 5.22, however we don't use repeated phases and sub-phases. We fix some large enough w,x and y to be described later. At time 0 we release a B-block of size x and at time x we release a Z-block of size w, with the modification that the last two jobs of the Z-block are released at the same time step $x+w-2$. At $x+w-1$ we release y B-jobs (note that this is not a B-block because all of the jobs are released at the same time step), and at $x+w$ we release a job K with $h_K = R - 1/R^w$. At $x+w+y$ we release a very large B-block of size c.

FIFO will schedule the first B-block, followed by all the jobs in the Z-block, then schedule the final B-block, as all of these jobs are released before K. This means that the temperature of FIFO at $x+w+y$ will be at least $1-1/R^{y+1}$. We can therefore always choose values of w and y such that K is not admissible at this time for any $\epsilon < 1/(R-1)$, using techniques similar to those in Theorem 5.22. FIFO will therefore schedule the final B-block, and therefore not cool down enough during these time steps to schedule K. The flow time of K will therefore be at least c as it can't be scheduled until the B-block has completed.

Meanwhile OPT will schedule all of the jobs from the first B-block fol-

lowed by those from the Z-block. OPT will then schedule K at time $x + w$ followed by the remaining two B-blocks in FIFO order . This means that the maximum flow time for a job in OPT will be $y + 2$. As c can be made much larger than the values of w,x and y this gives a competitive ratio of $\Omega(n)$ for FIFO. □

We now examine an algorithm which can be seen as another interpretation of FIFO, and show that it has a competitive ratio of at least $\Omega(n)$. We will call the algorithm FIFO_withIdle. This algorithm is defined as starting the job with the earliest release time (whether admissible or not) at each time step, if there are several such jobs then pick one that is admissible (if any), and if none of these jobs are admissible then idle. We now show that when the maximum heat of a job is bounded to $R - 1 + \delta$ for any $\delta > 0$ then FIFO_withIdle has a competitive ratio of at least $\Omega(n)$. Note that the way these proofs are constructed means they will also hold for the case of total/average flow time.

Theorem 5.30. *When the maximum job heat is $R - 1 + \delta$ for any $\delta > 0$, FIFO_withIdle has a competitive ratio of at least $\Omega(n)$.*

Proof. This proof involves the release of a large number of phases. We first define a phase. Consider a phase starting at time t. At t we release a B-block of size x, for some large enough x. This value depends on the values of R and δ. At time $t + x$ we then release an B-job B and a Z-job Z. Then at time $t + x + 1$ we release a B-block of size $x - 1$. Then at time $t + 2x + 1$ we release a job K with $h_K = \min\{R - 1 + \delta, R - 1/R\}$. This completes the description of a phase.

We release an arbitrarily large number of phases, each immediately following the preceding one. For each phase OPT will schedule all $2x$ of the B-jobs (including B) then follow them with Z and then be cool enough to schedule K as soon as it is released. FIFO_withIdle will always schedule the jobs in the order they are released meaning that for the jobs from each phase it will schedule the first B-block, followed by B and Z (in either order) then the final B-block. This means the temperature of FIFO_withIdle will be at least $1 - 1/R^x$ by the time K is the next job to be scheduled. An x value can always be chosen that is large enough so that FIFO_withIdle is too hot to schedule K, and therefore will idle.

As the schedule of FIFO_withIdle introduces an idle step with every phase, the execution of each job in the next phase is postponed by one step for the algorithm. As the algorithm schedules jobs in a FIFO order, it must be that for each phase the jobs are still scheduled in the same order as they were in the first phase. This means that the maximum flow time of a job after c phases in OPT is x while in FIFO_withIdle it is $\Theta(c)$. As each phase involves a constant number of jobs being released this means that the maximum flow time of any job in FIFO_withIdle is $\Omega(n)$, while in OPT it remains constant, concluding the proof. □

We can also show that FIFO_withIdle has a competitive ratio of at least $\Omega(n)$ for the increased temperature threshold model, for all $\epsilon < 1/(R-1)$. This proof is similar to the proof in Theorem 5.22. Note that again this proof also shows that FIFO_withIdle has a competitive ratio of at least $\Omega(n)$ for the objective of minimising total flow time.

Theorem 5.31. *If the temperature threshold of the online algorithm is* $1+\epsilon$ *where* $\epsilon < 1/(R-1)$ *then FIFO_withIdle has a competitive ratio of at least* $\Omega(n)$.

Proof. This proof involves the release of a large number of phases. We first define a phase. Consider a phase starting at time t. At t we release a B-block of size x, for some large enough x. These jobs will all be scheduled by FIFO_withIdle and OPT as soon as they are released. At $t+x$ we then release w Z-jobs (note that this is not a Z-block as all the jobs are released at the same time step) , which are also scheduled by both OPT and FIFO_withIdle bringing both system temperatures to arbitrarily close to $1/R^w$ at time $t+x+w$. At time step $t+x+w-1$, the time step before all the final Z-job is scheduled, we release y B-jobs. At the next time step $t+x+w$ we release a job K_1 with a heat contribution of $R-1/R^w$.

OPT will schedule the job K_1, followed by the B-jobs while FIFO_withIdle will schedule all y of the B-jobs as they were released first. We can always choose w and y values that are large enough so that FIFO_withIdle be too hot to schedule K_1 at $t+x+w+y$ for any $\epsilon < 1/(R-1)$, and so FIFO_withIdle will idle. This completes the description of a phase.

We release an arbitrarily large number of phases, each immediately following the preceding one. As the schedule of FIFO_withIdle introduces an idle step with every phase, the execution of each job in the next phase is postponed by one for the algorithm. This means that the maximum flow time of a job after c phases in OPT is $\max\{w, y\}$ while in FIFO_withIdle it is $\Theta(c)$. As each phase involves a constant number of jobs being released this means that the maximum flow time of any job in FIFO_withIdle is $\Omega(n)$,

while in OPT it remains constant, concluding the proof. □

These results seem to indicate that when considering the online model with temperature, the objective of minimising maximum flow time is at least as hard as minimising total flow time. We also know that the traditional algorithm for minimising maximum flow time, FIFO, does not perform well in the setting with temperature.

5.8 Summary

In this chapter we consider minimising the flow time of unit length jobs. We consider the offline case initially and show that when maximum job heats are bounded to any non-trivial heat, the problem of creating a schedule that is optimal for minimising total flow time is NP-hard. We also show that no polynomial time algorithm with an approximation ratio of less than $O(n^{1/2-\epsilon})$ can exist for any $\epsilon > 0$, unless $P = NP$ when job heats are unbounded. For the special case where all jobs are released at the same time step we show a polynomial time algorithm that has a 2-approximation ratio.

Next we consider the online case where maximum job heats are bounded and show a lower bound that increases as the maximum job heat increases. We also show that non-idling algorithms must have a competitive ratio of at least $\Omega(n)$, and we show an upper bound on non-idling algorithm that matches this for any fixed maximum job heat.

We also give results for the case with resource augmentation in the form of an increased temperature threshold and unbounded maximum job heat. We show a lower bound that increases as the amount of resource augmentation

decreases. We show counter examples for the competitiveness of non-idling algorithms and the algorithm Coolest First. We then show that Hottest First is 1-competitive when enough extra resources are available to the online algorithm. This contrasts with the case of maximising throughput where Coolest First (and all reasonable algorithms) perform better than Hottest First.

Finally we consider the objective of minimising the maximum flow time. We show that the lower bounds from the minimising maximum flow time objective still hold for this objective. We also show that several natural algorithms have competitive ratios of at least $\Omega(n)$.

Chapter 6

Conclusions

6.1 Summary of Results and Remarks

In this thesis we studied online algorithms for a variety of temperature aware models that require the temperature of the algorithms to remain below a given thermal threshold at all times, with two main scheduling objectives. The first objective was to maximise the throughput of a schedule, with the second objective being to minimise the flow time of a schedule. We give new lower bounds and upper bounds for these problems.

For maximising the throughput of unit length jobs we considered two sub-problems. For the unweighted case we extended the analysis from [19] to give an upper bound for reasonable algorithms with multiple machines and any $R > 1$. These algorithms have an upper bound of 2 whenever $R \geq 2$, and this increases towards infinity as R approaches 1. We then show matching lower bounds for the single machine case and give some lower bounds for the multiple machine case that are lower. We also show that for this objective

Hottest First performs worse than any reasonable algorithm.

For the weighted problem we show several bounds. For the single machine case with full heat we show an upper bound on a randomised algorithm that is almost optimal for the fully online case when W is not known in advance. For the multiple machine full heat case we extend the randomised single machine results from [11] that show the same upper and lower bounds as the single machine case hold. This means that the randomised algorithm analysed is optimal within a constant factor in the semi-online case, and almost optimal in the fully online case. For the deterministic full heat case on multiple machines we give a lower bound that matches the upper bound given in [11] for this model with a fixed number of machines m, within a constant factor. Finally we extend the single machine bounded heat results from [11] to show the same bounds still hold in the multiple machine case, making the algorithm optimal within constant factors.

It is interesting to note that in the non-weighted job model allowing multiple machines seems to give a significant advantage over the single machine case. However in the weighted job model the results for multiple machines are the same (within constant factors) as the single machine case, although the weighted results are for the randomised case while the non-weighted job results are deterministic.

Next we extend the model to consider jobs that are still equal length, but larger than unit length, and still with the objective of maximising throughput. We consider the case with single machines and unweighted jobs. We give an upper bound on the non-preemptive algorithm Coolest First for all $R > 1$, that increases when R decreases, and also as the job length p decreases. We

also show an upper bound on all non-idling algorithms that is exactly one higher for all R and p values. We then show a lower bound for the non-preemptive case that shows that Coolest First is optimal. We also give lower bounds for the cases with both preemptive resumes and preemptive restarts. We show that for most values of R and p allowing preemptive restarts don't allow any reduction of the competitive ratio over the non-preemptive case, and for the other R and p values the potential reduction is only small. It is also shown that the lower bound for preemptive resumes is exactly one less than for preemptive restarts, except for a very small R and p range where they have the same lower bounds.

This model is then further extended to consider jobs with variable lengths, again with the objective of maximising throughput. We considered two sub-problems for this model. The first sub-problem is the case where jobs are unweighted. We show an upper bound on non-idling algorithms that is linear in L. We then show almost matching lower bounds for the case without preemption, and with preemption we show a bound that is also optimal within constant factors when R is fixed. The next sub-problem is for the case where jobs have weights that are proportional to their length. We show an upper bound on the algorithm Longest First that is also linear in L. We then show a lower bound that works for the non-preemptive and both preemptive models, that matches for all small enough values of R, and is almost matching all other R values.

The second objective we study is that of minimising the flow time of a schedule. For this model we study the single machine case where jobs are all unit length. First we give some results for the offline case. We show

that even when the maximum heat of a job is bounded to $R - 1 + \delta$, the problem of minimising the total flow time of a schedule is NP-hard. We then show that for all $R > 1$, but with unbounded job heat, for the problem of minimising the total flow time no polynomial time approximation algorithm with an approximation ratio of less than $O(n^{1/2-\epsilon})$ can exist for any $\epsilon > 0$, unless $P = NP$. Finally we give a positive result for the restricted case when all jobs are released at the same time, which is a polynomial time algorithm with an approximation ratio of 2.

Next we give results for the online case where the maximum heat of a job is bounded to $R - \epsilon$. We show that no deterministic algorithm can be better than k-competitive where $\epsilon \leq (R-1)/R^k$, and for large ϵ we show that no deterministic algorithm will be better than 2-competitive. We then show that no non-idling algorithm can get a competitive ratio less than $\Omega(n)$ for any allowed value of ϵ. We then give a trivial upper bound on all non-idling algorithms of $O(kn)$ where $\epsilon \leq (R-1)/R^k$.

Results are then given for a model with resource augmentation. In this model job heats are no longer bounded, but the online algorithm is given an increased temperature threshold of $1 + \epsilon$ for some $0 < \epsilon < 1/(R-1)$. We show than no deterministic algorithm can be better than k-competitive where $\epsilon < (R-1)/R^{k+1}$. Next we give lower bounds for some of bad algorithms. We show that no non-idling algorithm can get a competitive ratio of less than $\Omega(n)$ when $\epsilon < 1/R^2$. We then show that Coolest First cannot achieve a competitive ratio of less than $\Omega(n)$ for all non-trivial values of ϵ. Finally for this section we show that Hottest First is 1-competitive when $\epsilon \geq \frac{R^2+R+1}{(R-1)(R+1)^2}$.

It is interesting to use this result to compare the algorithms that perform

well for the different scheduling objectives. We show in Chapters 2 and 3 for the objective of maximising the throughput of jobs that Coolest First is the best algorithm, with Theorem 2.9 showing that Hottest First performs worse than Coolest First for this objective. However in Chapter 5 when we consider the different objective of minimising flow time we show with resource augmentation Hottest First performs significantly better than Coolest First.

Finally, we present a discussion of algorithms for the problem of minimising the maximum flow time in a schedule. We show that all of the lower bound results for the minimising total flow time case hold for the maximum flow time objective. We also show that Hottest First and some other natural algorithms have competitive ratios of at least $\Omega(n)$, even in the case with an increased temperature threshold.

It is interesting to compare the difficulty of the two different flow time objectives. It seems from the results given here that the problem of minimising the maximum flow time of jobs in a schedule is at least as hard as minimising the total flow time. However the results in this work are not conclusive so investigating the difficulty of the two problems would be some interesting future work.

6.2 Open Problems and Future Work

We will now discuss some open problems in the models studied in this thesis. First we will consider the objective of maximising throughput. For unweighted unit-length jobs on a single machine, the majority of the work that has been done so far is for the deterministic case. In the randomised

case it is only known that it is possible to prove a lower bound of 4/3 for the case when $R \geq 2$ [19] but so far no improvement on the upper bound of 2 has been found.

For the multiple machine version of this problem there is a significant gap between the upper and lower bounds. In particular it seems as though the upper bound should be lower than for the single machine case. As shown by Theorem 2.14, in order to reduce the upper bound on the competitive ratio, it would be necessary to use an algorithm that sometimes leaves a machine idle, even if there are still some admissible jobs available.

For the equal length jobs problem it would be interesting to try to close the (small) gap between the non-preemptive upper bound and the preemptive resumes lower bound. This is likely to involve designing an algorithm that uses preemptive resumes to reduce the upper bound for the problem.

For the unweighted variable length jobs problem it has been shown that allowing preemption can only give up to a constant factor improvement in the upper bound for any fixed R. However when R is very small this constant can be large. Therefore it would be interesting to try and close this gap. It would also be interesting to investigate the difference in power between the preemptive resumes and preemptive restarts model. From the equal length job results it seems likely that allowing preemptive resumes should be more powerful than allowing preemptive restarts but so far no work has been undertaken for the variable length jobs model that gives different bounds for the two preemption models.

For the proportionally weighted variable length jobs problem it might also be interesting to consider the case where the value of a job increases in

a sub-linear way with the length of the job i.e. $w_J = p_J^\alpha$ for some $0 < \alpha < 1$. For $\alpha = 1$, the algorithm Longest First performs well, however it is unlikely to perform as well when the length of a job doesn't increase the value of the job as much.

For both the equal and variable length job problems it would be interesting to try and extend the results to multiple machines. In particular it seems from the unit length results that to reduce the single machine upper bound it would again be necessary to use algorithms that sometimes leave machines idle, even if there are still admissible jobs available.

It might also be interesting to consider the model where instead of idling when the threshold is reached, one could consider reducing the frequency of the processor. There could be discrete multiple frequencies, or it could be a continuous range of frequencies. This model is likely to involve combining the techniques from the unit length and variable length job problems.

There are also several open problems for the objective of minimising the flow time. For the offline problem it would be of interest to try and design polynomial time algorithms with better than $O(n)$-approximation ratios. It would also be interesting to investigate the inapproximability of the bounded job heat model. The given inapproximability result requires jobs of very large temperatures and so does not still hold in the bounded job heat model, while the NP-hardness result does still hold.

The online case with bounded job heats also has several open problems. As the inapproximability result from the offline case implies that any online algorithm with a competitive ratio better than $O(\sqrt{n})$ will not be able to run in polynomial time, it might be interesting to consider some online algorithms

with larger than polynomial running time to discover if this bound can be broken. However analysing these algorithms is unlikely to be easy. Therefore it could be more achievable and still interesting to prove an upper bound on a polynomial time online algorithm.

An interesting open problem in the increased thermal threshold model is closing the gap between the 1-competitive upper bound on the Hottest First algorithm for large enough ϵ, and the $\Omega(n)$ lower bound for $\epsilon < 1/R^2$ which is proven for all non-idling algorithms. It would be especially interesting to see if an upper bound can be found that gradually increases towards $\Omega(n)$, or if there is an ϵ value where Hottest First suddenly stops being 1-competitive and immediately becomes $O(n)$-competitive. It might also be interesting to analyse other algorithms to try and close this gap. From Corollary 5.20 we know that for small ϵ no algorithm can be 1-competitive, and so there is a large gap between the ϵ values necessary for the known 1-competitive algorithm and the ϵ values where no algorithm can be 1-competitive.

It might also be interesting to try different types of resources augmentation for this model i.e. increasing the cooling factor of the online algorithm relative to the adversary. This would be equivalent to increasing the cooling of a processor, for example by fitting a larger fan.

Bibliography

[1] Susanne Albers. Energy-efficient algorithms. *Communications of the ACM*, 53(5):86–96, May 2010.

[2] Nir Andelman, Yishay Mansour, and An Zhu. Competitive queueing policies for qos switches. In *Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms (SODA)*, pages 761–770, 2003.

[3] Baruch Awerbuch, Yair Bartal, Amos Fiat, and Adi Rosén. Competitive non-preemptive call control. In *Proceedings of the fifth annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 312–320, 1994.

[4] Hakan Aydin, Pedro Mejía-Alvarez, Daniel Mossé, and Rami G. Melhem. Dynamic and aggressive scheduling techniques for power-aware real-time systems. In *Proceedings of the 22nd IEEE Real-Time Systems Symposium (RTSS)*, pages 95–105, 2001.

[5] Michael O. Ball and Maurice Queyranne. Toward robust revenue management: Competitive analysis of online booking. *Operations Research*, 57:950–963, July 2009.

[6] Evripidis Bampis, Dimitrios Letsios, Giorgio Lucarelli, Evangelos Markakis, and Ioannis Milis. On multiprocessor temperature-aware scheduling problems. In *Proceedings of Frontiers in Algorithmics and Algorithmic Aspects in Information and Management (FAW-AAIM)*, pages 149–160, 2012.

[7] Nikhil Bansal, Tracy Kimbrel, and Kirk Pruhs. Speed scaling to manage energy and temperature. *Journal of the ACM*, 54(1), 2007.

[8] Sanjoy K. Baruah, Jayant R. Haritsa, and Nitin Sharma. On-line scheduling to maximize task completions. In *Proceedings of the 15th IEEE Real-Time Systems Symposium (RTSS)*, pages 228–236. IEEE Computer Society, 1994.

[9] Sanjoy K. Baruah, Gilad Koren, D. Mao, Bhubaneswar Mishra, Arvind Raghunathan, Louis E. Rosier, Dennis Shasha, and Fuxing Wang. On the competitiveness of on-line real-time task scheduling. *Real-Time Systems*, 4(2):125–144, 1992.

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

[11] Martin Birks, Daniel Cole, Stanley P. Y. Fung, and Huichao Xue. On-line algorithms for maximizing weighted throughput of unit jobs with temperature constraints. *Journal of Combinatorial Optimization*, (to appear).

[12] Martin Birks and Stanley P. Y. Fung. Temperature aware online scheduling with a low cooling factor. In *Proceedings of 7th annual Conference on Theory and Applications of Models of Computation, (TAMC)*, pages 105–116, 2010 (Journal version submitted).

[13] Martin Birks and Stanley P. Y. Fung. Temperature aware online algorithms for scheduling equal length jobs. *Theoretical Computer Science*, 2012 (to appear).

[14] Allan Borodin and Ran El-Yaniv. *Online computation and competitive analysis*. Cambridge University Press, 1998.

[15] Thidapat Chantem, Xiaobo Sharon Hu, and Robert P. Dick. Temperature-aware scheduling and assignment for hard real-time applications on mpsocs. *IEEE Transactions on VLSI Systems*, 19(10):1884–1897, 2011.

[16] Bo Chen, Chris N. Potts, and Gerhard J. Woeginger. A review of machine scheduling: Complexity, algorithms and approximability. *Handbook of Combinatorial Optimization*, 3:21–169, 1998.

[17] Francis Y. L. Chin, Marek Chrobak, Stanley P. Y. Fung, Wojciech Jawor, Jiri Sgall, and Tomás Tichý. Online competitive algorithms for maximizing weighted throughput of unit jobs. *Journal of Discrete Algorithms*, 4(2):255–276, 2006.

[18] Francis Y. L. Chin and Stanley P. Y. Fung. Online scheduling with partial job values: Does timesharing or randomization help? *Algorithmica*, 37:149–164, 2003.

[19] Marek Chrobak, Christoph Dürr, Mathilde Hurand, and Julien Robert. Algorithms for temperature-aware task scheduling in microprocessor systems. *Sustainable Computing: Informatics and Systems*, 1(3):241 – 247, 2011.

[20] Marek Chrobak, Wojciech Jawor, Jiri Sgall, and Tomás Tichý. Online scheduling of equal-length jobs: Randomization and restarts help. *SIAM Journal on Computing*, 36(6):1709–1728, 2007.

[21] Ayse K. Coskun, Tajana S. Rosing, and Keith Whisnant. Temperature aware task scheduling in MPSoCs. In *Proceedings of Design, Automation and Test in Europe Conference*, pages 1659–1664, 2007.

[22] Jihuan Ding, Tomáš Ebenlendr, Jiří Sgall, and Guochuan Zhang. Online scheduling of equal-length jobs on parallel machines. In *Proceedings of the 15th annual European conference on Algorithms*, ESA, pages 427–438, Berlin, Heidelberg, 2007. Springer-Verlag.

[23] Jihuan Ding and Guochuan Zhang. Online scheduling with hard deadlines on parallel machines. In *Proceedings of Second International Conference Algorithmic Aspects in Information and Management (AAIM)*, volume 4041, pages 32–42. Springer, 2006.

[24] Shlomi Dolev and Alexander Keizelman. Non-preemptive real-time scheduling of multimedia tasks. *Real-Time Systems*, 17(1):23–39, 1999.

[25] Matthias Englert and Matthias Westermann. Considering suppressed packets improves buffer management in qos switches. In *Proceedings of*

the eighteenth annual ACM-SIAM symposium on Discrete algorithms, SODA, pages 209–218, 2007.

[26] Stanley P. Y. Fung, Chung Keung Poon, and Duncan K. W. Yung. On-line scheduling of equal-length intervals on parallel machines. *Information Processing Letters*, 112(10):376–379, 2012.

[27] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., 1990.

[28] Pawel Gepner and Michal Filip Kowalik. Multi-core processors: New way to achieve high system performance. In *Fifth International Conference on Parallel Computing in Electrical Engineering (PARELEC)*, pages 9–13. IEEE Computer Society, 2006.

[29] Sally A. Goldman, Jyoti Parwatikar, and Subhash Suri. Online scheduling with hard deadlines. *Journal of Algorithms*, 34:370–389, February 2000.

[30] Michael H. Goldwasser and Mark Pedigo. Online nonpreemptive scheduling of equal-length jobs on two identical machines. *ACM Transactions on Algorithms*, 5(1):2:1–2:18, December 2008.

[31] R.L. Graham, E.L. Lawler, J.K. Lenstra, and A.H.G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Operations Research*, 5:287–326, 1979.

[32] Han Hoogeveen, Chris N. Potts, and Gerhard J. Woeginger. On-line scheduling on a single machine: maximizing the number of early jobs. *Operations Research Letters*, 27(5):193 – 197, 2000.

[33] Sandy Irani and Kirk Pruhs. Algorithmic problems in power management. *SIGACT News*, 36(2):63–76, 2005.

[34] Ravindra Jejurikar and Rajesh Gupta. Procrastination scheduling in fixed priority real-time systems. In *Proceedings of the ACM SIGPLAN/SIGBED conference on Languages, compilers, and tools for embedded systems*, LCTES, pages 57–66, 2004.

[35] Bala Kalyanasundaram and Kirk Pruhs. Speed is as powerful as clairvoyance. *Journal of the ACM*, 47(4):617–643, July 2000.

[36] Bala Kalyanasundaram and Kirk R. Pruhs. Maximizing job completions online. *Journal of Algorithms*, 49(1):63–85, October 2003.

[37] Hans Kellerer, Thomas Tautenhahn, and Gerhard J. Woeginger. Approximability and nonapproximability results for minimizing total flow time on a single machine. *SIAM Journal on Computing*, 28(4):1155–1166, 1999.

[38] Sven O. Krumke, Alfred Taudes, and Stephan Westphal. Online scheduling of weighted equal-length jobs with hard deadlines on parallel machines. *Computers and Operations Research*, 38:1103–1108, August 2011.

[39] Eren Kursun, Chen yong Cher, Alper Buyuktosunoglu, and Pradip Bose. Investigating the effects of task scheduling on thermal behavior. In *Third Workshop on Temperature-Aware Computer Systems (TACS)*, 2006.

[40] Woo-Cheol Kwon and Taewhan Kim. Optimal voltage allocation techniques for dynamically variable voltage processors. *ACM Transactions on Embedded Computing Systems*, 4:211–230, 2005.

[41] E. Lawler. A dynamic programming algorithm for preemptive scheduling of a single machine to minimize the number of late jobs. *Annals of Operations Research*, 26:125–133, 1990. 10.1007/BF02248588.

[42] Wonbok Lee, Kimish Patel, and Massoud Pedram. Dynamic thermal management for mpeg-2 decoding. In *Proceedings of the international symposium on Low power electronics and design*, ISLPED, pages 316–321, 2006.

[43] Cynthia A. Phillips, Clifford Stein, Eric Torng, and Joel Wein. Optimal time-critical scheduling via resource augmentation. *Algorithmica*, 32(2):163–200, 2002.

[44] Kirk Pruhs, Jiri Sgall, and Eric Torng. Online scheduling. In *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, chapter 15, pages 15–1 – 15–41. CRC Press, 2004.

[45] Kevin Skadron, Mircea R. Stan, Wei Huang, Sivakumar Velusamy, Karthik Sankaranarayanan, and David Tarjan. Temperature-aware microarchitecture. *SIGARCH Computer Architecture News*, 31:2–13, May 2003.

[46] Kevin Skadron, Mircea R. Stan, Karthik Sankaranarayanan, Wei Huang, Sivakumar Velusamy, and David Tarjan. Temperature-aware microarchitecture: Modeling and implementation. *ACM Transactions on Architecture and Code Optimization*, 1:94–125, March 2004.

[47] Daniel Dominic Sleator and Robert Endre Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.

[48] Nguyen Kim Thang. Improved online scheduling in maximizing throughput of equal length jobs. In *The 6th International Computer Science Symposium in Russia (CSR)*, pages 429–44, 2011.

[49] Nodari Vakhania. A fast on-line algorithm for the preemptive scheduling of equal-length jobs on a single processor. In *Proceedings of the 2nd WSEAS International Conference on Computer Engineering and Applications*, pages 158–161, 2008.

[50] Gerhard J. Woeginger. On-line scheduling of jobs with fixed start and end times. *Theoretical Computer Science*, 130(1):5–16, 1994.

[51] Jun Yang, Xiuyi Zhou, Marek Chrobak, Youtao Zhang, and Lingling Jin. Dynamic thermal management through task scheduling. In *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 191–201, 2008.

[52] Andrew Chi-Chin Yao. Probabilistic computations: Toward a unified measure of complexity. In *Proceedings of the 18th annual Symposium*

on Foundations of Computer Science, pages 222–227. IEEE Computer Society, 1977.

[53] Frances Yao, Alan J. Demers, and Scott Shenker. A scheduling model for reduced CPU energy. In *36th annual Foundations of Computer Science Symposium (FOCS)*, pages 374–382. IEEE Computer Society, 1995.