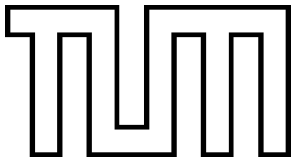


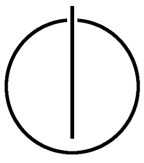
DEPARTMENT OF INFORMATICS

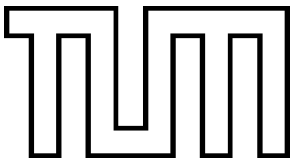
TECHNICAL UNIVERSITY OF MUNICH

Bachelor's Thesis in Informatics

Algorithms for Dynamic Right-Sizing in Data Centers

Kevin Kappelmann





DEPARTMENT OF INFORMATICS

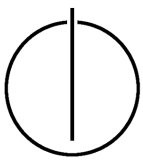
TECHNICAL UNIVERSITY OF MUNICH

Bachelor's Thesis in Informatics

Algorithms for Dynamic Right-Sizing in Data Centers

Algorithmen für dynamische Kapazitätsanpassung in Datenzentren

Author:	Kevin Kappelmann
Supervisor:	Prof. Dr. Susanne Albers
Advisor:	Jens Quedenfeld
Submission Date:	Submission date



I confirm that this bachelor's thesis in informatics is my own work and I have documented all sources and material used.

Munich, Submission date Kevin Kappelmann

Acknowledgments

Abstract

Contents

1 Introduction

TODO: Hardware prices vs. energy costs in data centers, related work and purpose of this paper (offline algorithm, approximation algorithm,...). TODO: definition of optimal and x-approximation algorithm

1.1 Model Description

In order to address the described ever-growing energy consumption, we want to examine a scheduling problem that commonly arises in data centers. More specifically, we consider a model consisting of a fixed number of homogeneous servers denoted by $m \in \mathbb{N}$ and a fixed number of time slots denoted by $T \in \mathbb{N}$. Each server possesses two power states, that is a machine is either powered on (*active state*) or powered off (*sleep state*).

For any time slot $t \in [T]$, we have a *mean arrival rate* denoted by λ_t, describing the amount of expected load to process in time slot t. The arrival rates are expected to be normalized such that each server can handle a load between 0 and 1 in any time slot. We denote the assigned load for server i in time slot t by $\lambda_{i,t} \in [0, 1]$. Consequently, for any time slot t, we expect an arrival rate between 0 and m, that is $\lambda_t \in [0, m]$; otherwise, the servers would not be able to process the given load in time.

The costs incurred by a single machine are described by the sum of the machine's *operating costs*, specified by $f : [0, 1] \to \mathbb{R}$, as well as its *(power state) switching costs*, specified by $\beta \in \mathbb{R}_{\geq 0}$. The operating costs function f may not exclusively account for energy costs. For example, f may also allow for costs incurred by delays, such as lost revenue caused by users waiting for their responses. Similarly, β may also allow for delay costs, wear and tear costs or the like.[2]

We assume that a sleeping server does not cause any costs. Note that $f(0)$ denotes the costs incurred by an idle server, not a sleeping one; in particular, $f(0)$ may be non-zero. Further, we expect f to be a convex function. This may seem like a notable restriction at first, but it indeed captures the behavior of most modern server models. Since we are dealing with homogeneous servers, f and β are the same for all machines.

For convenience, we assume that all machines sleep at time $t = 0$ and also force all machines to sleep after the scheduling process, that is at times $t > T$. Consequently,

any server must power down exactly as many times as it powers on. This allows us to consolidate a machine's power-up and power-down costs into β and to model both costs as being incurred when powering up a server; that is, a model with power-up costs $\beta_\uparrow$ and power-down costs $\beta_\downarrow$ can be simply transferred to our model by setting $\beta \coloneqq \beta'_\uparrow \coloneqq \beta_\uparrow + \beta_\downarrow$ and $\beta'_\downarrow \coloneqq 0$. As a consequence, we expect that there are no loads at times $t \notin [T]$, that is $\lambda_t = \lambda_{i,t} = 0$ for $t \notin [T]$.

1.2 Problem Statement

Using the above definitions, we can define the input of our model by setting $\mathcal{I} \coloneqq (m, T, \Lambda, \beta, f)$ where $\Lambda = (\lambda_1, \dots, \lambda_T)$ is the sequence of arrival rates. We will subsequently identify a problem instance by its input $\mathcal{I}$. Naturally, given a problem instance $\mathcal{I}$, we want to schedule our servers in such a way that we minimize the sum of incurred costs while ensuring that we process the given loads in time. For this, consider for each server $i \in [m]$ the sequence of its states S_i and the sequence of its assigned loads L_i:

$$S_i \coloneqq (s_{i,1}, \dots, s_{i,T}) \in \{0,1\}^T$$
$$L_i \coloneqq (\lambda_{i,1}, \dots, \lambda_{i,T}) \in [0,1]^T$$

where $s_{i,t} \in \{0,1\}$ denotes whether server i at time t is sleeping (0) or active (1). Recall that we assume all machines are sleeping at times $t \notin [T]$; thus, for $t \notin [T]$ and $i \in [m]$, we have $s_{i,t} = 0$. We can now define the sequence of all state changes and the sequence of all assigned loads:

$$\mathcal{S} \coloneqq (S_1, \dots, S_m)$$
$$\mathcal{L} \coloneqq (L_1, \dots, L_m)$$

We will subsequently call a pair $\Sigma \coloneqq (\mathcal{S}, \mathcal{L})$ a *schedule*. Finally, we are ready to define our problem statement. Given an input $\mathcal{I}$, our goal is to find a schedule Σ that satisfies the following optimization:

$$\text{minimize} \qquad c(\Sigma) \coloneqq \underbrace{\sum_{t=1}^{T}\sum_{i=1}^{m}\big(f(\lambda_{i,t})s_{i,t}\big)}_{\text{operating costs}} + \underbrace{\beta\sum_{t=1}^{T}\sum_{i=1}^{m}\min\{0, s_{i,t} - s_{i,t-1}\}}_{\text{switching costs}} \tag{1.1}$$

$$\text{subject to} \qquad \sum_{i=1}^{m}(\lambda_{i,t}s_{i,t}) = \lambda_t, \quad \forall t \in [T] \tag{1.2}$$

We call a schedule *feasible* if it satisfies (1.2), and *optimal* if it satisfies (1.1) and (1.2).

2 Preliminaries

In this chapter, we conduct the preparatory work that will lay the foundations for our algorithms. For this, we will analyze the structure of feasible schedules to find characteristics of optimal strategies. These characteristics will then allow us to greatly simplify our optimization conditions.

We begin by examining the state sequences of feasible schedules. As we consider homogeneous servers, we do not care which exact machines process the given work loads. Rather we only care about the number of active servers and the distribution of loads between them. It is in particular unreasonable to power down a machine and to power on a different one in return; we could just keep the first server powered on, thereby saving switching costs. This investigation is captured by our first proposition.

Proposition 2.1 (Reasonable switching)**.** *Given a problem instance $\mathcal{I}$ and a feasible schedule Σ, there exists a feasible schedule Σ' such that*

(i) $c(\Sigma') \leq c(\Sigma)$ *and*

(ii) Σ' *never powers on and shuts down servers at the same time slot, that is* Σ' *satisfies the formula*

$$\forall t \in [T]\Big[\big(\forall i \in [m](s_{i,t} - s_{i,t-1} \geq 0)\big) \vee \big(\forall i \in [m](s_{i,t} - s_{i,t-1} \leq 0)\big)\Big] \tag{2.2}$$

Proof. Let $\Sigma = (\mathcal{S}, \mathcal{L})$ be a feasible schedule for $\mathcal{I}$. We give a procedure that repeatedly modifies Σ such that it satisfies (2.2) and reduces or retains its costs.

Let $t \in [T]$ be the first time slot that falsifies (2.2). If there does not exist such a time slot, we are finished. Otherwise, we can obtain machines $i, j \in [m]$ such that $s_{i,t} - s_{i,t-1} = 1$ and $s_{j,t} - s_{j,t-1} = -1$, that is server i powers on at time t and server j powers off. Without loss of generality, we may assume $i < j$. Since all servers are sleeping at time $t = 0$, we have

$$s_{k,1} - s_{k,0} = s_{k,1} - 0 = s_{k,1} \geq 0, \quad \forall k \in [m]$$

which satisfies formula (2.2) for $t = 1$. Thus, we further assume $t > 1$. Now consider the state sequences of server i and j:

$$S_i = (s_{i,1}, \ldots, s_{i,t-1} = 0, s_{i,t} = 1, \ldots, s_{i,T})$$
$$S_j = (s_{j,1}, \ldots, s_{j,t-1} = 1, s_{j,t} = 0, \ldots, s_{j,T})$$

We modify S_i and S_j by swapping their states for time slots $\geq t$, that is we set

$$S_i' \coloneqq (s_{i,1}, \ldots, s_{i,t-1} = 0, s_{j,t} = 0, \ldots, s_{j,T})$$
$$S_j' \coloneqq (s_{j,1}, \ldots, s_{j,t-1} = 1, s_{i,t} = 1, \ldots, s_{i,T})$$

Similarly, we need to swap the assigned loads for server i and j:

$$L_i' \coloneqq (\lambda_{i,1}, \ldots, \lambda_{i,t-1}, \lambda_{j,t}, \ldots, \lambda_{j,T})$$
$$L_j' \coloneqq (\lambda_{j,1}, \ldots, \lambda_{j,t-1}, \lambda_{i,t}, \ldots, \lambda_{i,T})$$

Finally, we construct a new schedule $\Sigma' \coloneqq (\mathcal{S}', \mathcal{L}')$ by setting

$$\mathcal{S}' \coloneqq (S_1, \ldots, S_{i-1}, S_i', S_{i+1}, \ldots, S_{j-1}, S_j', S_{j+1}, \ldots, S_T)$$
$$\mathcal{L}' \coloneqq (L_1, \ldots, L_{i-1}, L_i', L_{i+1}, \ldots, L_{j-1}, L_j', L_{j+1}, \ldots, L_T)$$

We now want to verify that Σ' is a feasible schedule, that is Σ' satisfies (1.2). For time slots $< t$, the schedules Σ' and Σ still coincide. For time slots $\geq t$, we only changed the order of summation in (1.2). Thus, Σ' is feasible.

Further, Σ and Σ' coincide in their operating costs; however, $\mathcal{S}'$ reduced the switching costs at time t. As we assume $\beta \geq 0$, we conclude $c(\Sigma') \leq c(\Sigma)$.

Moreover, we decreased the number of servers violating (2.2) at time t. Hence, by repeating described process on Σ', we obtain a terminating procedure that returns a schedule satisfying the conditions. □

Our next proposition – which will pose the cornerstone of our subsequent works – requires the use of Jensen's inequality, a well-known and frequently used analytic result found by Johan Jensen in 1906. It generalizes the idea that the secant line of a convex function lies above the graph of the function; more specifically, it states that the value of a convex function at a finite convex-combination of sampling points is less than or equal to the convex-combination of the function values at the sampling points.

Lemma 2.3 (Jensen's inequality)**.** *Let $f : \mathbb{R} \to \mathbb{R}$ be a convex function, $n \in \mathbb{N}$, $\lambda_1, \ldots, \lambda_n \in \mathbb{R}$, and $\mu_1, \ldots, \mu_n \in [0, 1]$ satisfying $\sum_{i=1}^{n} \mu_i = 1$. Then the following inequality holds:*

$$f\Big(\sum_{i=1}^{n} \mu_i \lambda_i\Big) \leq \sum_{i=1}^{n} \mu_i f(\lambda_i)$$

Proof. We proof the claim by induction on $n \in \mathbb{N}$.

- Basis: For $n = 1$, we have $\mu_1 = 1$ and thus

$$f\Big(\sum_{i=1}^{1} \mu_i \lambda_i\Big) = f(\lambda_1) = \sum_{i=1}^{1} \mu_i f(\lambda_i)$$

- Step: Let $n \in \mathbb{N}$ be arbitrary and fixed.
  - I.H.: The assertion holds for n.
  - Claim: The assertion holds for $n+1$.
  - Proof: Since $\sum_{i=1}^{n+1} \mu_i = 1$, $\mu_i \in [0,1]$, and $n+1 \geq 2$, at least one μ_i must be smaller than 1. Without loss of generality, we may assume $\mu_1 < 1$.

$$f\Big(\sum_{i=1}^{n+1} \mu_i \lambda_i\Big) = f\Big(\mu_1 \lambda_1 + \sum_{i=2}^{n+1} \mu_i \lambda_i\Big) \overset{\mu_1 \neq 1}{=} f\Big(\mu_1 \lambda_1 + (1-\mu_1) \sum_{i=2}^{n+1} \frac{\mu_i \lambda_i}{1-\mu_1}\Big)$$

As f is convex, we have

$$f\Big(\mu_1 \lambda_1 + (1-\mu_1) \sum_{i=2}^{n+1} \frac{\mu_i \lambda_i}{1-\mu_1}\Big) \overset{\text{f convex}}{\leq} \mu_1 f(\lambda_1) + (1-\mu_1) f\Big(\sum_{i=2}^{n+1} \frac{\mu_i \lambda_i}{1-\mu_1}\Big)$$

Since $\sum_{i=2}^{n+1} \frac{\mu_i}{1-\mu_1} = 1$, we can apply our induction hypothesis.

$$\mu_1 f(\lambda_1) + (1-\mu_1) f\Big(\sum_{i=2}^{n+1} \frac{\mu_i \lambda_i}{1-\mu_1}\Big) \overset{\text{I.H.}}{\leq} \mu_1 f(\lambda_1) + (1-\mu_1) \sum_{i=2}^{n+1} \frac{\mu_i}{1-\mu_1} f(\lambda_i)$$

We combine our steps and obtain

$$\begin{aligned} f\Big(\sum_{i=1}^{n+1} \mu_i \lambda_i\Big) &\leq \mu_1 f(\lambda_1) + (1-\mu_1) \sum_{i=2}^{n+1} \frac{\mu_i}{1-\mu_1} f(\lambda_i) \\ &= \mu_1 f(\lambda_1) + \sum_{i=2}^{n+1} \mu_i f(\lambda_i) = \sum_{i=1}^{n+1} \mu_i f(\lambda_i) \end{aligned}$$

□

Next, we consider the sequence of active servers. For this, let $\mathcal{X}$ denote the sequence of sums of active servers at each time slot t, that is

$$\mathcal{X} \coloneqq \big(x_1 = \sum_{i=1}^{m} s_{i,1}, \ldots, x_T = \sum_{i=1}^{m} s_{i,T}\big) \in \{0, \ldots, m\}^T$$

As we assume all machines sleeping at times $t \notin [T]$, we have $x_t = 0$ for $t \notin [T]$. We now want to establish an optimal scheduling strategy given a fixed number of active servers. It turns out that an even load distribution seems a very desirable strategy.

Proposition 2.4 (Even load distribution)**.** *Given $x_t \in \mathbb{N}$ active servers in time slot t, an arrival rate $\lambda_t \in [0, x_t]$, and a convex cost function f, a most cost-efficient and feasible scheduling strategy is to assign each active server a load of λ_t / x_t.*

Proof. Let Σ be an arbitrary, feasible schedule using x_t servers in time slot t, and let A be its set of active servers in time slot t, that is $A \coloneqq \{i \in [m] \mid s_{i,t} = 1\}$. Consider the operating costs of Σ at time t given by

$$\sum_{i=1}^{m} \big(f(\lambda_{i,t}) \cdot s_{i,t}\big) = \sum_{i \in A} \big(f(\lambda_{i,t}) \cdot 1\big) + \sum_{i \in [m] \setminus A} \big(f(\lambda_{i,t}) \cdot 0\big) = \sum_{i \in A} f(\lambda_{i,t})$$

Since Σ is feasible (see constraint (1.2)), we have

$$\sum_{i \in A} \lambda_{i,t} = \lambda_t$$

Hence, we can obtain weights $\mu_1, \dots, \mu_{x_t} \in [0, 1]$ that relate $\lambda_{i,t}$ and λ_t for $i \in A$ such that

$$\sum_{i=1}^{x_t} \mu_i = 1 \quad \text{and} \quad \sum_{i \in A} f(\lambda_{i,t}) = \sum_{i=1}^{x_t} f(\mu_i \lambda_t) \tag{2.5}$$

In particular, we have

$$\sum_{i=1}^{x_t} \mu_i \lambda_t = \lambda_t \tag{2.6}$$

Using these weights, we now consider the operating costs of a schedule Σ^* that evenly distributes λ_t to its x_t active servers:

$$\sum_{i=1}^{x_t} f\Big(\frac{\lambda_t}{x_t}\Big) = x_t f\Big(\frac{\lambda_t}{x_t}\Big) \overset{(2.6)}{=} x_t f\Big(\sum_{i=1}^{x_t} \frac{\mu_i \lambda_t}{x_t}\Big)$$

With the fact that $\sum_{i=1}^{x_t} (1/x_t) = 1$ and the use of Jensen's inequality (Lemma 2.3), we can give an upper bound for the costs:

$$x_t f\Big(\frac{\lambda_t}{x_t}\Big) \overset{2.3}{\leq} x_t \sum_{i=1}^{x_t} \frac{1}{x_t} f(\mu_i \lambda_t) = \frac{x_t}{x_t} \sum_{i=1}^{x_t} f(\mu_i \lambda_t) = \sum_{i=1}^{x_t} f(\mu_i \lambda_t) \overset{(2.5)}{=} \sum_{i \in A} f(\lambda_{i,t})$$

Thus, the operating costs of Σ^* give a lower bound for the operating costs of Σ, and the claim follows. □

As a special case, we can apply our just derived proposition to optimal schedules.

Corollary 2.7. *Given a problem instance $\mathcal{I}$, there exists an optimal schedule Σ^* that evenly distributes its arrival rates to its active servers in each time slot.*

Proof. Let $\Sigma = (\mathcal{S}, \mathcal{L})$ be an optimal schedule for $\mathcal{I}$. We exchange $\mathcal{L}$ with a new strategy $\mathcal{L}^*$ that evenly distributes the arrival rates to all active servers of Σ in each time slot, that is we set $\Sigma^* \coloneqq (\mathcal{S}, \mathcal{L}^*)$. By Proposition 2.4, we have $c(\Sigma^*) = c(\Sigma)$. The claim follows. □

As a result of Corollary 2.7, we can restrict ourselves to finding an optimal schedule that evenly distributes its arrival rates to its active servers. We now combine our results to derive the main theorem of our preliminary work.

Theorem 2.8. *Given a problem instance $\mathcal{I}$, there exists an optimal schedule Σ^* that evenly distributes its arrival rates to its active servers in each time slot and further satisfies (2.2).*

Proof. By Corollary 2.7, we obtain an optimal schedule Σ that evenly distributes its arrival rates to its active servers. Applying the procedure given in Proposition 2.1 to Σ yields Σ^* that further satisfies (2.2). □

Theorem 2.8 allows us to subsequently identify an optimal schedule by its sequence of active servers $\mathcal{X}$ and thereby to simplify our optimization conditions (1.1) and (1.2). For this, given a problem instance $\mathcal{I}$, we define the operating costs function $c_{op}(x, \lambda)$, which describes the costs incurred by evenly distributing λ on x active servers using f:

$$c_{op} : \{0, \dots, m\} \times [0, m] \to \mathbb{R} \cup \{\infty\}, \quad c_{op}(x, \lambda) = \begin{cases} 0, & \text{if } x = 0 \\ x f(\lambda / x), & \text{if } x \neq 0 \wedge \lambda \leq x \\ \infty, & \text{if } x \neq 0 \wedge \lambda > x \end{cases}$$

We assign infinite costs in case $\lambda > x$ as there would be too few active servers to process the arrival rate, that is the schedule would not be feasible. Next, we define the switching costs function $c_{sw}(x_{t-1}, x_t)$, which describes the incurred costs when changing the number of active server from x_{t-1} to x_t:

$$c_{sw}(x_{t-1}, x_t) \coloneqq \beta \max\{0, x_t - x_{t-1}\} \tag{2.9}$$

Lastly, we can define the costs function $c(x_{t-1}, x_t, \lambda_t)$, which describes the incurring costs for a single time step using an even distribution of loads:

$$c(x_{t-1}, x_t, \lambda_t) \coloneqq c_{op}(x_t, \lambda_t) + c_{sw}(x_{t-1}, x_t) \tag{2.10}$$

The optimization conditions for a schedule now simplify to one single minimization:

$$\text{minimize} \quad c(\mathcal{X}) \coloneqq \sum_{t=1}^{T} c(x_{t-1}, x_t, \lambda_t) \tag{2.11}$$

We subsequently call a schedule $\mathcal{X}$ *optimal* if it satisfies (2.11). Finally, we can approach the first main goal of our work: the development of an optimal offline algorithm.

3 Optimal Offline Scheduling

Now, as we have finished our preliminary work, we are able to derive our first two algorithms; more precisely, we will derive two optimal offline algorithms in this chapter. We will start by reducing our problem specified by $\mathcal{I}$ to a shortest path problem of a level structured graph G. We then proceed to find a shortest path in G and thereby an optimal schedule for $\mathcal{I}$ in pseudo-polynomial time $\Theta(Tm^2)$. After that, we refine our initial approach to derive an improved algorithm with pseudo-linear time complexity $\Theta(Tm)$.

3.1 A Pseudo-Polynomial Algorithm

Let $\mathcal{I}$ be a problem instance. Thanks to our preliminary work, we know that there exists an optimal schedule which is identifiable by its sequence of active servers $\mathcal{X}$. In order to find this sequence $\mathcal{X}$, consider the weighted, level structured graph G defined as follows:

$$
\begin{aligned}
V &\coloneqq \left\{v_{x,t} \mid x \in \{0,\dots,m\}, t \in [T]\right\} \uplus \{v_{0,0}, v_{0,T+1}\} \\
E &\coloneqq \left\{(v_{x,t}, v_{x',t+1}) \mid x, x' \in \{0,\dots,m\}, t \in \{0,\dots,T\}, v_{x,t}, v_{x',t+1} \in V\right\} \\
c_G(v_{x,t}, v_{x',t+1}) &\coloneqq c(x, x', \lambda_{t+1}) \\
G &\coloneqq (V, E, c_G)
\end{aligned}
$$

For any possible number of active servers x and any time slot t, we add a node $v_{x,t}$. Moreover, we add a start node $v_{0,0}$ as well as an end node $v_{0,T+1}$. Next, we connect all nodes to their successors with respect to time. Semantically, $v_{x,t}$ denotes the state of distributing the arrival rate λ_t evenly to x servers in time slot t. For any edge connecting $v_{x,t}$ with $v_{x',t+1}$, we assign costs $c(x, x', \lambda_{t+1})$, i.e. the edge's cost corresponds to switching from x to x' machines and processing the load λ_{t+1} with x' machines. A graphical representation can be found in the following figure.

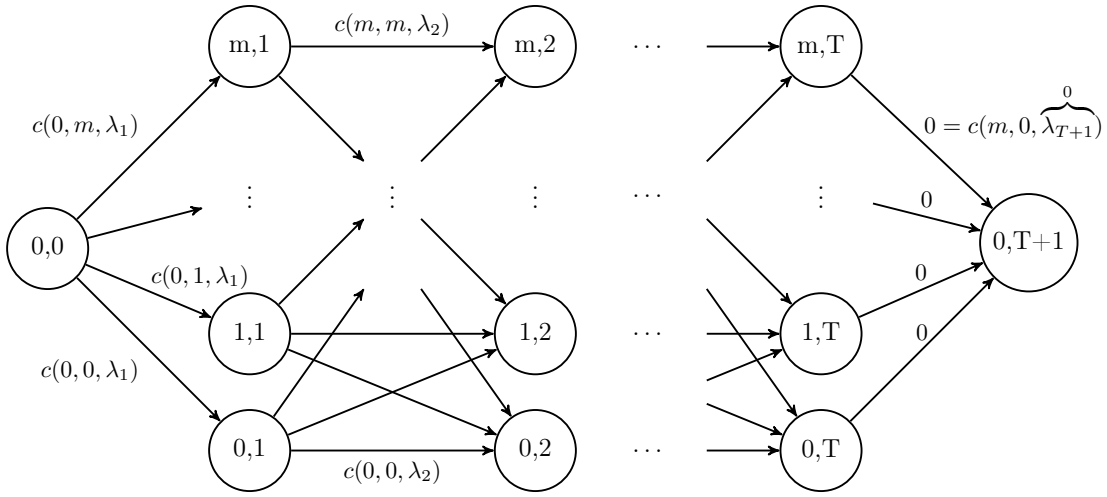


Figure 3.1: Level structured graph for a pseudo-polynomial, optimal offline algorithm

As we can see in Figure 3.1, the cost of a path $P = (v_{0,0}, v_{x_1,1}, \ldots, v_{x_T,T}, v_{0,T+1})$ from our start node $v_{0,0}$ to our end node $v_{0,T+1}$ is given by

$$c(P) = c(0, x_1, \lambda_1) + \sum_{t=2}^{T} c(x_{t-1}, x_t, \lambda_t) + \overbrace{c(x_T, 0, 0)}^{0} = c(0, x_1, \lambda_1) + \sum_{t=2}^{T} c(x_{t-1}, x_t, \lambda_t) \quad (3.1)$$

Note that the cost of such a path directly corresponds to that of a schedule $\mathcal{X}$ (see (2.11)). Any shortest path from $v_{0,0}$ to $v_{0,T+1}$ is thus forced to minimize the cost of the corresponding schedule. Needless to say, this demands for a proof of correctness.

Lemma 3.2. *Let $\boldsymbol{\mathcal{X}}$ be the set of all schedules $\mathcal{X}$ for $\mathcal{I}$, and let $\boldsymbol{\mathcal{P}}$ the set of all paths from $v_{0,0}$ to $v_{0,T+1}$. The map*

$$\Phi : \boldsymbol{\mathcal{P}} \to \boldsymbol{\mathcal{X}}, \quad (v_{0,0}, v_{x_1,1}, v_{x_2,2}, \ldots, v_{x_T,T}, v_{0,T+1}) \mapsto (x_1, \ldots, x_T)$$

is a bijection with inverse map

$$\Psi : \boldsymbol{\mathcal{X}} \to \boldsymbol{\mathcal{P}}, \quad (x_1, \ldots, x_T) \mapsto (v_{0,0}, v_{x_1,1}, v_{x_2,2}, \ldots, v_{x_T,T}, v_{0,T+1})$$

satisfying $c(\mathcal{X}) = c\big(\Psi(\mathcal{X})\big)$.

Proof. It is easy to check that $\Psi \circ \Phi = id_{\boldsymbol{\mathcal{P}}}$ and $\Phi \circ \Psi = id_{\boldsymbol{\mathcal{X}}}$. Thus, Φ is indeed bijective with inverse map Ψ. Next, let $\mathcal{X} = (x_1, \ldots, x_T)$ be a schedule for $\mathcal{I}$. We have

$$P := \Psi(\mathcal{X}) = (v_{0,0}, v_{x_1,1}, v_{x_2,2}, \ldots, v_{x_T,T}, v_{0,T+1})$$

We examine the cost of $\mathcal{X}$ and conclude

$$c(\mathcal{X}) \overset{(2.11)}{=} \sum_{t=1}^{T} c(x_{t-1}, x_t, \lambda_t) = \underbrace{c(x_0, x_1, \lambda_1)}_{=c(0,x_1,\lambda_1)} + \sum_{t=2}^{T} c(x_{t-1}, x_t, \lambda_t) \overset{(3.1)}{=} c(P)$$

Which shows that Ψ and, as a consequence, Φ are cost-preserving maps. □

We can now use this bijection to obtain our desired result: the correspondence between optimal schedules and shortest paths.

Theorem 3.3. *There is a cost-preserving bijection between optimal schedules $\mathcal{X}$ for $\mathcal{I}$ and shortest paths from $v_{0,0}$ to $v_{0,T+1}$.*

Proof. By Lemma 3.2, we have a bijection Ψ between schedules $\mathcal{X}$ and paths from $v_{0,0}$ to $v_{0,T+1}$ obeying $c(\mathcal{X}) = c\big(\Psi(\mathcal{X})\big)$. Thus, we have

$$c(\mathcal{X}) \text{ minimal} \iff c\big(\Psi(\mathcal{X})\big) \text{ minimal}$$

and the claim follows. □

In the following, we give an algorithm based on our just verified construction. We split our procedure into two subroutines.

SHORTEST_PATHS calculates the minimum costs of the nodes of our graph, layer by layer; that is, it calculates the shortest paths following the topological sorting of the graph. It returns the minimum costs to all nodes as well as the predecessor of any node in respect to its shortest path.

EXTRACT_SCHEDULE uses the predecessors calculated by SHORTEST_PATHS in order to obtain the sequence of nodes describing a shortest path; thereby, it calculates an optimal schedule for our problem instance.

Algorithm 1 Pseudo-polynomial optimal offline scheduling

```
1: function OPTIMAL_OFFLINE_SCHEDULING(m, T, Λ, β, f)
2:     (C, P) ← SHORTEST_PATHS(m, T, Λ, β, f)
3:     X ← EXTRACT_SCHEDULE(P, T)
4:     return X

5: function SHORTEST_PATHS(m, T, Λ, β, f)
       ▷ Allocate nodes' costs and predecessors tables
6:     let C[0...m, 1...T] and P[0...m, 1...T] be new tables
7:     for x ← 0 to m do                                   ▷ Initialize first layer
8:         P[x, 1] ← 0 and C[x, 1] ← c(0, x, λ_1)
9:     for t ← 2 to T do                  ▷ Iterative calculate costs and predecessors
10:        for x' ← 0 to m do
11:            P[x', t] ← argmin_{x∈{0,...,m}} {C[x, t−1] + c(x, x', λ_t)}   ▷ Get best preceding choice
12:            C[x', t] ← C[P[x', t], t−1] + c(P[x', t], x', λ_t)            ▷ Set the costs
13:    return (C, P)

14: function EXTRACT_SCHEDULE(P, T)
15:    let X[T] be a new array                              ▷ Allocate the schedule array
16:    X[T] ← argmin_{x∈{0,...,m}} {C[x, T]}                ▷ Find best choice for last time slot
17:    for t ← T − 1 to 1 do          ▷ Iteratively obtain schedule from predecessors table
18:        X[t] ← P[X[t + 1], t + 1]
19:    return X
```

The correctness of Algorithm 1 directly follows from the correctness of the shortest path calculation for directed acyclic graphs (for a proof see [1]) and from Theorem 3.3.

Naturally, we are interested in our algorithm's runtime and memory complexity. For this, we first need to consider the size of our input $\mathcal{I} = (m, T, \Lambda, \beta, f)$. In theory, our function $f : [0, 1] \to \mathbb{R}$ may be, for example, easily defined by saying $f(x) \coloneqq x^2$; however, practically, it is difficult to answer how such a function may be specified and what the size of such a function as part of the input may be. For simplicity, we consider the size of f negligible in comparison with the size of the remaining input variables. Further, we assume a real number r to require $\Theta(\log_2(r))$ bits for its encoding. The size of our input is then given by

Real numbers? How should I write this? What about $\log_2(0)$?

The following calculation. $\mathcal{O}(m)$?

$$
\begin{aligned}
size(\mathcal{I}) &= size(m) + size(T) + size(\Lambda) + size(\beta) \\
&= \Theta\big(\log_2(m)\big) + \Theta\big(\log_2(T)\big) + \sum_{i=1}^{T} \Theta\big(\log_2(\lambda_i)\big) + \Theta\big(\log_2(\beta)\big) \\
&= \Theta\big(\log_2(m) + \log_2(T) + \log_2(\beta)\big) + \sum_{i=1}^{T} \Theta\big(\log_2(\overbrace{\lambda_i}^{\leq m})\big) \\
&\in \Theta\big(\log_2(m) + \log_2(T) + \log_2(\beta)\big) + \mathcal{O}\big(T \log_2(m)\big) \\
&\in \mathcal{O}\big(T \log_2(m) + \log_2(\beta)\big)
\end{aligned} \tag{3.4}
$$

For our runtime analysis, we assume that a call of the costs function $c(\cdot, \cdot, \cdot)$ has constant cost. Subroutine SHORTEST_PATHS needs $\Theta(m)$ iterations for its initialization and $\Theta(Tm^2)$ steps for the iterative calculation. In addition, EXTRACT_SCHEDULE needs $\Theta(m)$ steps for its initial minimization search and $\Theta(T)$ iterations for its schedule retrieval. Hence, we receive a runtime of

$$\Theta(m + Tm^2 + m + T) = \Theta(Tm^2)$$

The runtime is polynomial in the numeric value of m and T; however, it is exponential in the size of the input since, as we saw in (3.4), we only need $\log_2(m)$ bits to encode m. Hence, the algorithm is pseudo-polynomial.

Our memory demand is defined by the size of the tables C and P. As both tables are of size $\Theta(Tm)$, we have a memory complexity of $\Theta(Tm)$ Is this enough?

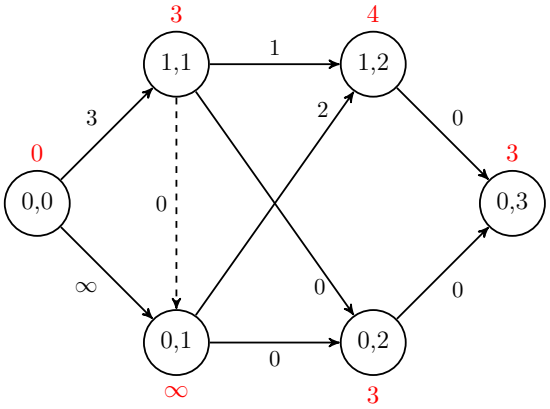
3.2 A Pseudo-Linear Algorithm

The algorithm developed in Section 3.1 is of a quite simple nature. Its underlying graph G is able to represent any possible schedule $\mathcal{X}$ since we simply we add an edge for any possible scheduling choice at any possible time slot. This approach seems rather intuitive and readily verifiable, but this convenience comes with a cost; the density of G causes a quadratic runtime in the number of servers m. In order to improve the runtime to pseudo-linear complexity, we need to "thin out" our graph.

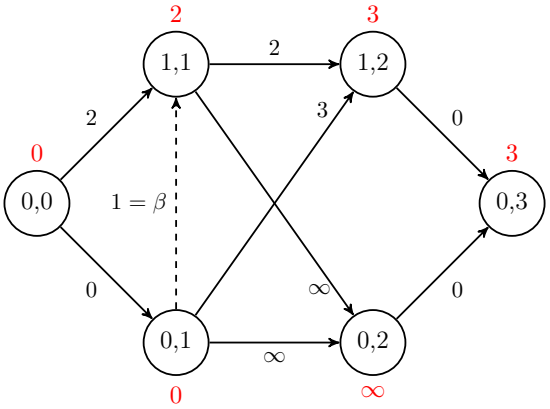
Let us revise our initial algorithm. Our graph consists of nodes $v_{x,t}$. Any such node $v_{x,t}$ denotes the state of distributing the arrival rate λ_t evenly to x servers in time slot t. The algorithm calculates the minimum costs to all nodes. The cost of a node $v_{x,t}$ thus corresponds to the lowest achievable cost up to time slot t of all schedules $\mathcal{X}$ that assign x servers at time t to process the arrival rate λ_t. In particular, the cost of the end node $v_{0,T+1}$ tells us the minimum cost of all schedules. This approach, however, does not consider the possibility to schedule $y \neq x$ servers in time slot t and to switch to x servers

just at the very last moment of t when calculating the cost of $v_{x,t}$. Consider the following example:

Example 3.5. Let $\mathcal{I}_i = (m = 1, T = 2, \Lambda_i, \beta = 1, f)$ be the inputs for two problem instances with $f(x) = x^2 + 1$, $\Lambda_1 = (1, 0)$, and $\Lambda_2 = (0, 1)$. Below we illustrate the graphs of the two problem instances and the corresponding calculations done by the algorithm derived in Section 3.1.



Problem $\mathcal{I}_1$: State $v_{0,1}$ could be reached with cost 3 by moving down from node $v_{1,1}$.



Problem $\mathcal{I}_2$: State $v_{1,1}$ could be reached with cost 1 by moving up from $v_{0,1}$.

Figure 3.2: Two examples depicting a shortcoming of our initial approach. The calculated nodes' costs are highlighted in red. Dashed edges are not part of the initial algorithm.

Although our algorithm delivers the correct end results, its immediate steps are somewhat unsatisfying. We want our states to capture a more general notion than given in Section 3.1; preferably, we would like a node $v_{x,t}$ to denote the state of having x active servers at the end of time slot t. In practice, we may reach such a state $v_{x,t}$ by moving down from a state $v_{y^{\downarrow},t}$ where $y^{\downarrow} > x$ with cost 0 or by moving up from a state $v_{y^{\uparrow},t}$ where $y^{\uparrow} < x$ with cost $\beta(x - y^{\uparrow})$. In order to allow for these new possibilities, given a problem instance $\mathcal{I}$,

we define a directed, weighted graph as follows:

$$
\begin{aligned}
V &:= \{v_{x,t\downarrow} \mid x \in \{0,\dots,m\}, t \in [T]\} \cup \{v_{x,t\uparrow} \mid x \in \{0,\dots,m\}, t \in [T-1]\} \cup \{v_{0,0}\} \\
E_s &:= \{(v_{0,0}, v_{x,1\downarrow}) \mid x \in \{0,\dots,m\}\} \\
E_{\downarrow} &:= \{(v_{x,t\downarrow}, v_{x-1,t\downarrow}) \mid x \in [m], t \in [T]\} \\
E_{\uparrow} &:= \{(v_{x-1,t\uparrow}, v_{x,t\uparrow}) \mid x \in [m], t \in [T-1]\} \\
E_{\downarrow\uparrow} &:= \{(v_{x,t\downarrow}, v_{x,t\uparrow}) \mid x \in \{0,\dots,m\}, t \in [T-1]\} \\
E_{\uparrow\downarrow} &:= \{(v_{x,t\uparrow}, v_{x,t+1\downarrow}) \mid x \in \{0,\dots,m\}, t \in [T-1]\} \\
E &:= E_s \cup E_{\downarrow} \cup E_{\uparrow} \cup E_{\downarrow\uparrow} \cup E_{\uparrow\downarrow} \\
c_G(e) &:= \begin{cases} c(0,x,\lambda_1), & \text{if } e = (v_{0,0}, v_{x,1\downarrow}) \in E_s \\ c_{op}(x,\lambda_{t+1}), & \text{if } e = (v_{x,t\uparrow}, v_{x,t+1\downarrow}) \in E_{\uparrow\downarrow} \\ \beta, & \text{if } e \in E_{\uparrow} \\ 0, & \text{if } e \in (E_{\downarrow} \cup E_{\downarrow\uparrow}) \end{cases} \\
G &:= (V, E, c_G)
\end{aligned}
$$

A more appealing, graphical representation can be found in the following figure.

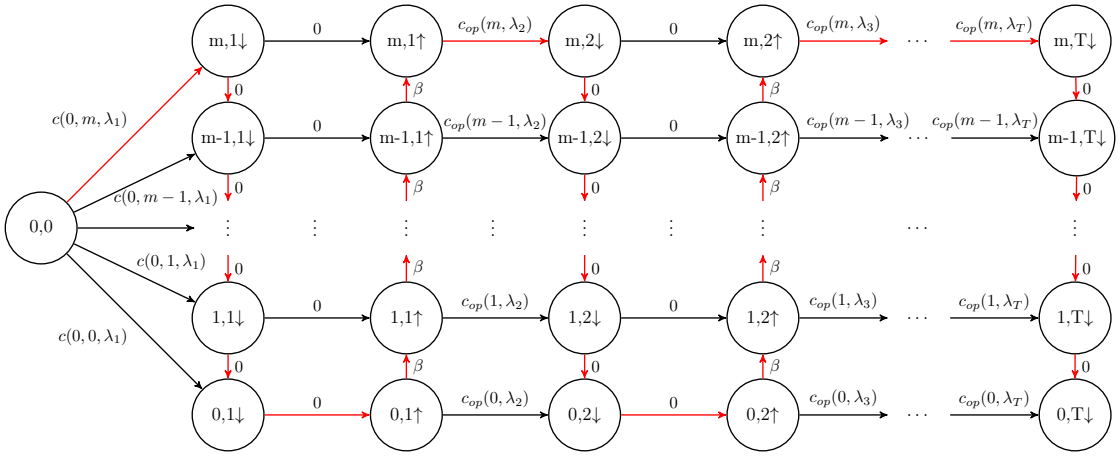


Figure 3.3: Graph for a pseudo-linear, optimal offline algorithm; the path of the topological sorting is highlighted in red.

For any possible number of active servers x and any time slot t, we add a node $v_{x,t\downarrow}$. Semantically, the cost of $v_{x,t\downarrow}$ will denote the minimum cost up to time slot t when processing λ_t with x or more servers. Further, for any time slot $t \in [T-1]$, we add a node $v_{x,t\uparrow}$. The cost of $v_{x,t\uparrow}$ will denote the minimum cost of having x active servers at the end of time slot t. Moreover, we add a start node $v_{0,0}$. Our end node will be $v_{0,T\downarrow}$.

The set of edges E_s denotes the start initialization step. An edge $(v_{x,t\uparrow}, v_{x,t+1\downarrow}) \in E_{\uparrow\downarrow}$ accounts for the operating costs that incur when processing the arrival rate λ_{t+1} with x active servers.

After any time step from $t-1$ to t that incurs operating costs, we have a minimization step in our graph. For this, we first move down the chain $v_{m,t\downarrow}, v_{m-1,t\downarrow}, \ldots, v_{0,t\downarrow}$ using edges from $E_\downarrow$ with cost 0. Then we proceed to move to the right from $v_{0,t\downarrow}$ to $v_{0,t\uparrow}$. Lastly, we move up the chain $v_{0,t\uparrow}, v_{1,t\uparrow}, \ldots, v_{m,t\uparrow}$ using edges from $E_\uparrow$ with cost β. In order to have the possibility to keep the calculated costs of $v_{x,t\downarrow}$ during the upward movement, we add edges $(v_{x,t\downarrow}, v_{x,t\uparrow}) \in E_{\downarrow\uparrow}$ with cost 0.

This minimization step is the key to our runtime improvement. It facilitates the determination of the best predecessor of a state $v_{x,t\downarrow}$ since we already know that the minimum cost of having x servers at the end of time slot $t-1$ is stored in $v_{x,t-1\uparrow}$. Thus, the cheapest possibility to process the next arrival rate λ_t using x servers can simply be calculated by adding $c_{op}(x, \lambda_t)$ to the cost of $v_{x,t-1\uparrow}$. Consequently, the cost of $v_{x,t\downarrow}$ is given by the minimum of $v_{x,t-1\uparrow} + c_{op}(x, \lambda_t)$ and $v_{x+1,\downarrow}$.

As one can see in Figure 4.1, we stretched our graph but at the same time also greatly reduced the number of edges compared to our initial approach in Section 3.1. By following the colored path of the topological sorting, we can work our way through the graph to calculate the shortest paths, ultimately reaching the destination $v_{0,T\downarrow}$. The cost of our destination $v_{0,T\downarrow}$ will denote the minimum cost up to time slot T when processing λ_T with 0 or more servers. Hence, it will contain our desired end result — the minimum cost of all possible schedules.

Our next task shall be the verification of our new construction. For this, we first examine the possible paths from $v_{0,0}$ to $v_{0,T\downarrow}$ in our graph. In contrast to our approach in Section 3.1, our new graph contains paths that do not directly correspond to a schedule $\mathcal{X}$. For example, consider the following path:

$$P \coloneqq (v_{0,0}, v_{1,1\downarrow}, v_{0,1\downarrow}, v_{0,1\uparrow}, v_{1,1\uparrow}, v_{2,1\uparrow}, v_{2,2\downarrow}, \ldots, v_{0,T\downarrow})$$

A schedule corresponding to P would use one active server in its first time slot, then power down this server, and subsequently turn on two servers to process the next arrival rate. This seems somewhat unreasonable. We could just keep the initial server on to save switching costs (note the correspondence to Proposition 2.1). In fact, this behavior cannot even be represented using our schedule notation $\mathcal{X}$ and optimization condition (2.11). We can, however, modify P to represent a more reasonable sequence, that is we set

$$P' \coloneqq (v_{0,0}, v_{1,1\downarrow}, v_{1,1\uparrow}, v_{2,1\uparrow}, v_{2,2\downarrow}, \ldots, v_{0,T\downarrow})$$

This revised path now pleasantly translates to a schedule $\mathcal{X}$, in this case $\mathcal{X} = (1, 2, \ldots)$. We now want to give a more formal definition of our observation.

Definition 3.6 (Reasonable paths)**.** Let P be a path from $v_{0,0}$ to $v_{0,T\downarrow}$. For any time slot $t \in [T]$, let $E^t_\downarrow$ be the set of edges $(v_{x,t\downarrow}, v_{x-1,t\downarrow}) \in E_\downarrow$ used by P at time t. Similarly, let $E^t_\uparrow$ be the set of edges $(v_{x,t\uparrow}, v_{x+1,t\uparrow}) \in E_\uparrow$ used by P at time $t \in [T-1]$. The path P is called *reasonable* if for any time slot $t \in [T-1]$, the path does not shut down and power on servers simultaneously at t. More formally, P must satisfy the formula

$$\forall t \in [T-1] \left(E^t_\downarrow = \emptyset \vee E^t_\uparrow = \emptyset\right) \tag{3.7}$$

Using this definition, we can now verify that our reasonable paths indeed deserve to be called *reasonable*.

Proposition 3.8. *Any given path P from $v_{0,0}$ to $v_{0,T\downarrow}$ can be transformed to a reasonable path P' with $c(P') \leq c(P)$.*

Proof. Let P be a path from $v_{0,0}$ to $v_{0,T\downarrow}$. We give a procedure that repeatedly modifies P such that it satisfies (3.7) and reduces or retains its costs.

Let $t \in [T-1]$ be the first time slot that falsifies (3.7). If there does not exist such a time slot, we are finished. Otherwise, let $E^t_\downarrow$ and $E^t_\uparrow$ be its sets of edges as defined in Definition 3.6. Since P falsifies (3.7) at time t, both $E^t_\downarrow$ as well as $E^t_\uparrow$ must be non-empty. Thus, we can obtain the "maximum" nodes involved in these sets.

$$x_s \coloneqq \max\left\{x \in [m] \mid (v_{x,t\downarrow}, v_{x-1,t\downarrow}) \in E^t_\downarrow\right\}$$
$$x_e \coloneqq \max\left\{x \in [m] \mid (v_{x-1,t\uparrow}, v_{x,t\uparrow}) \in E^t_\uparrow\right\}$$

Next, consider the subpath $S = (v_{x_s,t\downarrow}, \ldots, v_{x_e,t\uparrow})$ of P. Note that the subpath in particular uses all edges from $E^t_\downarrow$ and $E^t_\uparrow$. Naturally, a most cost-efficient path S' from node $v_{x_s,t\downarrow}$ to $v_{x_e,t\uparrow}$ minimizes the number of edges $(v_{x,t\uparrow}, v_{x+1,t\uparrow})$ since each of these edges incurs cost $\beta \geq 0$. For construction of S', we observe that we must not shut down servers if $x_s \leq x_e$, and that we must not power on servers if $x_s \geq x_e$; thus, we consider three cases for S':

$$S' \coloneqq \begin{cases} (v_{x_s,t\downarrow}, v_{x_s,t\uparrow}, v_{x_s+1,t\uparrow}, \ldots, v_{x_e,t\uparrow}), & \text{if } x_s < x_e \\ (v_{x_s,t\downarrow}, v_{x_s-1,t\downarrow}, \ldots, v_{x_e,t\downarrow}, v_{x_e,t\uparrow}), & \text{if } x_s > x_e \\ (v_{x_s,t\downarrow}, v_{x_e,t\uparrow}), & \text{if } x_s = x_e \end{cases}$$

In each case, S' uses edges from at most one of the sets $E^t_\downarrow$ and $E^t_\uparrow$. Thus, by replacing the subpath S of P with S', we obtain a new schedule P' that satisfies (3.7) up to and including time slot t. Moreover, the paths P and P' coincide in their costs before visiting $v_{x_s,t\downarrow}$ and after visiting $v_{x_e,t\uparrow}$; however, they differ in that there are less switching costs β at time t using P'. As we assume $\beta \geq 0$, we conclude $c(P') \leq c(P)$.

Hence, by repeating described process on P', we obtain a terminating procedure that returns a path satisfying the conditions. □

As a result of Proposition 3.8, we shall subsequently focus our attention on reasonable paths. Our goal is to establish the connection between reasonable paths P of our graph and schedules $\mathcal{X}$. Intuitively, one can see that such a path P is uniquely determined by the "maximum" nodes $v_{x,t\downarrow}$ taken by P at any time slot t; these nodes represent the state of processing λ_t with x servers. Consider the following example:

Example 3.9. Let $\mathcal{I} = \big(m = 3, T = 3, \Lambda = (3, 0, 1), \beta, f\big)$ be the input for a problem instance. Then one example of a reasonable path is given by

$$P = (v_{0,0}, v_{3,1\downarrow}, v_{2,1\downarrow}, v_{1,1\downarrow}, v_{0,1\downarrow}, v_{0,1\uparrow}, v_{0,2\downarrow}, v_{0,2\uparrow}, v_{1,2\uparrow}, v_{1,3\downarrow}, v_{0,3\downarrow})$$

A schedule corresponding to P would use three active server in its first time slot, then power down all servers for the second time slot, and ultimately power on one server to process the last arrival rate; that is, the corresponding schedule of P is $\mathcal{X} = (3, 0, 1)$. This sequence also corresponds to the sequence of "maximum" nodes $v_{x,t\downarrow}$ taken by P, as can be seen in the following figure.

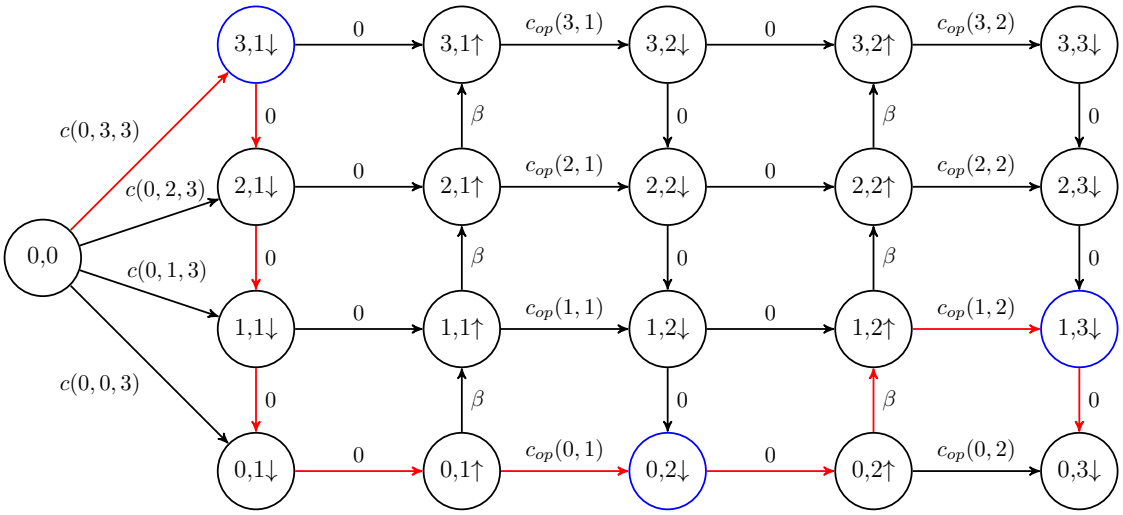


Figure 3.4: Illustration of the path P. The path is highlighted in red. The maximum nodes $v_{x,t\downarrow}$ taken by P are highlighted in blue.

Conversely, given the schedule $\mathcal{X} = (3, 0, 1)$, is can easily be seen that there exists only one reasonable path corresponding to $\mathcal{X}$, namely P.

Before we approach our next lemma, which formalizes these ideas, we need to give a precise definition of what we understand as *"maximum" nodes* $v_{x,t\downarrow}$ used by a reasonable path.

Definition 3.10. Let P be a reasonable path. For any $t \in [T]$, let $V_{\downarrow}^{t}$ be the set of nodes $v_{x,t\downarrow}$ visited by P. The "maximum" node $v_{x,t\downarrow}$ at each time slot $t \in [T]$ can then be identified by

$$x_t \coloneqq \max\big\{x \in \{0,\dots,m\} \mid v_{x,t\downarrow} \in V_{\downarrow}^{t}\big\}$$

Remember that a schedule $\mathcal{X} = (x_1,\dots,x_T)$ also uses the notation x_t to identify the number of active severs at time t. Needless to say, this clash of names is intentional and justified by the next lemma.

Lemma 3.11. *Let $\boldsymbol{\mathcal{X}}$ be the set of all schedules $\mathcal{X}$ for $\mathcal{I}$, and let $\boldsymbol{\mathcal{P}}$ the set of all reasonable paths. The map*

$$\Phi : \boldsymbol{\mathcal{P}} \to \boldsymbol{\mathcal{X}}, \quad P \mapsto (x_1,\dots,x_T)$$

is a bijection satisfying $c(P) = c\big(\Phi(P)\big)$.

Proof. First, we check that Φ is bijective, i.e. Φ is injective and surjective.

Let P and P' be reasonable paths with $\Phi(P) = \Phi(P')$. Then P as well as P' must start with the edge $(v_{0,0}, v_{x_1,1\downarrow})$. As P and P' are reasonable, they both satisfy $E_{\downarrow}^{t} = \emptyset \vee E_{\uparrow}^{t} = \emptyset$ for each time slot t. Due to this restriction, their paths between $v_{x_t,t\downarrow}$ and $v_{x_{t+1},t+1\downarrow}$ must coincide for $t \in [T-1]$. Further, the path from $v_{x_T,T}$ to $v_{0,T}$ is unique in our graph. Thus, we have $P = P'$, which shows the injectivity of Φ.

Next, let $(x_1,\dots,x_T) \in \boldsymbol{\mathcal{X}}$ be a schedule for $\mathcal{I}$. For any $t \in [T-1]$, we set

$$S_t \coloneqq \begin{cases} (v_{x_t,t\downarrow}, v_{x_t,t\uparrow}, v_{x_t+1,t\uparrow}, \dots, v_{x_{t+1},t\uparrow}), & \text{if } x_t < x_{t+1} \\ (v_{x_t,t\downarrow}, v_{x_t-1,t\downarrow}, \dots, v_{x_{t+1},t\downarrow}, v_{x_{t+1},t\uparrow}), & \text{if } x_t > x_{t+1} \\ (v_{x_t,t\downarrow}, v_{x_{t+1},t\uparrow}), & \text{if } x_t = x_{t+1} \end{cases}$$

We then concatenate these subpaths to construct a reasonable path

$$P \coloneqq (v_{0,0}, S_1, S_2, \dots, S_{T-1}, v_{x_T,T\downarrow}, v_{x_T-1,T\downarrow}, \dots, v_{0,T\downarrow})$$

Evidently, the constructed path satisfies $\Phi(P) = \mathcal{X}$, which shows the surjectivity of Φ.

It remains to show that Φ is cost-preserving. For this, let P be a reasonable path and let $\mathcal{X} \coloneqq \Phi(P) = (x_1,\dots,x_T)$ be its image. As P is reasonable, we have $E_{\downarrow}^{t} = \emptyset \vee E_{\uparrow}^{t} = \emptyset$ for any time slot t. If $E_{\uparrow}^{t}$ is empty, we have $x_{t-1} \geq x_t$. The cost between the nodes $v_{x_{t-1},t-1\downarrow}$ and $v_{x_t,t\downarrow}$ is then given by $c_{op}(x_t, \lambda_t)$. If $E_{\uparrow}^{t}$ is non-empty, we have $x_{t-1} < x_t$. In this case, the cost is given by $\beta(x_t - x_{t-1}) + c_{op}(x_t, \lambda_t)$. Using these observations, the cost of P can

be calculated by

$$\begin{aligned} c(P) &= c(0, x_1, \lambda_1) + \sum_{t=2}^{T} \Big(\overbrace{\beta \max\{0, x_t - x_{t-1}\}}^{c_{sw}(x_{t-1}, x_t)} + c_{op}(x_t, \lambda_t) \Big) \\ &\overset{(2.9)}{=} c(0, x_1, \lambda_1) + \sum_{t=2}^{T} \overbrace{\big(c_{sw}(x_{t-1}, x_t) + c_{op}(x_t, \lambda_t) \big)}^{c(x_{t-1}, x_t, \lambda_t)} \\ &\overset{(2.10)}{=} c(0, x_1, \lambda_1) + \sum_{t=2}^{T} c(x_{t-1}, x_t, \lambda_t) = \sum_{t=1}^{T} c(x_{t-1}, x_t, \lambda_t) \overset{(2.11)}{=} c(\mathcal{X}) \end{aligned}$$

Which shows that Φ is a cost-preserving map. □

Again, we can use this bijection to obtain our desired result: the correspondence between optimal schedules and shortest, reasonable paths.

Theorem 3.12. *There is a cost-preserving bijection between optimal schedules $\mathcal{X}$ for $\mathcal{I}$ and shortest, reasonable paths.*

Proof. By Lemma 3.11, we have a bijection Φ between reasonable paths P and schedules $\mathcal{X}$ obeying $c(P) = c\big(\Phi(P)\big)$. Thus, we have

$$c(P) \text{ minimal} \iff c\big(\Phi(P)\big) \text{ minimal}$$

and the claim follows. □

Naturally, common shortest path algorithms do not have any knowledge about our "reasonable" paths. What if we obtain a shortest path that coincidentally is not reasonable? This shall not turn out to be a problem, as the next corollary shows us.

Corollary 3.13. *Any shortest path P from $v_{0,0}$ to $v_{0,T\downarrow}$ can be transformed to an optimal schedule $\mathcal{X}$ for $\mathcal{I}$ with $c(\mathcal{X}) = c(P)$.*

Proof. Let P be a shortest path from $v_{0,0}$ to $v_{0,T\downarrow}$. By Proposition 3.8, the path P can be transformed to a reasonable path P' with $c(P') = c(P)$. In turn, P' corresponds to an optimal schedule $\mathcal{X}$ with $c(\mathcal{X}) = c(P')$ by Theorem 3.12. Thus, we have $c(\mathcal{X}) = c(P)$, and the claim follows. □

In the following, we give an algorithm based on our just verified constructions. Like in Section 3.1, we split our procedure into two subroutines.

SHORTEST_PATHS calculates the minimum costs of the nodes of our graph by following the topological sorting of the graph. Although we could search for an arbitrary shortest

path in our graph and transform it to an optimal schedule, as shown in Corollary 3.13, our algorithm only considers reasonable paths. For this, it consolidates the costs and predecessors of the nodes $v_{x,t\downarrow}$ and $v_{x,t\uparrow}$ in each time slot $t \in [T-1]$. Ultimately, it only keeps the relevant information for each node $v_{x,t\uparrow}$, namely the node's cost and best scheduling choice at time t. The information that would tell us the path between $v_{x,t\uparrow}$ and its best scheduling choice at time t in our graph is lost; however, this information is of no concern since we only want to consider reasonable paths which are already uniquely identified by their scheduling choices (see Lemma 3.11). This restriction to reasonable paths additionally improves the memory complexity of our algorithm as we can keep the size of the tables C and P down to mT instead of $2mT$. The function returns the minimum costs to all nodes $v_{x,t\uparrow}$ for $t \in [T-1]$ and $v_{x,T\downarrow}$. Moreover, it returns the predecessors of these nodes in respect to the best scheduling choice.

EXTRACT_SCHEDULE uses the predecessors calculated by SHORTEST_PATHS in order to obtain the sequence of nodes describing a shortest path; thereby, it calculates an optimal schedule for our problem instance.

Algorithm 2 Pseudo-linear optimal offline scheduling

```
1: function OPTIMAL_OFFLINE_SCHEDULING(m, T, Λ, β, f)
2:     (C, P) ← SHORTEST_PATHS(m, T, Λ, β, f)
3:     X ← EXTRACT_SCHEDULE(P, T)
4:     return X

5: function SHORTEST_PATHS(m, T, Λ, β, f)
       ▷ Allocate nodes' costs and predecessors tables
6:     let C[0...m, 1...T] and P[0...m, 1...T] be new tables
7:     P[m, 1] ← m and C[m, 1] ← c(0, m, λ_1)          ▷ Initialize first node in first layer
8:     for x ← m − 1 to 0 do      ▷ Initialize first layer (downward minimization steps)
9:         if C[x + 1, 1] < c(0, x, λ_1) then
10:            P[x, 1] ← P[x + 1, 1] and C[x, 1] ← C[x + 1, 1]
11:        else
12:            P[x, 1] ← x and C[x, 1] ← c(0, x, λ_1)
13:    for t ← 1 to T − 1 do                ▷ Iterative calculate costs and predecessors
14:        for x ← 1 to m do                                  ▷ Upward minimization steps
15:            if C[x − 1, t] + β < C[x, t] then
16:                P[x, t] ← P[x − 1, t] and C[x, t] ← C[x − 1, t] + β
17:        P[m, t + 1] ← m and C[m, t + 1] ← C[m, t] + c_op(m, λ_{t+1})
18:        for x ← m − 1 to 0 do                          ▷ Downward minimization steps
19:            if C[x + 1, t + 1] < C[x, t] + c_op(x, λ_{t+1}) then
20:                P[x, t + 1] ← P[x + 1, t + 1] and C[x, t + 1] ← C[x + 1, t + 1]
21:            else
22:                P[x, t + 1] ← x and C[x, t + 1] ← C[x, t] + c_op(x, λ_{t+1})
23:    return (C, P)

24: function EXTRACT_SCHEDULE(P, T)
25:    let X[T] be a new array
26:    X[T] ← P[0, T]                                  ▷ Get best choice for last time slot
27:    for t ← T − 1 to 1 do   ▷ Iteratively obtain a schedule by using the predecessors
28:        X[t] ← P[X[t + 1], t]
29:    return X
```

The correctness of Algorithm 2 directly follows from the correctness of the shortest path calculation for directed acyclic graphs (for a proof see [1]) and from Theorem 3.12.

For our runtime analysis, we take the same assumptions as done for Algorithm 1. Subroutine SHORTEST_PATHS needs $\Theta(m)$ iterations for its initialization and $\Theta(2Tm)$ steps for the

iterative calculation. In addition, EXTRACT_SCHEDULE needs $\Theta(T)$ iterations for its schedule retrieval. Hence, we receive a runtime of

$$\Theta(m + 2Tm + T) = \Theta(Tm)$$

The runtime is linear in the numeric value of m and T; however, it is exponential in the size of the input since, as we saw in (3.4), we only need $\log_2(m)$ bits to encode m. Hence, the algorithm is pseudo-linear, which is an improvement over the pseudo-polynomial runtime $\Theta(Tm^2)$ of Algorithm 1.

Our memory demand is defined by the size of the tables C and P. As both tables are of size $\Theta(Tm)$, we have a memory complexity of $\Theta(Tm)$; thus, our memory complexity does not change in comparison to Algorithm 1.

Although we have achieved a notable improvement compared to our initial approach, we shall not stop here. Our runtime is, strictly speaking, still exponential; thus, a large value of m may cause undesired long execution times. Our next goal is therefore the reduction of our runtime to subexponential complexity.

4 Approximative Offline Scheduling

Heretofore, we have derived two optimal offline algorithms for our scheduling problem; unfortunately, the runtime complexities of both algorithms are exponential in the input size of the number of servers m. Needless to say, we want to reduce this exponential runtime. For this, we must slightly loosen our aspirations, that is we move to approximative methods. Further, as we will see in the following sections, we need to assume that our convex operating costs function f is monotonically increasing; however, this restriction is of no great significance since in practice most operating costs functions fulfill this requirement. Any reference?

In this chapter, we will first modify our algorithm derived in Section 3.2 to obtain a 4-optimal offline algorithm with linear runtime complexity. As a next step, we will generalize our first approach to allow for arbitrary $1+\varepsilon$-approximations with TODO complexity.

4.1 A 4-Optimal Algorithm

Recall our algorithm and its corresponding graph G derived in Section 3.2. The algorithm's runtime complexity of $\Theta(Tm)$ is determined by the number of nodes and edges of G. Hence, as we desire to reduce our runtime complexity, we need to reduce the number of nodes and edges in G. In particular, we must get rid of the factor m. This factor is a consequence of the "height" of our graph, that is the number of nodes in each layer. Therefore, we have to "thin out" G by reducing its number of nodes in each layer.

As we saw in Equation (3.4), the size of our input $\mathcal{I}$ is given by $\mathcal{O}\big(T\log_2(m)+\log_2(\beta)\big)$. Consequently, in order to obtain a linear runtime complexity, we may reduce the graph's height from $m+1$ to a logarithmic one of $\log(m)$. Given this observation, it seems intuitive for a computer scientist to choose a logarithmic scale for the number of servers in each

layer. More formally, given a problem instance $\mathcal{I}$, we consider the following graph:

$$
\begin{aligned}
b &\coloneqq \lfloor \log_2(m) \rfloor \\
B &\coloneqq \{0, 2^0, 2^1, \ldots, 2^b, m\} \\
V &\coloneqq \left\{v_{x,t\downarrow} \mid x \in B, t \in [T]\right\} \cup \left\{v_{x,t\uparrow} \mid x \in B, t \in [T-1]\right\} \cup \{v_{0,0}\} \\
E_s &\coloneqq \left\{(v_{0,0}, v_{x,1\downarrow}) \mid x \in B\right\} \\
E_\downarrow &\coloneqq \left\{(v_{2^i,t\downarrow}, v_{2^{i-1},t\downarrow}) \mid i \in [b], t \in [T]\right\} \cup \left\{(v_{2^0,t\downarrow}, v_{0,t\downarrow}) \mid t \in [T]\right\} \cup \\
&\quad \left\{(v_{m,t\downarrow}, v_{2^b,t\downarrow}) \mid t \in [T]\right\} \\
E_\uparrow &\coloneqq \left\{(v_{2^{i-1},t\uparrow}, v_{2^i,t\uparrow}) \mid i \in [b], t \in [T-1]\right\} \cup \left\{(v_{0,t\uparrow}, v_{2^0,t\uparrow}) \mid t \in [T-1]\right\} \cup \\
&\quad \left\{(v_{2^b,t\uparrow}, v_{m,t\uparrow}) \mid t \in [T-1]\right\} \\
E_{\downarrow\uparrow} &\coloneqq \left\{(v_{x,t\downarrow}, v_{x,t\uparrow}) \mid x \in N, t \in [T-1]\right\} \\
E_{\uparrow\downarrow} &\coloneqq \left\{(v_{x,t\uparrow}, v_{x,t+1\downarrow}) \mid x \in N, t \in [T-1]\right\} \\
E &\coloneqq E_s \cup E_\downarrow \cup E_\uparrow \cup E_{\downarrow\uparrow} \cup E_{\uparrow\downarrow} \\
c_G(e) &\coloneqq \begin{cases} c(0, x, \lambda_1), & \text{if } e = (v_{0,0}, v_{x,1\downarrow}) \in E_s \\ c_{op}(x, \lambda_{t+1}), & \text{if } e = (v_{x,t\uparrow}, v_{x,t+1\downarrow}) \in E_{\uparrow\downarrow} \\ (x' - x)\beta, & \text{if } e = (v_{x,t\uparrow}, v_{x',t\uparrow}) \in E_\uparrow \\ 0, & \text{if } e \in (E_\downarrow \cup E_{\downarrow\uparrow}) \end{cases} \\
G &\coloneqq (V, E, c_G)
\end{aligned}
$$

A more appealing, graphical representation can be found in the following figure.

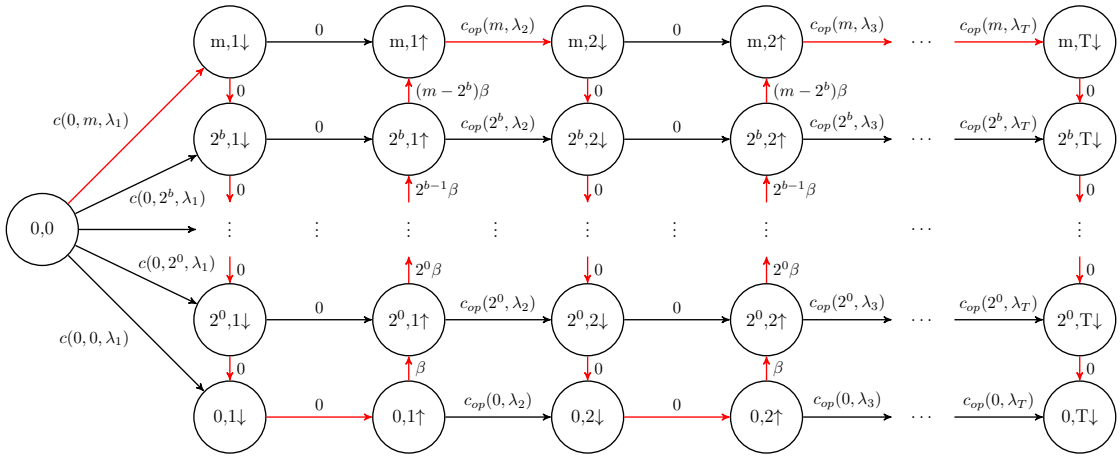


Figure 4.1: Graph for a linear, 4-optimal offline algorithm; the path of the topological sorting is highlighted in red.

4.2 A $1+\varepsilon$-Optimal Algorithm

List of Algorithms

Bibliography

[1] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. Third. The MIT Press, 2009. Chap. 24.2.

[2] Minghong Lin, Adam Wierman, Lachlan L. H. Andrew, and Eno Thereska. "Dynamic right-sizing for power-propotional data centers". In: *IEEE/ACM Transactions on Networking (TON)* 21 (2013), pp. 1378–1391.

Notation

Below, we give an overview of the definitions and conventions commonly referred to in our paper.

Input

- $m \in \mathbb{N}$... number of homogeneous servers
- $T \in \mathbb{N}$... number of time slots
- $\lambda_1, \ldots, \lambda_T \in [0, m]$... arrival rates
- $\Lambda \coloneqq (\lambda_1, \ldots, \lambda_T)$... sequence of arrival rates
- $\beta \in \mathbb{R}_{\geq 0}$... switching costs of a server
- $f : [0, 1] \to \mathbb{R}$... convex operating costs function of a server
- $\mathcal{I} \coloneqq (m, T, \Lambda, \beta, f)$... input of a problem instance

Problem statement

- $s_{i,t} \in \{0, 1\}$... state of server i at time t, i.e. sleeping (0) or active(1)
- $S_i \coloneqq (s_{i,1}, \ldots, s_{i,T})$... sequence of states for server i
- $\lambda_{i,t} \in [0, 1]$... assigned load for server i at time t
- $L_i \coloneqq (\lambda_{i,1}, \ldots, \lambda_{i,T})$... sequence of assigned loads for server i
- $\mathcal{S} \coloneqq (S_1, \ldots, S_m)$... sequence of all state changes
- $\mathcal{L} \coloneqq (L_1, \ldots, L_m)$... sequence of all assigned loads
- $\Sigma \coloneqq (\mathcal{S}, \mathcal{L})$... schedule for a problem instance $\mathcal{I}$

Miscellaneous

- $x_t \in \{0, \dots, m\}$ … number of active servers at time t
- $\mathcal{X} \coloneqq (x_1, \dots, x_T)$ … sequence of number of active servers

Conventions

- $\lambda_t = 0$ for all $t \notin [T]$, i.e. there is no load before and after the scheduling process
- $s_{i,t} = 0$ for all $t \notin [T]$, i.e. all servers are powered down before and after the scheduling process