

# Algorithms for Dynamic Right-Sizing in Data Centers

Kevin Kappelmann

*Chair for Theoretical Computer Science,  
Technical University of Munich*

May 25, 2017

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Model description . . . . .	1
1.2	Problem statement . . . . .	1
<b>2</b>	<b>Preliminaries</b>	<b>2</b>
<b>3</b>	<b>Optimal offline scheduling</b>	<b>7</b>
3.1	A pseudo-polynomial algorithm . . . . .	7
3.2	A pseudo-linear algorithm . . . . .	10
<b>4</b>	<b>Approximative offline scheduling</b>	<b>14</b>
4.1	A polynomial 4-approximation algorithm . . . . .	14

# 1 Introduction

TODO: Hardware prices vs. energy costs in data centres, related work and purpose of this paper (offline algorithm, approximation algorithm, ...).

## 1.1 Model description

We want to address the issue of above-mentioned ever-growing energy consumption by examining a scheduling problem that commonly arises in data centres. More specifically, we consider a model consisting of a fixed amount of homogeneous servers denoted by  $m \in \mathbb{N}$  and a fixed amount of time slots denoted by  $T \in \mathbb{N}$ . In turn, each server possesses two power states, i.e. each server 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  $\lambda_t$ , i.e. the amount of expected load to process in time slot  $t$ . We expect the arrival rates to be normalised 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  $f : [0, 1] \rightarrow \mathbb{R}$ , as well as its (*power state*) *switching costs*, specified by  $\beta \in \mathbb{R}_{\geq 0}$ . The operating costs  $f$  may not exclusively consider 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,  $\beta$  may also allow for delay costs, wear and tear costs or the like. [?]

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 assume convexity for  $f$ . This may seem like a notable restriction at first, but it indeed captures the behaviour of most modern server models. Since we are dealing with homogeneous servers,  $f$  and  $\beta$  are the same for all machines.

For convenience, we assume all machines sleeping at time  $t = 0$  and force all machines to sleep after the scheduling process, i.e. at times  $t > T$ . Consequently, any server must power down exactly as many times as it powers on. This allows us to consolidate power up and power down costs into  $\beta$  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  $\beta := \beta'_{\uparrow} := \beta_{\uparrow} + \beta_{\downarrow}$  and  $\beta'_{\downarrow} := 0$ . Similarly, we assume 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 above definitions, we can define the input of our model by setting  $\mathcal{I} := (m, T, \Lambda, \beta, f)$  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 minimise the sum of incurred costs while warranting that we are processing 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  $L_i$ ; that is

$$\begin{aligned} S_i &:= (s_{i,1}, \dots, s_{i,T}) \in \{0, 1\}^T \\ L_i &:= (\lambda_{i,1}, \dots, \lambda_{i,T}) \in [0, 1]^T \end{aligned}$$

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 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:

$$\begin{aligned} \mathcal{S} &:= (S_1, \dots, S_m) \\ \mathcal{L} &:= (L_1, \dots, L_m) \end{aligned}$$

We will subsequently call a pair  $\Sigma := (\mathcal{S}, \mathcal{L})$  a *schedule*. Finally, we are ready to define our problem statement.

Given an input  $\mathcal{I}$ , our goal is to find a schedule  $\Sigma$  that satisfies the following optimisation:

$$\begin{aligned} \text{minimise} \quad & c(\Sigma) := \underbrace{\sum_{t=1}^T \sum_{i=1}^m (f(\lambda_{i,t}) s_{i,t})}_{\text{operating costs}} + \beta \underbrace{\sum_{t=1}^T \sum_{i=1}^m \min\{0, s_{i,t} - s_{i,t-1}\}}_{\text{switching costs}} \quad (1) \\ \text{subject to} \quad & \sum_{i=1}^m (\lambda_{i,t} s_{i,t}) = \lambda_t, \quad \forall t \in [T] \quad (2) \end{aligned}$$

We call a schedule *feasible* if it satisfies (2) and *optimal* if it satisfies (1) and (2).

## 2 Preliminaries

In this section, we conduct the preparatory work that will lay the foundations for our algorithms. For this, we analyse the structure of feasible schedules concerning their cost efficiency in order to find characteristics of optimal schedules; these characteristics will then allow us to greatly simplify our optimisation conditions.

We begin by examining the state sequences of feasible schedules. As we are considering homogeneous servers, we do not care which exact servers process the given work loads. Rather we only care about the amount 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 machine in return; we could just keep the first machine powered on, 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  $\Sigma$ , there exists a feasible schedule  $\Sigma'$  such that*

$$(i) \quad c(\Sigma') \leq c(\Sigma) \text{ and}$$

(ii)  $\Sigma'$  never powers on and shuts down servers at the same time slot, i.e.  $\Sigma'$  satisfies the following formula:

$$\forall t \in [T] \left[ (\forall i \in [m] (s_{i,t} - s_{i,t-1} \geq 0)) \vee (\forall i \in [m] (s_{i,t} - s_{i,t-1} \leq 0)) \right] \quad (3)$$

*Proof.* Let  $\Sigma = (\mathcal{S}, \mathcal{L})$  be a feasible schedule for  $\mathcal{I}$ . We give a procedure that repeatedly modifies  $\Sigma$  such that it satisfies (3) and reduces or retains its costs.

Let  $t \in [T]$  be the first time slot falsifying (3). If there does not exist such a time slot, we are done. Otherwise, we can obtain machines  $i, j \in [m]$  such that  $s_{i,t} - s_{i,t-1} = 1$  and  $s_{j,t} - s_{j,t-1} = -1$ , i.e. server  $i$  powers on at time  $t$  and server  $j$  powers off. Without loss of generality, we may assume  $i < j$ .

First, 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 (3) for  $t = 1$ . Thus, we may assume  $t > 1$ .

Now consider the state sequences of server  $i$  and  $j$ :

$$\begin{aligned} S_i &= (s_{i,1}, \dots, s_{i,t-1} = 0, s_{i,t} = 1, \dots, s_{i,T}) \\ S_j &= (s_{j,1}, \dots, s_{j,t-1} = 1, s_{j,t} = 0, \dots, s_{j,T}) \end{aligned}$$

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

$$\begin{aligned} S'_i &:= (s_{i,1}, \dots, s_{i,t-1} = 0, s_{j,t} = 0, \dots, s_{j,T}) \\ S'_j &:= (s_{j,1}, \dots, s_{j,t-1} = 1, s_{i,t} = 1, \dots, s_{i,T}) \end{aligned}$$

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

$$\begin{aligned} L'_i &:= (\lambda_{i,1}, \dots, \lambda_{i,t-1}, \lambda_{j,t}, \dots, \lambda_{j,T}) \\ L'_j &:= (\lambda_{j,1}, \dots, \lambda_{j,t-1}, \lambda_{i,t}, \dots, \lambda_{i,T}) \end{aligned}$$

Finally, we construct a new schedule  $\Sigma' := (\mathcal{S}', \mathcal{L}')$  given by

$$\begin{aligned} \mathcal{S}' &:= (S_1, \dots, S_{i-1}, S'_i, S_{i+1}, \dots, S_{j-1}, S'_j, S_{j+1}, \dots, S_T) \\ \mathcal{L}' &:= (L_1, \dots, L_{i-1}, L'_i, L_{i+1}, \dots, L_{j-1}, L'_j, L_{j+1}, \dots, L_T) \end{aligned}$$

We want to verify that  $\Sigma'$  is a feasible schedule, that is  $\Sigma'$  satisfies (2). For time slots  $< t$ , the schedules  $\Sigma'$  and  $\Sigma$  still coincide. For time slots  $\geq t$ , we only changed the order of summation in (2). Thus,  $\Sigma'$  is feasible.

Further,  $\Sigma$  and  $\Sigma'$  coincide in their operating costs; however  $\mathcal{S}'$  minimises 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 (3) at time  $t$ . Hence, by repeating described process on  $\Sigma'$ , we obtain a terminating procedure that returns a schedule satisfying the conditions.  $\square$

Next, we want to 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

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

As we assume all machines sleeping at times  $t \notin [T]$ , we have  $x_t = 0$  for  $t \notin [T]$ .

Our next proposition will require the use of Jensen's inequality, a well-known and frequently used analytic result found by Johan Jensen in 1906.

**Lemma 2.2** (Jensen's inequality). *Let  $f : \mathbb{R} \rightarrow \mathbb{R}$  be a convex function,  $n \in \mathbb{N}$ ,  $\lambda_1, \dots, \lambda_n \in \mathbb{R}$ , and  $\mu_1, \dots, \mu_n \in [0, 1]$  satisfying  $\sum_{i=1}^n \mu_i = 1$ . Then the following inequality holds:*

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

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

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

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

- 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  $\sum_{i=1}^{n+1} \mu_i = 1$ , at least one  $\mu_i$  must be positive. Without loss of generality, we may assume  $\mu_1 > 0$ .

$$f\left(\sum_{i=1}^{n+1} \mu_i \lambda_i\right) = f\left(\mu_1 \lambda_1 + \sum_{i=2}^{n+1} \mu_i \lambda_i\right) \stackrel{\mu_1 \neq 0}{=} f\left(\mu_1 \lambda_1 + (1 - \mu_1) \sum_{i=2}^{n+1} \frac{\mu_i \lambda_i}{1 - \mu_1}\right)$$

As  $f$  is convex, we have

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

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

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

We combine our steps and obtain

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

The following proposition poses the cornerstone of our subsequent works. We want to establish an optimal scheduling strategy given a fixed amount of active servers. It turns out that an even load distribution seems a very desirable strategy.

**Proposition 2.3** (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  $\lambda_t/x_t$ .*

*Proof.* Let  $\Sigma$  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  $A := \{i \in [m] \mid s_{i,t} = 1\}$ . Consider the operating costs of  $\Sigma$  at time  $t$  given by

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

Since  $\Sigma$  is feasible (see constraint (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  $\lambda_t$  for  $i \in A$  such that

$$\sum_{i=1}^{x_t} \mu_i = 1 \quad \text{and} \quad \sum_{i \in A} f(\lambda_{i,t}) = \sum_{i=1}^{x_t} f(\mu_i \lambda_t) \quad (4)$$

In particular, we have

$$\sum_{i=1}^{x_t} \mu_i \lambda_t = \lambda_t \quad (5)$$

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

$$\sum_{i=1}^{x_t} f\left(\frac{\lambda_t}{x_t}\right) = x_t f\left(\frac{\lambda_t}{x_t}\right) \stackrel{(5)}{=} x_t f\left(\sum_{i=1}^{x_t} \frac{\mu_i \lambda_t}{x_t}\right)$$

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

$$x_t f\left(\frac{\lambda_t}{x_t}\right) \leq x_t \sum_{i=1}^{x_t} \frac{1}{x_t} f(\mu_i \lambda_t) = \frac{x_t}{x_t} \sum_{i=1}^{x_t} f(\mu_i \lambda_t) = \sum_{i=1}^{x_t} f(\mu_i \lambda_t) \stackrel{(4)}{=} \sum_{i \in A} f(\lambda_{i,t})$$

Thus, the operating costs of  $\Sigma^*$  give a lower bound for the operating costs of  $\Sigma$ , and the claim follows.  $\square$

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

**Corollary 2.4.** *Given a problem instance  $\mathcal{I}$ , there exists an optimal schedule  $\Sigma^*$  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  $\Sigma$  in each time slot. Setting  $\Sigma^* := (\mathcal{S}, \mathcal{L}^*)$  we have  $c(\Sigma^*) = c(\Sigma)$  by Proposition 2.3. The claim follows.  $\square$

As a result of Corollary 2.4, 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 2.5.** *Given a problem instance  $\mathcal{I}$ , there exists an optimal schedule  $\Sigma^*$  that evenly distributes its arrival rates to its active servers in each time slot and further satisfies (3).*

*Proof.* By Corollary 2.4, we obtain an optimal schedule  $\Sigma$  that evenly distributes its arrival rates to its active servers. Applying the procedure given in Proposition 2.1 to  $\Sigma$  yields  $\Sigma^*$  that further satisfies (3).  $\square$

Theorem 2.5 allows us to subsequently identify an optimal schedule by its sequence of active servers  $\mathcal{X}$  and thereby to simplify our optimisation conditions (1) and (2). For this, given a problem instance  $\mathcal{I}$ , we define the operating costs function  $c_{op}(x, \lambda)$  that describes the costs incurred by evenly distributing  $\lambda$  on  $x$  active servers using  $f$ :

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

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

$$c_{sw}(x_{t-1}, x_t) := \beta \max\{0, x_t - x_{t-1}\}$$

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

$$c(x_{t-1}, x_t, \lambda_t) := c_{op}(x_t, \lambda_t) + c_{sw}(x_{t-1}, x_t)$$

The optimisation conditions for a schedule now simplify to one single minimisation:

$$\text{minimise } c(\mathcal{X}) := \sum_{t=1}^T c(x_{t-1}, x_t, \lambda_t) \tag{6}$$

We subsequently call a schedule  $\mathcal{X}$  *optimal* if it satisfies (6).



### 3 Optimal offline scheduling

In this section, we derive two optimal offline algorithms based on our preliminary work. First, we reduce our problem specified by  $\mathcal{I}$  to a shortest path problem of a level structured graph  $G$ . We then use a dynamic programming approach 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 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:

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

For any possible amount 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  $\lambda_t$  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 costs of taking the edge correspond to switching from  $x$  to  $x'$  machines and processing the load  $\lambda_{t+1}$  with  $x'$  machines. A graphical representation can be found in the following figure.

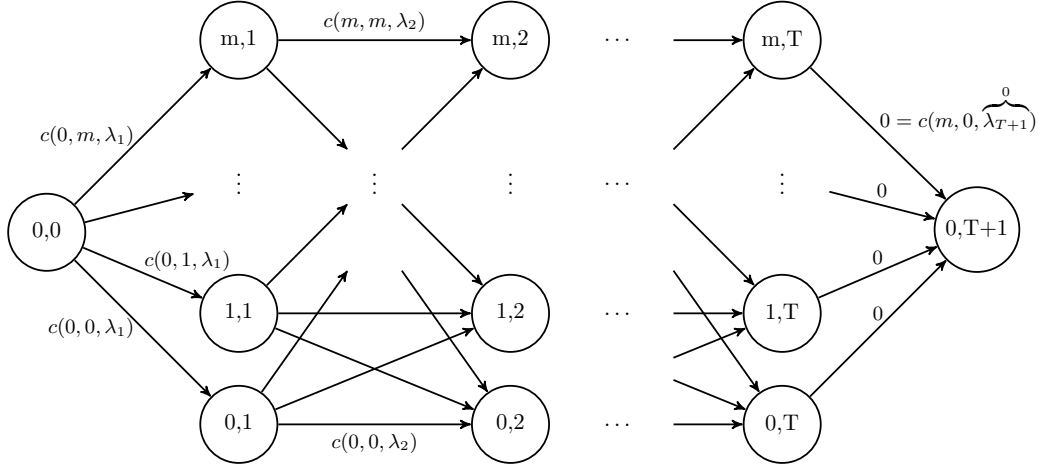


Figure 1: Level structured graph for a pseudo-polynomial, optimal offline algorithm

The costs of a path  $P = (v_{x_0,T_0}, v_{x_1,T_0+1}, \dots, v_{x_n,T_0+n})$  with length  $n$  in our graph (where

$x_t \in \{0, \dots, m\}$ ,  $T_0 \in \{0, \dots, T\}$  and  $T_0 + n \leq T + 1$ ) are thus given by

$$c(P) := \sum_{t=1}^n c(x_{t-1}, x_t, \lambda_{T_0+t})$$

In particular, for a path  $P = (v_{0,0}, v_{x_1,1}, \dots, v_{x_T,T}, v_{0,T+1})$  from our start node  $v_{0,0}$  to our end node  $v_{0,T+1}$  we have

$$c(P) = c(0, x_1, \lambda_1) + \sum_{t=2}^T c(x_{t-1}, x_t, \lambda_t) + \underbrace{c(x_T, 0, 0)}_{=0} = c(0, x_1, \lambda_1) + \sum_{t=2}^T c(x_{t-1}, x_t, \lambda_t) \quad (7)$$

Note that the costs of such a path directly correspond to those of a schedule  $\mathcal{X}$  (see (6)). Any shortest path from  $v_{0,0}$  to  $v_{0,T+1}$  is thus forced to minimise the costs of the corresponding schedule. Needless to say, this demands for a proof of correctness.

**Lemma 3.1.** *Let  $\mathcal{X}$  be the set of all schedules  $\mathcal{X}$  for  $\mathcal{I}$ , and let  $\mathcal{P}$  the set of all paths from  $v_{0,0}$  to  $v_{0,T+1}$ . The map*

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

*is a bijection with inverse map*

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

*satisfying  $c(\mathcal{X}) = c(\Phi(\mathcal{X}))$ .*

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

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

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

$$c(\mathcal{X}) \stackrel{(6)}{=} \sum_{t=1}^T c(x_{t-1}, x_t, \lambda_t) = \underbrace{c(x_0, x_1, \lambda_1)}_{=c(0, x_1, \lambda_1)} + \sum_{t=2}^T c(x_{t-1}, x_t, \lambda_t) \stackrel{(7)}{=} c(P)$$

Which shows that  $\Phi$  and, as a consequence,  $\Psi$  are cost-preserving maps. The claim follows.  $\square$

**Theorem 3.2.** *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 3.1, we have a bijection  $\Phi$  between schedules  $\mathcal{X}$  and paths from  $v_{0,0}$  to  $v_{0,T+1}$  obeying  $c(\mathcal{X}) = c(\Phi(\mathcal{X}))$ . Thus, we have

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

and the claim follows.  $\square$

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

SHORTEST\_PATHS uses a dynamic programming approach similar to the well-known Bellman-Ford algorithm. 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 and thereby a schedule for our problem instance.

---

**Algorithm 1** Pseudo-polynomial optimal offline scheduling

---

```

1: function OPTIMAL_OFFLINE_SCHEDULING( $m, T, \Lambda, \beta, f$ )
2:    $(C, P) \leftarrow$  SHORTEST_PATHS( $m, T, \Lambda, \beta, f$ )
3:    $\mathcal{X} \leftarrow$  EXTRACT_SCHEDULE( $P, T$ )
4:   return  $\mathcal{X}$ 

5: function SHORTEST_PATHS( $m, T, \Lambda, \beta, f$ )
6:   let  $C[T+1, m]$  and  $P[T+1, m]$  be new arrays     $\triangleright$  Costs and predecessors of nodes
7:   for  $x \leftarrow 0$  to  $m$  do     $\triangleright$  Initialisation
8:   |    $C[1, x] \leftarrow c(0, x, \lambda_1)$ 
9:   |    $P[1, x] \leftarrow 0$ 
10:  for  $t \leftarrow 2$  to  $T$  do     $\triangleright$  Iterative calculation of costs and predecessors
11:  |   for  $x' \leftarrow 0$  to  $m$  do
12:  |   |    $P[t, x'] \leftarrow \arg \min_{x \in \{0, \dots, m\}} \{C[t-1, x] + c(x, x', \lambda_t)\}$      $\triangleright$  Find best preceding choice
13:  |   |    $C[t, x'] \leftarrow C[t-1, P[t, x']] + c(P[t, x'], x', \lambda_t)$ 
14:  |    $P[T+1, 0] \leftarrow \arg \min_{x \in \{0, \dots, m\}} \{C[T, x]\}$      $\triangleright$  Find best choice for last time slot
15:  |    $C[T+1, 0] \leftarrow C[T, P[T+1, 0]]$ 
16:  return  $(C, P)$ 

17: function EXTRACT_SCHEDULE( $P, T$ )
18:  let  $\mathcal{X}[T]$  be a new array
19:   $\mathcal{X}[T] \leftarrow P[T+1, 0]$      $\triangleright$  Get best choice for last time slot
20:  for  $t \leftarrow T-1$  to  $1$  do     $\triangleright$  Iteratively obtain a schedule by using the predecessors
21:  |    $\mathcal{X}[t] \leftarrow P[t+1, \mathcal{X}[t+1]]$ 
22:  return  $\mathcal{X}$ 

```

---

The correctness of Algorithm 1 directly follows from the correctness of the Bellman-Ford algorithm and Theorem 3.2.

Naturally, we are interested in our algorithm's time and memory complexity. Subroutine SHORTEST\_PATHS needs  $\Theta(m)$  iterations for its initialisation,  $\Theta(Tm^2)$  steps for the iterative calculation, and  $\Theta(m)$  steps for its final minimisation search. In addition, EXTRACT\_SCHEDULE needs  $\Theta(T)$  iterations for its schedule retrieval. Thus, we receive a run-time of

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

The runtime is polynomial in the numeric value of the input; however, as we just need  $\log_2(m)$  bits to encode  $m$ , it is exponential in the length of the input. Hence, the algorithm is pseudo-polynomial.

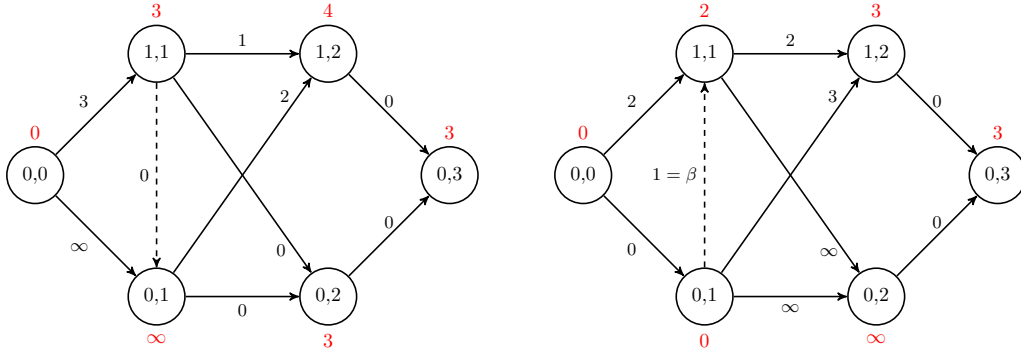
Our memory demand is defined by the size of the arrays  $C$  and  $P$ . As both arrays are of size  $\Theta(Tm)$ , we have a memory complexity of  $\Theta(Tm)$ .

### 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 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 node  $v_{x,t}$  denotes the state of distributing the arrival rate  $\lambda_t$  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  $\lambda_t$ . In particular, the cost of the end node  $v_{0,T+1}$  tell 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 3.3.** 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 for 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 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 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.

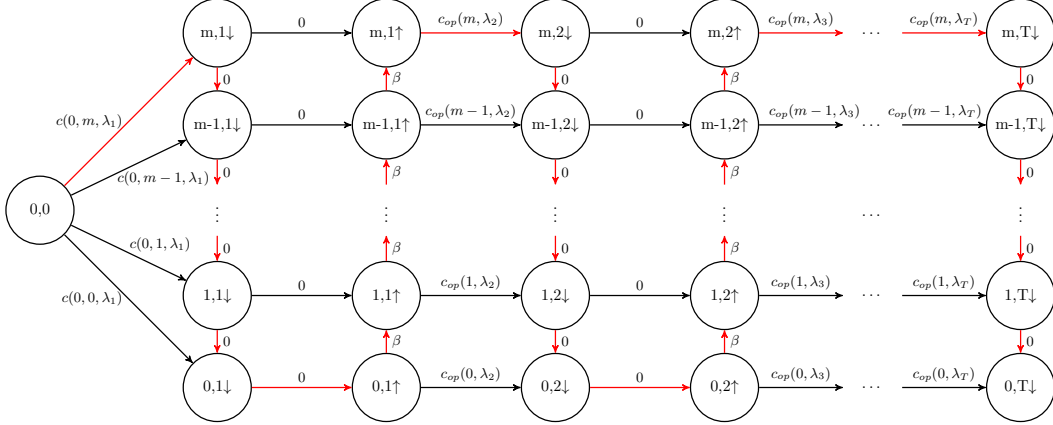


Figure 3: Graph for a pseudo-linear, optimal offline algorithm; the path of the topological sorting is highlighted in red.

For any possible amount 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  $\lambda_t$  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 initialisation 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  $\lambda_{t+1}$  with  $x$  active servers.

After any time step from  $t-1$  to  $t$  that incurs operating costs, we have a minimisation step in our graph. For this, we first move down the chain  $v_{m,t\downarrow}, v_{m-1,t\downarrow}, \dots, v_{0,t\downarrow}$  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  $v_{0,t\uparrow}, v_{1,t\uparrow}, \dots, v_{m,t\uparrow}$  using edges from  $E_{\uparrow}$  with cost  $\beta$ . 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 minimisation step is the key to our runtime improvement. It facilitates the determination of the best predecessor of a state  $v_{x,t\downarrow}$  because 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  $\lambda_t$  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  $v_{x+1,t\downarrow}$ .

As one can see in Figure 3, we stretched our graph but at the same time also greatly reduced the amount of edges compared to our initial approach in section 3.1. By following the coloured 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  $\lambda_T$  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:

$$P := (v_{0,0}, v_{1,1\downarrow}, v_{0,1\downarrow}, v_{0,1\uparrow}, v_{1,1\uparrow}, v_{2,1\uparrow}, v_{2,2\downarrow}, \dots, v_{0,T\downarrow})$$

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 behaviour cannot even be represented using our schedule notation  $\mathcal{X}$  and optimisation condition (6). We can, however, modify  $P$  to represent a more reasonable sequence, that is we set

$$P' := (v_{0,0}, v_{1,1\downarrow}, v_{1,1\uparrow}, v_{2,1\uparrow}, v_{2,2\downarrow}, \dots, v_{0,T\downarrow})$$

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

**Definition 3.4** (reasonable paths). Let  $P$  be a path from  $v_{0,0}$  to  $v_{0,T\downarrow}$ . For any time slot  $t \in [T]$ , let  $E_{\downarrow}^t$  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  $E_{\uparrow}^t$  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

$$\forall t \in [T-1] (E_{\downarrow}^t = \emptyset \vee E_{\uparrow}^t = \emptyset) \quad (8)$$

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

**Proposition 3.5.** *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 (8) and reduces or retains its costs.

Let  $t \in [T-1]$  be the first time slot falsifying (8). If there does not exist such a time slot, we are done. Otherwise, let  $E_{\downarrow}^t$  and  $E_{\uparrow}^t$  be its sets of edges as defined in Definition 3.4. Since  $P$  falsifies (8) at time  $t$ , both  $E_{\downarrow}^t$  as well as  $E_{\uparrow}^t$  must be non-empty. Thus, we can obtain the “maximum” nodes involved in these sets.

$$\begin{aligned} x_s &:= \max\{x \in [m] \mid (v_{x,t\downarrow}, v_{x-1,t\downarrow}) \in E_{\downarrow}^t\} \\ x_e &:= \max\{x \in [m] \mid (v_{x-1,t\uparrow}, v_{x,t\uparrow}) \in E_{\uparrow}^t\} \end{aligned}$$

Next, consider the subpath  $S = (v_{x_s,t\downarrow}, \dots, v_{x_e,t\uparrow})$  of  $P$ . Note that the subpath in particular uses all edges from  $E_{\downarrow}^t$  and  $E_{\uparrow}^t$ . Naturally, a most cost-efficient path  $S'$  from node  $v_{x_s,t\downarrow}$  to  $v_{x_e,t\uparrow}$  minimises the number of edges  $(v_{x,t\uparrow}, v_{x+1,t\uparrow})$  since each of these edges incurs cost  $\beta \geq 0$ . For this, 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  $S'$ :

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

In each case,  $S'$  uses edges from at most one of the sets  $E_{\downarrow}^t$  and  $E_{\uparrow}^t$ . Thus, by replacing the subpath  $S$  of  $P$  with  $S'$ , we obtain a new schedule  $P'$  that satisfies (8) 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  $\beta$  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.  $\square$

**Lemma 3.6.** *Let  $\mathcal{X}$  be the set of all schedules  $\mathcal{X}$  for  $\mathcal{I}$ , and let  $\mathcal{P}$  the set of all reasonable paths. There exists a bijective map*

$$\Phi : \mathcal{X} \rightarrow \mathcal{P}$$

*with inverse*

$$\Psi : \mathcal{P} \rightarrow \mathcal{X}$$

*satisfying  $c(\mathcal{X}) = c(\Phi(\mathcal{X}))$ .*

*Proof.* TODO

Which shows that  $\Phi$  and, as a consequence,  $\Psi$  are cost-preserving maps, and the claim follows.  $\square$

**Theorem 3.7.** *There is a cost-preserving bijection between optimal schedules  $\mathcal{X}$  for  $\mathcal{I}$  and shortest, reasonable paths.*

*Proof.* By Lemma 3.6, we have a bijection  $\Phi$  between schedules  $\mathcal{X}$  and reasonable paths beying  $c(\mathcal{X}) = c(\Phi(\mathcal{X}))$ . Thus, we have

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

and the claim follows.  $\square$

**Corollary 3.8.** *Any shortest path  $P$  from  $v_{0,0}$  to  $v_{0,T+1}$  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  $v_{0,T+1}$ . By Proposition 3.5, 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 3.7. Thus, we have  $c(\mathcal{X}) = c(P)$  and, the claim follows.  $\square$

## 4 Approximative offline scheduling

### 4.1 A polynomial 4-approximation algorithm



## Appendix

Below, we give an overview of just given 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
  - $\lambda_1, \dots, \lambda_T \in [0, m]$ ... arrival rates
  - $\Lambda := (\lambda_1, \dots, \lambda_T)$ ... sequence of arrival rates
  - $\beta \in \mathbb{R}_{\geq 0}$ ... switching costs of a server
  - $f : [0, 1] \rightarrow \mathbb{R}$ ... convex operating costs function of a server
  - $\mathcal{I} := (m, T, \Lambda, \beta, f)$ ... 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)
  - $S_i := (s_{i,1}, \dots, s_{i,T})$ ... sequence of states for server  $i$
  - $\lambda_{i,t} \in [0, 1]$ ... assigned load for server  $i$  at time  $t$
  - $L_i := (\lambda_{i,1}, \dots, \lambda_{i,T})$ ... sequence of assigned loads for server  $i$
  - $\mathcal{S} := (S_1, \dots, S_m)$ ... sequence of all state changes
  - $\mathcal{L} := (L_1, \dots, L_m)$ ... sequence of all assigned loads
  - $\Sigma := (\mathcal{S}, \mathcal{L})$ ... schedule for a problem instance  $\mathcal{I}$
- Miscellaneous:
  - $x_t \in \{0, \dots, m\}$ ... number of active servers at time  $t$
  - $\mathcal{X} := (x_1, \dots, x_T)$ ... 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