

Polylog-Competitive Algorithms for Dynamic Balanced Graph Partitioning for Ring Demands

Harald Räcke¹, Stefan Schmid², and Ruslan Zabrodin¹

¹*Technical University of Munich*

²*Technical University of Berlin*

The performance of many large-scale and data-intensive distributed systems critically depends on the capacity of the interconnecting network. This paper is motivated by the vision of self-adjusting infrastructures whose resources can be adjusted according to the workload they currently serve, in a demand-aware manner. Such dynamic adjustments can be exploited to improve network utilization and hence performance, by dynamically moving frequently interacting communication partners closer, e.g., collocating them in the same server or datacenter rack.

In particular, we revisit the online balanced graph partitioning problem which captures the fundamental tradeoff between the benefits and costs of dynamically collocating communication partners. The demand is modelled as a sequence σ (revealed in an online manner) of communication requests between n processes, each of which is running on one of the ℓ servers. Each server has capacity $k = n/\ell$, hence, the processes have to be scheduled in a balanced manner across the servers. A request incurs cost 1, if the requested processes are located on different servers, otherwise the cost is 0. A process can be migrated to a different server at cost 1.

This paper presents the first online algorithm for online balanced graph partitioning achieving a polylogarithmic competitive ratio for the fundamental case of ring communication patterns. Specifically, our main contribution is a $O(\log^3 n)$ -competitive randomized online algorithm for this problem. We further present a randomized online algorithm which is $O(\log^2 n)$ -competitive when compared to a static optimal solution. Our two results rely on different algorithms and techniques and hence are of independent interest.

1. Introduction

Data-centric applications, including distributed machine learning, batch processing, scale-out databases, or streaming, produce a significant amount of communication traffic and their performance often critically depends on the underlying datacenter network [1]. In particular, large flows (also known as elephant flows) may require significant network resources if communicated across multiple hops, resulting in a high “bandwidth tax” and consuming valuable resources which would otherwise be available for additional flows [2, 3].

An intriguing approach to reduce communication overheads and make a more efficient use of the available bandwidth capacity, is to leverage the resource allocation flexibilities available in modern distributed systems (e.g., using virtualization), and render the infrastructures self-adjusting: by collocating two processes which currently exchange much data on the same server or datacenter rack, in a demand-aware manner, communication can be kept local and resources saved. When and how to collocate processes however is an algorithmically challenging problem, as it introduces a tradeoff: as migrating a process to different server comes with overheads, it should not be performed too frequently and only when the migration cost can be amortized by an improved communication later. Devising good migration strategies is particularly difficult in the realm of online algorithms and competitive analysis, where the demand is not known ahead of time.

The fundamental algorithmic problem underlying such self-adjusting infrastructures is known as dynamic balanced graph (re-)partitioning and has recently been studied intensively [4–11], see also the recent SIGACT News article on the problem [12]. In its basic form, the demand is modelled as a sequence σ (revealed in an online manner) of communication requests between n processes, each of which is running (i.e., scheduled) on one of the ℓ servers. Each server has capacity $k = n/\ell$, hence, the processes have to be scheduled in a balanced manner across the servers. A request incurs cost 1, if both requested processes are located on different servers, otherwise the cost is 0. A process can be migrated to a different server at cost 1. The goal is to design online algorithms which do not know σ ahead of time, yet, they are competitive against an optimal offline algorithm with complete knowledge of the demand.

Unfortunately, deterministic online algorithms cannot achieve a low competitive ratio: Avin et al. [8] (DISC 2016) presented a lower bound of $\Omega(k)$ for any deterministic algorithm for this problem, even if the communication requests are sampled from a ring graph, and even in a resource augmentation model. Accordingly, most related work revolves around deterministic algorithms with polynomial competitive ratios [5–8, 11]. Hardly anything is known about the competitive ratio achievable by randomized online algorithms, except for a “learning variant” introduced by Henzinger et al. in [10] (SIGMETRICS 2019) and later studied by Henzinger et al. in [4] (SODA 2021): in this learning model, it is guaranteed that the communication requests can be perfectly partitioned, that is, the requests in σ are drawn from a graph whose connected components can be assigned to servers such that no connected component needs to be distributed across multiple servers. For this learning variant, the authors presented a polynomial-time randomized algorithm achieving a polylogarithmic competitive ratio of $O(\log \ell + \log k)$

which is asymptotically optimal. Unfortunately, however, these results on the learning variant are not applicable to communication patterns which do not perfectly fit into the servers, but which continuously require inter-server communication and/or migrations.

Motivated by this gap, we in this paper study the design of randomized online algorithms for the balanced graph partitioning problem. We consider two models:

- *Static*: The performance of the online algorithm is compared to an optimal static solution (for the given demand). This static model can be seen as a natural generalization of the learning variant discussed in prior work: while in prior work, it is assumed that a static solution exists which does not accrue any communication cost, in our static model, we do not make such an assumption on the communication pattern.
- *Dynamic*: The performance of the online algorithm is compared to an optimal dynamic solution, which may also perform migrations over time.

As a first step, we consider a most fundamental setting where the demand σ is chosen from a ring communication pattern. This is not only interesting because a ring pattern cannot be solved by existing algorithms designed for the learning variant (the ring is a large connected component which does not fit into a server) and because of the high lower bound for deterministic algorithms mentioned above, but also because of its practical relevance: machine learning workloads often exhibit ring like traffic patterns [3, 13–15].

1.1. Our Contributions

We present the first online algorithm for online balanced graph partitioning achieving a polylogarithmic competitive ratio for the fundamental case of ring communication patterns. We first present a randomized online algorithm which is $O(\log^2 n)$ -competitive when compared to an optimal static solution; this ratio is strict, i.e., without any additional additive terms. Our second and main contribution is a $O(\log^3 n)$ -competitive randomized online algorithm for this problem when compared to an optimal dynamic algorithm. Our two results rely on different algorithms and techniques and hence are of independent interest.

1.2. Related Work

The dynamic balanced graph partitioning problem was introduced by Avin et al. [6, 8]. Besides the lower bound mentioned above (which even holds for ring graphs), they also present a deterministic online algorithm which achieves a competitive ratio of $O(k \log k)$. Their algorithm however relies on expensive repartitioning operations and has a super-polynomial runtime. Forner et al. [5] later showed that a competitive ratio of $O(k \log k)$ can also be achieved with a polynomial-time online algorithm which monitors the connectivity of communication requests over time, rather than the density. Pacut et al. [7] further contributed an $O(\ell)$ -competitive online algorithm for a scenario without resource augmentation and the case where $k = 3$. The dynamic graph partitioning problem has

also been studied from an offline perspective by Räcke et al. who presented a polynomial-time $O(\log n)$ -approximation algorithm [16], using LP relaxation and Bartal’s clustering algorithm to round it.

Deterministic online algorithms for the ring communication pattern have also been studied already, in a model where the adversary needs to generate the communication sequence from a random distribution in an *i.i.d.* manner [11, 17]: in this scenario, it has been shown that even deterministic algorithms can achieve a polylogarithmic competitive ratio. The problem is however very different from ours and the corresponding algorithms and techniques are not applicable in our setting.

So far, to the best of our knowledge, randomized online algorithms have only been studied in the learning variant introduced by Henzinger et al. [4, 10]. In their first paper on the learning variant, Henzinger et al. [10] still only studied deterministic algorithms and presented a deterministic exponential-time algorithm with competitive ratio $O(\ell \log \ell \log k)$ as well as a lower bound of $\Omega(\log k)$ on the competitive ratio of any deterministic online algorithm. While their derived bounds are tight for $\ell = O(1)$ servers, there remains a gap of factor $O(\ell \log \ell)$ between upper and lower bound for the scenario of $\ell = \omega(1)$. In [4], Henzinger et al. present deterministic and randomized algorithms which achieve (almost) tight bounds for the learning variant. In particular, a polynomial-time randomized algorithm is described which achieves a polylogarithmic competitive ratio of $O(\log \ell + \log k)$; it is proved that no randomized online algorithm can achieve a lower competitive ratio. Their approach establishes and exploits a connection to generalized online scheduling, in particular, the works by Hochbaum and Shmoys [18] and Sanders et al. [19].

More generally, our model is related to dynamic bin packing problems which allow for limited *repacking* [20]: this model can be seen as a variant of our problem where pieces (resp. items) can both be dynamically inserted and deleted, and it is also possible to open new servers (i.e., bins); the goal is to use only an (almost) minimal number of servers, and to minimize the number of piece (resp. item) moves. However, the techniques of [20] do not extend to our problem.

1.3. Organization

The remainder of this paper is organized as follows. We introduce our model and give an overview of our results in Section 2. The dynamic model is studied in Section 3 and the static model in Section 4. We conclude our paper in Section 5.

2. Model and Results

We formally define the dynamic balanced graph partitioning problem for ring demands, as follows. Let ℓ denote the number of servers, and k the *capacity* of a server, i.e., the maximum number of processes that can be scheduled on a single machine. We use $P = \{p_0, p_1, \dots, p_{n-1}\}$ with $n \leq \ell k$ to denote the set of processes. Naming of processes is done modulo n , i.e., p_i with $i \geq n$ refers to process $p_{i \bmod n}$.

In each time step t we receive a request $\sigma_t = \{p_j, p_{j+1}\}$ with $p_j, p_{j+1} \in P$, which means that these two processes communicate. Notice, that this restricts the communication pattern to a cycle.

Serving a communication request incurs cost of exactly 1, if both requested processes are located on different servers, otherwise 0. We call this the *communication cost* of the request. After the communication an online algorithm may (additionally) decide to perform an arbitrary number of migrations. Each migration of a process to another server induces a cost of 1, and contributes to the *migration cost* of the request.

In the end after performing all migrations each server should obey its capacity constraint, i.e., it should have at most k processes assigned to it. The goal is to find an online scheduling of processes to servers for each time step that minimizes the sum of migration and communication cost and obeys the capacity constraints.

We will compare our online algorithms to optimum offline algorithms that on the one hand are more powerful as they know the whole request sequence in advance, but on the other hand are more restricted in the migrations that they are allowed to perform. Firstly, we use resource augmentation. This means the offline algorithms have to strictly obey the capacity constraint, i.e., they can schedule at most k processes on any server, while the online algorithm may schedule αk processes on any server, for some factor $\alpha > 1$. Then we say that the online algorithm uses resource augmentation α .

In this model we obtain the following result.

Theorem 2.1. *There is a randomized algorithm that solves the dynamic balanced graph partitioning problem for ring demands with expected cost $O(\frac{1}{\epsilon} \log^3 k) \text{OPT} + c$ and resource augmentation $2 + \epsilon$, where OPT is the cost of an optimal dynamic algorithm and c is a constant not dependent on the request sequence.*

One disadvantage of this result is the additive constant c , which means that the algorithm is not strictly competitive. Note that there may exist very long request sequences that have a very low optimum cost. Then the above theorem would not give good guarantees.

In a second model we compare the performance of an online algorithm (that again uses resource augmentation) to that of an optimal *static* algorithm. Such an algorithm is only allowed to perform migrations in the beginning before the first request arrives. In this model our goal is to be *strictly competitive*. We show the following theorem.

Theorem 2.2. *There is a randomized algorithm that solves the dynamic balanced graph partitioning problem for ring demands with expected cost $O(\frac{1}{\epsilon^2} \log^2 k) \text{OPT}$ and resource augmentation $3 + \epsilon$, where OPT is the cost of an optimal static algorithm.*

3. The Dynamic Model

In this section we present an online polylog-competitive algorithm against a dynamic optimal algorithm. The main idea of our online algorithm is the reduction of the balanced graph partitioning problem for ring demands to the metrical task systems problem on a line. Metrical task system (MTS) is an online minimization problem that was first

introduced in [21] and has been extensively studied with various results for different types of the underlying metric [21–25]. There exist a tight $(2n - 1)$ competitive deterministic algorithm [21] and a tight $\mathcal{O}(\log^2 n)$ competitive randomized algorithm [25, 26], if n is the number of states in the system.

Notation

For ease of exposition we model the problem of scheduling processes on servers as a dynamic coloring problem, as follows. We identify each server s with a unique color c_s and *color* a process with the color of the server on which it is currently scheduled.

We call the set of consecutive processes $S = \{p_s, \dots, p_{s+\ell-1}\}$ the *segment* of length ℓ starting with p_s . For ease of notation we use $S = [s, s + \ell - 1]$ to refer to this set. We refer to a process pair $\{p_i, p_{i+1}\}$ as an *edge* of the cycle. To simplify the notation we denote such an edge with $(i, i + 1)$.

For an algorithm ALG we use $\text{ALG}(\sigma)$ to denote the cost of the algorithm ALG on σ . However, usually σ will be clear from the context. Then we use ALG to denote both, the algorithm and its cost.

3.1. Algorithm

Our strategy is to maintain a set of cut-edges which partition the cycle into *slices*. We ensure that we have at most ℓ slices, each of size at most $(2 + \epsilon)k$, such that we can map the slices directly to servers with resource augmentation $(2 + \epsilon)$. Each cut-edge is contained in an interval I , i.e., its position is constrained to the interval. The problem of choosing the cut-edge in some interval I reduces to the metrical task systems problem (MTS). The idea is to run a black box MTS algorithm for each interval independently.

Formally, let $k' := \lceil (1 + \epsilon)k \rceil$ and $\ell' = \lceil \frac{n}{k'} \rceil$. The algorithm ONL_R uses a *shift parameter* $R \in \{0, \dots, k' - 1\}$ that gives rise to intervals $I_1, \dots, I_{\ell'}$, where $\{I_i = [R + (i - 1)k', R + ik']\}$. Observe, that successive intervals share one vertex and intervals $I_{\ell'}$ and I_1 may also share edges. Each vertex is contained in at most two intervals.

Reduction to MTS A metrical task systems problem is defined as follows. We are given a metric (S, d) with $|S| = n$ states and a starting state $s_0 \in S$. Each time step we receive a task σ_i and a cost vector T_i , with $T_i(s)$ describing the cost of processing the task σ_i in state s . In response to σ_i an algorithm for the metrical task systems must choose a state $s_i \in S$ and pay the cost $d(s_{i-1}, s_i) + T_i(s_i)$. The goal is to minimize the overall incurred cost.

The online algorithm starts an MTS-instance for every interval. This instance is responsible for choosing an edge inside the interval and is defined as follows. Let σ_I be the restriction of the request sequence to the requests in I . For each interval I we start an instance M_I of an MTS algorithm, where the states are the edges of the interval. On a request $e \in \sigma_I$ we generate a cost vector T with $T(e') = 1$, if $e' = e$ and $T(e') = 0$ otherwise. We forward the cost vector T to the MTS instance M_I and observe the new state $e'' \in I$. Subsequently, we move our cut-edge to e'' .

Server Mapping The cut-edges inside the intervals induce a mapping of processes to servers as follows. Let $e_i = (a, a + 1)$ and $e_{i+1} = (b, b + 1)$ be the cut-edges chosen in intervals I_i and I_{i+1} , respectively. We schedule the segment $[a + 1, b]$ on the server s_i . Observe, that a server receives exactly one slice. However, the slice formed between $e_{i'}$ and e_1 could be empty because of the overlap between intervals $I_{i'}$ and I_1 .

3.2. Analysis

We first argue that the schedule produced by the online algorithm roughly balances the processes among the servers.

Lemma 3.1. *The load of each server is at most $2(1 + \epsilon)k$.*

Proof. A slice is the set of processes between the two cut-edges in two consecutive intervals. Since the intervals contain at most $(1 + \epsilon)k + 1$ processes, a slice can contain at most $2(1 + \epsilon)k$ processes. \square

The overall cost of the algorithm consist of two parts. The first part is the *communication cost*, which we denote by cost_{hit} . The second part is the *migration cost*, which we denote by cost_{mig} .

Interval Based Strategy

For the analysis we introduce different types of (optimum) algorithms under various restrictions and compare them to one another. The first concept is the concept of an *interval based strategy*. This is an algorithm that has to choose a cut-edge in each interval and pays 1 if the cut-edge is requested, and d if it moves the cut-edge by d positions. For an interval I and an interval based strategy ALG we define $\text{cost}_{\text{hit}}^{\text{ALG}}(I)$ as the total hit cost experienced by algorithm ALG on the cut-edge maintained in interval I . Similarly, we define $\text{cost}_{\text{move}}^{\text{ALG}}(I)$ as the cost for moving the cut-edge. Observe that our online algorithm ONL is an interval based strategy.

Observation 3.2. *The communication cost and migration cost of the online algorithm can be bounded by the interval cost. This means*

- $\text{cost}_{\text{hit}} \leq \sum_I \text{cost}_{\text{hit}}^{\text{ONL}}(I)$
- $\text{cost}_{\text{mig}} \leq \sum_I \text{cost}_{\text{move}}^{\text{ONL}}(I)$

Proof. If an edge is moved in an interval (by one position) the incident slices may change by at most one process. This means at most one process has to be migrated to a different server (note that due to the overlap it could happen that no slice changes when an edge is moved). Similarly, whenever two processes located on different servers want to communicate this communication takes place along a cut-edge. This is a cut-edge in at least one interval (perhaps in two due to overlap) and therefore this communication cost is counted in the interval cost. \square

We define $\text{ONL}_R := \sum_I \text{cost}_{\text{hit}}^{\text{ONL}}(I) + \sum_I \text{cost}_{\text{move}}^{\text{ONL}}(I)$ and use it as a proxy for the cost of our online algorithm when using shift parameter R (observe that the real cost could be lower due to the above observation). Let $\text{OPT}_R := \sum_I \text{cost}_{\text{hit}}^{\text{OPT}}(I) + \sum_I \text{cost}_{\text{move}}^{\text{OPT}}(I)$ denote the cost of an optimal interval based strategy (that uses shift parameter R).

Lemma 3.3. *For any R we have $E[\text{ONL}_R] \leq \alpha(k) \cdot \text{OPT}_R + c$, where $\alpha(k)$ is the competitive ratio of the underlying MTS algorithm on k states, and c a constant, that is independent of the request sequence.*

Proof. Each subproblem on I is essentially a metrical task systems problem on a line metric. An interval I consist of k' edges that form the states. The cut-edge e is the current state. On a request $e \in I$, we generate the cost vector T , where $T(e') = 1$ if $e' = e$ and 0 otherwise. Let σ_I be the request sequence, constrained only to edges in I . Let $\text{OPT}_{\text{MTS}}(I)$ be the cost of an MTS algorithm (on the line I), that serves σ_I optimally. By definition this is equal to $\text{cost}_{\text{hit}}^{\text{OPT}_R}(I) + \sum_I \text{cost}_{\text{move}}^{\text{OPT}_R}(I)$.

Because we are using an $\alpha(k)$ -competitive algorithm in each interval we get $E[\text{ONL}_R(I)] \leq \alpha(k) \cdot \text{OPT}_{\text{MTS}}(I) + c'$ for some constant c that does not depend on the request sequence. Summing over all I gives $\text{ONL}_R \leq \alpha(k) \cdot \text{OPT}_R + \ell c'$. \square

Well Behaved Strategy

Now we introduce a so-called *well behaved* clustering strategy and analyze its cost. We first show that a well behaved strategy can simulate the real optimum with a small loss and then we show that an interval based strategy with a random choice of R can simulate the best well behaved strategy with a small loss. Then the final result follows from Lemma 3.3.

We define a well behaved clustering strategy W as a strategy that maintains a set of cut edges $E_W = \{e_1, \dots, e_m\}$ which partition the cycle into segments S_1, \dots, S_m . It can perform two operations:

- *Move:* The cut edge $e = (i, i + 1)$ is moved to $e' = (j, j + 1)$ which induces cost $|j - i|$. A *merge* of two segments is simulated by a move operation, where we move a cut edge e to the position of another cut edge e' and remove e from E_W .
- *Split:* The segment S_i is split into several subsegments by introducing new cut edges in S_i . The split operation induces no cost.

On a request σ_i , if $\sigma_i \in E_W$, then the algorithm has cost 1 (*hitting cost*) or it moves the corresponding cut edge and pays the moved distance (*moving cost*). Furthermore, the size of a segment S_i is limited to $(1 + \epsilon)k$. For technical reasons we assume $\epsilon \leq \frac{1}{4}$.

The following crucial lemma shows that an optimal well behaved clustering strategy is at most an $\mathcal{O}(\log k)$ factor away from OPT.

Lemma 3.4. *There exists a well behaved clustering strategy W with cost at most $\frac{4}{\epsilon} \log k \cdot \text{OPT} + 2n \log k$.*

Proof. We develop a well behaved clustering algorithm W with the knowledge of OPT's choices. Let $c(p)$ be the color of the server where OPT places p at the current time step. We denote by $E_O = \{e = (i, i + 1) \mid c(i) \neq c(i + 1)\}$ the cut-edges of the optimal algorithm.

Let \mathcal{S}_W be the set of segments algorithm W produces. Segment $S \in \mathcal{S}_W$ is δ -*monochromatic*, if at least $\delta|S|$ processes in S have the same color c . We fix $\delta := \frac{1}{1+\epsilon}$. Each time OPT migrates a process, we mark it. Marked processes will be a source of potential for our algorithm. At some point we will remove marks from processes. We maintain the following invariants:

(IH) $E_W \subseteq E_O$.

(IM) All segments S of algorithm W are δ -monochromatic.

(IS) All processes of a segment S with non-majority color are marked.

Due to invariant (IH) the hitting cost of W is at most the hitting cost of OPT. Due to invariant (IM) the size of each segment S is at most $(1 + \epsilon)k$, since the number of processes with the majority color in S is at least $\delta|S|$ and at most k . This gives $|S| \leq (1 + \epsilon)k$.

At the start, the segments of W are essentially 1-monochromatic segments of the initial distribution; $E_W = E_O$ and there are no marked processes. On a new request σ_t we mark all processes OPT migrated in time step t . Let M_t be the number of marked processes after time step t and let $k' = (1 + \epsilon)k$. We introduce the potential function

$$\Phi_t = \frac{1 + \epsilon}{\epsilon} \log(k') M_t + \sum_{S \in \mathcal{S}_W} |S| \log\left(\frac{k'}{|S|}\right).$$

Now, we analyze the amortized costs that we incur due to the movements of OPT and our adjustments.

OPT Movement. Let o_t denote the number of newly marked processes in time step t . Then, the potential increases by $\Delta\Phi_{\text{opt}} = \frac{1+\epsilon}{\epsilon} \log(k') o_t$. OPT's moving cost in this time step is at least $\sum_t o_t$.

After OPT performed its movements, we have to maintain the invariants. Let $e_j \in E_W \setminus E_O$. The edge e_j separates two segments, L and R , in for algorithm W . Prior to request σ_t , the segments L and R were δ -monochromatic due to invariant (IM). Let c_L and c_R be their majority colors, respectively. For each $e_j \in E_W \setminus E_O$ we perform a merge, a move, or a cut-out operation.

Merge Operation. If $c_L = c_R$, we merge the segments L and R (wlog. $|L| \leq |R|$) and pay $c_{\text{merge}} = |L|$. The new segment $S = L \cup R$ has the size $|S| = |L| + |R|$. Notice, that

the invariant (IS) is maintained, we do not remove any marks. Let $|L| = l$ and $|R| = r$. Our change in the potential function is then:

$$\begin{aligned}
\Delta\Phi_{merge} &= |S| \log\left(\frac{k'}{|S|}\right) - r \log\left(\frac{k'}{r}\right) - l \log\left(\frac{k'}{l}\right) \\
&= l(\log\left(\frac{k'}{l+r}\right) - \log\left(\frac{k'}{l}\right)) + r(\log\left(\frac{k'}{l+r}\right) - \log\left(\frac{k'}{r}\right)) \\
&= l \log\left(\frac{l}{l+r}\right) + r \log\left(\frac{r}{l+r}\right) \\
&= l \log\left(1 - \frac{r}{l+r}\right) + r \log\left(1 - \frac{l}{l+r}\right) \\
&\leq l\left(-\frac{r}{l+r}\right) + r\left(-\frac{l}{l+r}\right) \\
&= -l\left(\frac{2r}{l+r}\right) \\
&\leq -l
\end{aligned}$$

The first inequation is due to the well known fact $\log(1+x) \leq x$. The second inequality holds because $l \leq r$. Thus, the amortized cost of the merge operation is $\tilde{c}_{merge} = c_{merge} + \Delta\Phi_{merge} \leq 0$.

Now suppose that $c_L \neq c_R$. Let $e_l = (l, l+1) \in E_O$ be the nearest cut edge of OPT "left" of e_j , and $e_r = (r, r+1) \in E_O$ the nearest cut edge of OPT "right" of e_j , respectively. Let $F = [l+1, r]$. By construction F contains only processes with the same color c . We differentiate whether one of the colors c_L, c_R is equal to c or not. We need the following fact for the analysis.

Fact 3.5. *Let $f(d) = (s-d) \log\left(\frac{s}{s-d}\right)$ with constants $s \geq 2$, $1 \leq d < s$. Then, $f(d) \leq d$.*

Proof. We have $f(d) = (s-d) \log\left(\frac{s}{s-d}\right) = (s-d) \log\left(1 + \frac{d}{s-d}\right) \leq (s-d)\left(\frac{d}{s-d}\right) = d$, where the inequality is due to the well known fact $\log(1+x) \leq x$. \square

Move Operation. If $c_L = c$, all processes in $F \cap R$ must be marked due to invariant (IS). Then, we move e_j to e_r , a distance $d = |F \cap R|$ and remove the marks of $F \cap R$. If $c_R = c$, we act analogously. Our actual cost is $c_{move} = d$. Let $|L| = l$ and $|R| = r$. Then, the change in potential is:

$$\begin{aligned}
\Delta\Phi_{move} &= [(l+d) \log\left(\frac{k'}{l+d}\right) - l \log\left(\frac{k'}{l}\right)] + [(r-d) \log\left(\frac{k'}{r-d}\right) - r \log\left(\frac{k'}{r}\right)] - d \frac{1+\epsilon}{\epsilon} \log k' \\
&= l \log\left(\frac{l}{l+d}\right) + r \log\left(\frac{r}{r-d}\right) + d \log\left(\frac{k'}{l+d}\right) - d \log\left(\frac{k'}{r-d}\right) - d \frac{1+\epsilon}{\epsilon} \log k' \\
&\leq r \log\left(\frac{r}{r-d}\right) - d \log\left(\frac{k'}{r-d}\right) - d \log k' - d \log(l+d) \\
&\leq (r-d) \log\left(\frac{r}{r-d}\right) - d \log k' - d \log(l+d) \\
&\leq d - d \log k' - d \log(l+d) \leq -d
\end{aligned}$$

The first inequality holds because $\log\left(\frac{l}{l+d}\right) < 0$ and $\frac{1+x}{x} \geq 2$, for $0 < x \leq 1$. The second last inequality holds due to Fact 3.5. The last inequality holds since $k' \geq 2$ and $l+d \geq 2$. Hence, the amortized cost of the move operation is $\tilde{c}_{move} = c_{move} + \Delta\Phi_{move} \leq 0$.

Cut-out Operation. Now, assume that $c \neq c_L$ and $c \neq c_R$. Then, all $p \in F$ must be marked. Wlog. let $|j - l| \leq |j - r|$. We move e_j to e_l and make a split in R by creating a new cut edge e_r . F now becomes a new segment for algorithm W. We pay at most $|F|/2$ for the movement of e_j . Afterwards, we remove the marks from the processes in F . Notice, that the new segment F is 1-monochromatic. Furthermore invariants (IM) and (IS) are also maintained. Let $|L| = l$, $|R| = r$, $|F| = d$, $|j - l| = d_l$, and $|j - r| = d_r$. The change in potential is:

$$\begin{aligned}
\Delta\Phi_{cut-out} &= [(l - d_l) \log(\frac{k'}{l-d_l}) - l \log(\frac{k'}{l})] \\
&\quad + [(r - d_r) \log(\frac{k'}{r-d_r}) - r \log(\frac{k'}{r})] + d \log(\frac{k'}{d}) - d \frac{1+\epsilon}{\epsilon} \log k' \\
&\leq [l \log(\frac{l}{l-d_l}) - d_l \log(\frac{k'}{l-d_l})] + [r \log(\frac{r}{r-d_r}) - d_r \log(\frac{k'}{r-d_r})] - 5d \log k' \\
&\leq (l - d_l) \log(\frac{l}{l-d_l}) + (r - d_r) \log(\frac{r}{r-d_r}) - 5d \log k' \\
&\leq d_l + d_r - 5d \log k' \\
&= d - 5d \log k' \\
&\leq -d
\end{aligned}$$

The first inequality is due to $\frac{1+x}{x} \geq 6$, for $0 < x \leq \frac{1}{4}$. For the second inequality we use Fact 3.5. Hence, the amortized cost of the cut-out operation is $\tilde{c}_{cut-out} = c_{cut-out} + \Delta\Phi_{cut-out} \leq 0$.

After performing the above adjustments we ensured that invariants (IH) and (IS) hold, i.e., $E_W \subseteq E_O$ and all non-majority color processes are marked. We have to take care of the (IM) invariant that all segments are δ -monochromatic. For this we use the split-operation.

Split Operation. If there exists a segment S in W that is not δ -monochromatic anymore, we simply make a "full" split. We create new cut-edges $S \cap E_O$, such that segment S breaks up into smaller 1-monochromatic pieces. Since S is not δ -monochromatic anymore, there are at least $(1 - \delta)$ marked processes in S due to invariant (IS) . We remove marks from these processes. Let T_1, \dots, T_m be the resulting subsegments of S . The change in potential is:

$$\begin{aligned}
\Delta\Phi_{split} &\leq [\sum_i |T_i| \log(\frac{k'}{|T_i|}) - |S| \log(\frac{k'}{|S|})] - (1 - \delta) |S| \frac{1+\epsilon}{\epsilon} \log k' \\
&\leq \log(k') \sum_i |T_i| - |S| \log(\frac{k'}{|S|}) - |S| \log k' \\
&= |S| \log k' - |S| \log(\frac{k'}{|S|}) - |S| \log k' \\
&= -|S| \log(\frac{k'}{|S|}) \\
&\leq 0
\end{aligned}$$

The last inequality holds, because $|S| \leq (1 + \epsilon)k = k'$. Thus, the amortized cost of the split operation is $\tilde{c}_{split} = \Delta\Phi_{split} \leq 0$.

From the above analysis we conclude, that in each time step t our amortized cost is at most $\tilde{c}_t \leq \frac{1+\epsilon}{\epsilon} \log(k') o_t$. Then, our overall cost W is at most $W \leq \frac{1+\epsilon}{\epsilon} \log(k') \sum_t o_t + \Phi_0 \leq 2(1+\epsilon)^2/\epsilon \cdot \log(k) \cdot \text{OPT} + 2n \log k$. As $2(1+\epsilon)^2 \leq 4$ for $\epsilon \leq \frac{1}{4}$ the lemma follows. \square

Lemma 3.6. *Let OPT_W be an optimal well behaved clustering strategy. When choosing the shift parameter R uniformly at random from $\{0, k' - 1\}$ the optimum interval based strategy has expected cost $E_R[\text{OPT}_R] \leq 6\text{OPT}_W$.*

Proof. We add additional cost of k' to a well behaved strategy OPT_W each time a cut edge of OPT_W crosses the boundary of a interval. Let OPT'_W denote this adapted overall cost.

There are $l' + 1$ interval borders, i.e., the probability for a given process p to be an interval border is at most $\frac{l'+1}{n} \leq \frac{2}{k'}$. If a cut edge of OPT_W moves a distance of d , then each process on the way is an interval border with probability at most $2/k'$. This gives, that the expected movement costs are at most $d + d \frac{2}{k'} k' = 3d$. Hence, $E[\text{OPT}'_W] \leq 3\text{OPT}_W$.

We develop an interval based algorithm that tries to mimic OPT_W . Fix some interval I . Since $k' = (1+\epsilon)k$, we know that each well behaved strategy should have always at least one cut-edge inside I . We choose one such cut edge e and every time e is moved, we simply try to follow its movement. If the movement takes e outside of I , we know that OPT'_W pays cost k' . Then, instead of following e we choose another cut-edge inside I and travel there at a cost of at most k' .

By this construction all cost (hitting or movement) of the interval based algorithm can be charged to the movement or hitting cost of an edge in OPT'_W . However, note that intervals may overlap. This means there may be two intervals in the interval based algorithm that charge against the same edge of OPT'_W . Nevertheless we still get $E_R[\text{OPT}_R] \leq 2\text{OPT}'_W \leq 6\text{OPT}_W$, as desired. \square

Theorem 2.1. *There is a randomized algorithm that solves the dynamic balanced graph partitioning problem for ring demands with expected cost $O(\frac{1}{\epsilon} \log^3 k) \text{OPT} + c$ and resource augmentation $2 + \epsilon$, where OPT is the cost of an optimal dynamic algorithm and c is a constant not dependent on the request sequence.*

Proof. We have $E_R[\text{ONL}_R] \leq E_R[\alpha(k) \text{OPT}_R + c']$ (Lemma 3.3), $E_R[\text{OPT}_R] \leq 6\text{OPT}_W$ (Lemma 3.6) and $\text{OPT}_W \leq \frac{4}{\epsilon} \log k \cdot \text{OPT} + 2n \log k$ (Lemma 3.4), where $\alpha(k)$ denotes the competitive ratio of an algorithm for metrical task systems on a line with $\mathcal{O}(k)$ states. By using the algorithm presented in [25] (which achieves competitive ratio $\mathcal{O}(\log^2 k)$ on any metric space), we conclude that the cost of the online algorithm fulfills $\text{ONL} \leq \mathcal{O}(\frac{1}{\epsilon} \log^3 k) \text{OPT} + c$, where c is a constant not dependent on the request sequence. By Lemma 3.1 we know that the load of each server is at most $(2 + 2\epsilon')k$. Setting $\epsilon' = \epsilon/2$ gives the theorem. \square

4. The Static Model

In this section we present an polylog-competitive online algorithm against an optimal static algorithm. We start with a solution for a simpler problem that will serve as a main ingredient for our algorithm.

Notation

For ease of exposition we model the problem of scheduling processes on servers as a dynamic coloring problem, as follows. We identify each server s with a unique color c_s and *color* a process with the color of the server on which it is currently scheduled.

We call the set of consecutive processes $S = \{p_s, \dots, p_{s+\ell-1}\}$ the *segment* of length ℓ starting with p_s . For ease of notation we use $S = [s, s + \ell - 1]$ to refer to this set. For a parameter δ we call S δ -*monochromatic* for a color c if strictly more than $\delta|S|$ processes $p \in S$ are *initially* colored with c . We call a segment *monochromatic* if it is δ -monochromatic for a value $\delta \geq 1/2$. In this case we call c the majority color of segment S . Let $\bar{\delta} := \max\{\frac{2}{2+\epsilon}, \frac{14}{15}\}$, for a $\epsilon > 0$.

We refer to a process pair $\{p_\ell, p_{\ell+1}\}$ as an *edge* of the cycle. To simplify the notation we denote such an edge with $(\ell, \ell + 1)$.

4.1. Hitting Game on the Line

In this section we analyze a simplified problem, that will serve as a building block for our algorithm. The simplified problem is defined as follows.

A line of $k + 1$ nodes $V = \{v_1, v_2, \dots, v_{k+1}\}$ and k edges $E = \{e_1, e_2, \dots, e_k\}$ with $e_i = \{v_i, v_{i+1}\}$ is given. Our initial position is the central edge $e_s, s = \lceil \frac{k}{2} \rceil$. Each time step $t, 1 \leq t \leq N$, we receive a request $e \in E$. If our current position is e , we may stay there and pay cost of 1 (the *hitting cost*). Alternatively, we could change our position and pay the traveled distance (the *moving cost*), where the distance between edge e_i and e_j is $d(e_i, e_j) = |i - j|$.

In our algorithm we use techniques of Blum et al. [27] to maintain a probability distribution over the edge set. However, in [27] they consider a uniform metric, whereas we have a line, i.e., they could switch between any two states with a cost of 1. We need to choose our probability distribution carefully to correctly incorporate our moving cost.

We compare our algorithm's cost against an optimal static strategy that chooses one position e_p at the beginning, pays the distance $|s - p|$ and stays at this position for all subsequent requests. Let OPT be the cost of such an optimal static algorithm. Our first observation is that we have to use a randomized strategy in order to be better than $\Omega(k)$ OPT:

Lemma 4.1. *Any deterministic online algorithm has cost at least $\Omega(k)$ OPT.*

Proof. Since the adversary knows the position of the deterministic algorithm DET, it just requests its position each time. Then, after time step $T \geq k^2$ the algorithm DET has cost at least T . On the other hand, by an averaging argument there must be an edge that was requested at most $\frac{T}{k}$ times. Traveling there costs at most k . Then, $\frac{T}{T/k+k} \geq \frac{k}{2}$. \square

Observe, that a reasonably competitive algorithm should not move too far away from the starting position right at the beginning, since the optimal static algorithm could stay nearby to the starting position and pay only constant hitting cost. On the other hand, it has to react fast enough to increasing hitting cost.

Let $x^{(t)}$ denote the *request vector*, with $x_e^{(t)}$ describing the number of requests to edge e up until time step t and let $x_I^{(t)}$ be the restriction of the vector $x^{(t)}$ to the edges inside an interval $I = \{v_l, \dots, v_r\}$. Note, that $x^{(0)} = 0$.

Interval Growing Algorithm

The *interval growing algorithm* maintains an interval I around the starting edge and restricts the possible positions only to this interval. Note that an interval I contains $|I| - 1$ edges. The algorithm proceeds in phases. We begin with interval $I_0 = \{v_s, v_{s+1}\}$, i.e., only the starting edge is contained in I_0 . We say I_0 is the *initial* interval. Let $I = [\ell, r]$ be our current interval. We denote by $\min(I)$ the minimum $\min_{e \in I}(x_e^{(t)})$ at the current time step t . Whenever $\min(I)$ reaches $(1 - \delta)|I|$ we double the size of the interval and set $I' := [\ell - |I|/2, r + |I|/2]$ (and then choose a new edge inside I'). Thus, a new phase begins with I' as our new interval. There is one exception to this growth rule: when the length of the new interval would be larger than $k + 1$ we define $I' := [\ell - \lceil(k + 1 - |I|)/2\rceil, r + \lceil(k + 1 - |I|)/2\rceil]$, i.e., we only give the new interval a length of $k + 1$.

During each phase we maintain a probability distribution over the edge set of our current interval. Let I be our current interval. We select a random edge inside I according to the probability distribution $p^{(t)} = \nabla \text{smin}_{|I|-1}(x_I^{(t)})$ (see Appendix A).

Let $\text{smin}'(x)$ denote $\text{smin}_d(x)$ for a d -dimensional vectors x . On a new request our probability distribution changes from $p^{(t-1)}$ to $p^{(t)}$, subsequently we have to move according to the change in the probability distribution in order to maintain the invariant. At the end of a phase, we grow our interval, and subsequently choose a new edge inside the new interval I' , which incurs cost at most $|I'|$.

We denote by $\text{cost}_{\text{hit}}^{(t)}$ and $\text{cost}_{\text{move}}^{(t)}$ the incurred hitting and moving cost at time t , respectively. Let $\ell^{(t)}$ be a vector such that $\ell_i^{(t)} = 1$ if e_i is the requested edge at time t and $\ell_j^{(t)} = 0$ for $j \neq i$. Notice, that $x^{(t-1)} + \ell^{(t)} = x^{(t)}$. Then, the expected hitting cost at time t is at most $E[\text{cost}_{\text{hit}}^{(t)}] = (p^{(t-1)})^T \ell^{(t)}$. Since the distance between two edges is at most k , the expected moving cost is at most k times the Earthmover distance between distributions $p^{(t-1)}$ and $p^{(t)}$, which in our case is at most $k \|p^{(t)} - p^{(t-1)}\|_1$. The advantage of using the $\nabla \text{smin}'$ function is that we ensure that our moving cost is comparable to the hitting cost.

Lemma 4.2. *Let I be the current, non-initial interval of the interval growing algorithm. Then, $\text{OPT} \geq \max\{\frac{1}{2} \min(I), \frac{1-\delta}{2} |I|\}$.*

Proof. Let I' denote the interval I before the most recent growth step. At the beginning OPT makes its move and either stays in I' and suffers at least $\min(I')$ communication

cost or moves out of I' and pays $\frac{1}{2}|I'|$ moving cost. Since $|I| = 2|I'|$ and $\min(I') \geq (1 - \bar{\delta})|I'|$, we conclude that $\text{OPT} \geq \frac{1-\bar{\delta}}{2}|I|$.

Assume I has the maximum possible size $k + 1$. Then, $\text{OPT} \geq \min(I)$ is a trivial lower bound, since OPT has to choose one position in I . Otherwise, by construction $\min(I) \leq (1 - \bar{\delta})|I|$ and since $\text{OPT} \geq \frac{1-\bar{\delta}}{2}|I|$ we conclude that $\text{OPT} \geq \frac{1}{2} \min(I)$. \square

Let I be the current interval of the interval growing algorithm. We denote by $\text{cost}_{\text{hit}}(I)$ the overall hitting cost and by $\text{cost}_{\text{move}}(I)$ the overall moving cost incurred by the algorithm. Then, $\text{cost}(I) := \text{cost}_{\text{hit}}(I) + \text{cost}_{\text{move}}(I)$ denotes the overall cost of the interval growing algorithm.

Lemma 4.3. *Let I be the current interval of the interval growing algorithm. The algorithm incurs the following costs:*

- a) $E[\text{cost}_{\text{hit}}(I)] \leq 2 \min(I) + \mathcal{O}(\ln |I|) |I|$.
- b) $E[\text{cost}_{\text{move}}(I)] \leq 4 \min(I) + \mathcal{O}(\ln |I|) |I|$.
- c) $E[\text{cost}(I)] = 0$, if I is the initial interval.

Proof. If I is the initial interval, then there was no request to the initial edge yet, no costs were incurred.

Now, we fix some phase and derive the overall hitting and moving cost incurred in this phase. Let I_i denote the interval maintained in phase i . Assume that phase i start at time t_1 and ends at time t_2 . We denote by $P_{\text{hit}}^{(i)}$ and $P_{\text{move}}^{(i)}$ the incurred hitting and moving cost during phase i , respectively. We know that $E[\text{cost}_{\text{hit}}^{(t)}] \leq (\nabla \text{smin}'(x_{I_i}^{(t-1)}))^T \ell^{(t)}$. Furthermore, applying Lemma A.2 we derive that

$$\begin{aligned} E[\text{cost}_{\text{move}}^{(t)}] &\leq |I_i| \|\nabla \text{smin}'(x_{I_i}^{(t-1)} + \ell^{(t)})\| \\ &\leq 2(\nabla \text{smin}'(x_{I_i}^{(t-1)}))^T \ell^{(t)}. \end{aligned}$$

Again, using Lemma A.2 we obtain:

$$\begin{aligned} \sum_{t=t_1}^{t_2} \nabla \text{smin}'(x_{I_i}^{(t-1)})^T \ell^{(t)} &\leq 2 \sum_{t=t_1}^{t_2} (\text{smin}'(x_{I_i}^{(t-1)} + \ell^{(t)}) - \text{smin}'(x_{I_i}^{(t-1)})) \\ &= 2 \sum_{t=t_1}^{t_2} (\text{smin}'(x_{I_i}^{(t)}) - \text{smin}'(x_{I_i}^{(t-1)})) \\ &= 2(\text{smin}'(x_{I_i}^{(t_2)}) - \text{smin}'(x_{I_i}^{(t_1-1)})) \\ &\leq 2(\min(x_{I_i}^{(t_2)}) + |I_i| \ln |I_i|) \end{aligned}$$

Then, $P_{\text{hit}}^{(i)} \leq 2(\min(x_{I_i}^{(t_2)}) + |I_i| \ln |I_i|)$ and $P_{\text{move}}^{(i)} \leq 4(\min(x_{I_i}^{(t_2)}) + |I_i| \ln |I_i|)$.

Let m be the current phase. We know that for $i < m$ the minimum $\min(I_i)$ at end of phase i is exactly $(1-\delta)|I_i|$. Then, $\min(I_i) + |I_i| \ln |I_i| \leq (\ln |I_i| + 1)|I_i|$. The moving costs of the interval growing algorithm are essentially the costs for maintaining the probability distribution. Additionally, at the end of each phase i we pay at most $|I_{i+1}|$ for choosing a new position in the interval I_{i+1} . Then:

$$\begin{aligned}
\text{cost}_{\text{move}}(I) &= \sum_{i=1}^{m-1} (E[P_{\text{move}}^{(i)}] + |I_{i+1}|) + E[P_{\text{move}}^{(m)}] \\
&\leq \left[\sum_{i=1}^{m-1} 4(\ln |I_i| + 1)|I_i| + |I_{i+1}| \right] + E[P_{\text{move}}^{(m)}] \\
&\leq \left[\mathcal{O}(\log |I|) \sum_{i=1}^{m-1} |I_i| \right] + E[P_{\text{move}}^{(m)}] \\
&\leq \mathcal{O}(\log |I|) |I| + 4(\min(I) + |I| \ln |I|) \\
&\leq 4 \min(I) + \mathcal{O}(\log |I|) |I|
\end{aligned}$$

Using the same steps we derive that the hitting cost $\text{cost}_{\text{hit}}(I)$ of the interval growing algorithm is at most $2 \min(I) + \mathcal{O}(\log |I|) |I|$. \square

Corollary 4.4. *The expected cost of the interval growing algorithm $E[\text{cost}(I)]$ is at most $\mathcal{O}(\frac{1}{1-\delta} \log k) \text{OPT}$.*

4.2. Algorithm

The algorithm consists of three procedures. The *Slicing Procedure*, the *Clustering Procedure* and the *Scheduling Procedure*.

1. The *Slicing Procedure* gets the request sequence as input and maintains a set of *cut edges*. These are edges in the cycle for which the two endpoints *may* be scheduled on different servers (to be determined by the clustering and scheduling procedure). The cut edges partition the cycle into *slices* (segments that start and end in a cut-edge). During the algorithm the set of slices changes dynamically as cut edges are moved or deleted (a deletion of a cut edge causes a merge of the two incident segments).
2. The input to the *Clustering Procedure* is the sequence of slice change operations as generated by the Slicing Procedure. The Clustering Procedure maintains a grouping of slices into clusters such that slices that almost exclusively contain processes with a particular (initial) color are grouped together (these are $\frac{3}{4}$ -monochromatic slices).
3. Finally, the *Scheduling Procedure* schedules the clusters on the servers. It makes sure that clusters that contain monochromatic slices for a particular server s_c are scheduled on this server.

The Slicing Procedure

The basic idea of the Slicing Procedure is to maintain a set of cut-edges by using the interval growing algorithm from Section 4.1. Initially, the cut edges are edges in the cycle for which both end-points are on different servers w.r.t. the initial distribution of processes. We create an interval $I = [i, i + 1]$ around each initial cut edge $e = (i, i + 1)$. We call e the *center* of interval I . The idea is to run the interval growing algorithm for each interval independently.

Clearly, this naïve approach does not work. One major challenge is that the analysis for the Hitting Game (see Lemma 4.3) compares the cost of the online algorithm to the cost of an optimum algorithm inside the interval. In order to salvage this analysis we have to make sure that the intervals do not overlap too much. A second difficulty is that the lower bound on OPT for Lemma 4.2 hinges on the fact that OPT needs to choose at least one edge. However, if the initial distribution contains many cut-edges then OPT does not need to choose a cut edge for every interval.

Formally, we transform the Hitting Game approach to the set of intervals as follows. Let x denote the *request vector*, with x_e describing the number of requests to edge e and let x_I be the restriction of the vector x to the edges inside an interval I . We select a random cut-edge inside each interval according to the probability distribution $\nabla \text{smin}'(x_I)$ (where $\text{smin}'(x)$ denotes $\text{smin}_d(x)$ for a d -dimensional vectors x ; see Appendix A). Whenever for an interval $[\ell, r]$ the minimum $\min_{e \in I}(x_e)$ reaches $(1 - \bar{\delta})|I|$ we double the size of the interval and set $I' := [\ell - |I|/2, r + |I|/2]$ (and then choose a new cut-edge inside I'). There is one exception to this growth rule: when the length of the new interval would be larger than $k + 1$ we define $I' := [\ell - \lceil (k + 1 - |I|)/2 \rceil, r + \lceil (k + 1 - |I|)/2 \rceil]$, i.e., we only give the new interval a length of $k + 1$.

The first change to the hitting game approach is as follows. Whenever an interval I becomes $\bar{\delta}$ -monochromatic (note that this can only happen directly after growing the interval) for $\bar{\delta}$ we stop the growth process for this interval. The interval becomes *inactive* and we do not maintain a cut-edge inside it, anymore. Consequently, the cost for the interval will not increase in the future. Note that deleting a cut-edge may induce some cost for the scheduling algorithm as it has to ensure that the neighbouring slices are scheduled on the same server.

The second change is that we sometimes deactivate intervals, in order to guarantee that intervals do not overlap too much. This is done as follows. After an interval I grows, we deactivate all intervals that are completely contained in I (we call such intervals *dominated*). This means we stop the growth process of these intervals and do not maintain a cut-edge for them anymore. Also this type of deactivation may generate additional cost because the neighbouring slices may have to be moved. The formal Slicing Procedure is shown in Algorithm 1.

Note that initially an interval has length 2 as it contains the two end-points of an initial cut-edge in the distribution of processes. Right after the first request to the single edge inside an initial interval the interval will grow. Therefore, we refer to an interval of length 2 as an *initial* interval.

We call an interval that has reached its maximum length a *final* interval as it will not

Algorithm 1: Slicing procedure

Data: requested edge e , set of active intervals \mathcal{I}
Result: new set of active intervals \mathcal{I}
 $x(e) \leftarrow x(e) + 1$
forall $I \in \mathcal{I}$ s.t. $e \in I$ **do**
 \lfloor update cut-edge according to prob. distribution $\nabla \text{smin}'(x_I)$
while $\exists I$ such that $\min_{e \in I} x(e) \geq |I|$ **do**
 grow I
 if I is $\bar{\delta}$ -monochromatic **then**
 $\lfloor \mathcal{I} \leftarrow \mathcal{I} \setminus I$ // I becomes monochromatic
 else
 forall $J \in \mathcal{I}$ such that $J \subseteq I$ **do**
 $\lfloor \mathcal{I} \leftarrow \mathcal{I} \setminus J$ // J becomes dominated
 choose cut-edge according to prob. distribution $\nabla \text{smin}'(x_I)$
 return \mathcal{I}

grow anymore. Note that such an interval is always *active*. An interval may be *inactive* because it is either $\bar{\delta}$ -monochromatic or it is *dominated* (completely contained in another interval). We call two active intervals I and J *adjacent*, if no other active interval has its center between the centers of I and J .

The Clustering Procedure

The clustering procedure groups slices into clusters such that slices that are $\frac{3}{4}$ -monochromatic for the same color are all within the same cluster. In addition it must ensure that any cluster is not too large because the scheduling procedure will not split clusters among different servers and, thus, will not be able to find a good schedule if some clusters are very large.

For every color c the clustering procedure maintains one special cluster, which we call the color c cluster. A slice S either forms a singleton cluster that only contains slice S , or it belongs to the *color* c cluster, where c is the majority color of S . The assignment of a slice to a cluster is done as follows.

- Initially all slices consist only of a single color and belong to the cluster for this color.
- Whenever a slice S changes (by movement of a cut edge, or by merging a smaller slice to it¹) we examine the new slice S' .
 - If S' does not have a majority color it becomes a singleton cluster.
 - If S' is $\frac{3}{4}$ -monochromatic for some color c it is assigned to the color c cluster.

¹ties broken arbitrarily

- Otherwise it is assigned to the cluster of its majority color iff S was assigned to this cluster. This means that if S and S' have the same majority color c and S was assigned to the color c cluster we assign S' to it. Otherwise S' forms a singleton cluster.

The Scheduling Procedure

The scheduling procedure gets the input from the clustering procedure and maintains an assignment of the clusters to the servers such that the load on the servers is roughly equal.

Upon a new request the clustering algorithm could change existing clusters. Some processes might change their cluster (maybe deleting a cluster in the process), some clusters might split. This changes the size of clusters and, hence, the load distribution among the servers can become imbalanced.

Let X be the maximal size of a cluster and $D := \max\{2, X/k\}$. The scheduling procedure *rebalances* the distribution of clusters among servers such that there are at most $(D + \epsilon)k$ processes on any server, for an $\epsilon > 0$. This is done as follows.

Assume, that after the execution of the clustering algorithm server s has load greater than $(D + \epsilon)k$. We perform the following rebalancing procedure. While server s has load greater than Dk , we take the smallest cluster C in s ($|C| \leq D$) and move it to a server s' with load at most k (such a server must exist because the average load is at most k). If $|C| \leq k$ then s' has load at most $2k$ afterwards. Otherwise we migrate the content of s' (apart from cluster C) onto another server with load at most k .

4.3. Structure of Intervals and Slices

In order to derive upper bounds on the communication and moving cost of our algorithm, we have to understand the underlying structure of the intervals and slices produced in the slicing procedure.

The next lemma states that monochromatic segments that have a large enough intersection must have the same majority color.

Lemma 4.5. *Let I and J be two overlapping δ -monochromatic segments. If $|I \cap J| \geq \alpha \max\{|I|, |J|\}$ and $\delta \geq 1 - \frac{\alpha}{2}$, the segments have the same majority color.*

Proof. Let $|I \setminus J| = a$, $|I \cap J| = b$ and $|J \setminus I| = c$. Then, $|I \cup J| = a + b + c$. Assume that the segments have different majority colors c_I and c_J , respectively. Then, the segment $|I \cup J|$ must contain strictly more than $\delta(a + b)$ elements of color c_I and strictly more than $\delta(b + c)$ elements of color c_J . Trivially, the number of elements of both colors cannot exceed $a + b + c$, i.e., $\delta(a + b) + \delta(b + c) < a + b + c$ or $\delta < \frac{a+b+c}{a+2b+c}$. Furthermore, since $b \geq \alpha(a + b)$ and $b \geq \alpha(b + c)$ we get $b \geq \frac{\alpha}{2}(a + 2b + c)$. Then,

$$\delta < \frac{a + b + c}{a + 2b + c} = 1 - \frac{b}{a + 2b + c} \leq 1 - \frac{\alpha}{2},$$

which gives a contradiction. □

Next, we argue, that the union of a sequence of consecutive overlapping δ -monochromatic intervals of the same majority color is also monochromatic.

Lemma 4.6. *Let I_1, I_2, \dots, I_m be δ -monochromatic intervals with the same majority color, such that $I = \bigcup_i I_i$ forms a single contiguous segment. Then I is $\frac{\delta}{2-\delta}$ -monochromatic for c .*

Proof. Wlog. we assume that the set of intervals does not contain *redundant* intervals, i.e., for all j , $\bigcup_{i \neq j} I_i \neq I$. Further, we assume that the intervals are numbered by the order of their centers.

Define $I_0 = I_{m+1} = \emptyset$ and let $X_i = I_i \cap I_{i+1}$ and $A_i = I_i \setminus X_{i-1}$. Then, the intervals have the following structure: $I_i = X_{i-1} \cup A_i \cup X_{i+1}$. Notice, that X_i and A_j are disjoint. For a segment S we use $f(S)$ to denote the number of elements with color c in S . Trivially, $0 \leq f(S) \leq |S|$. Furthermore, we know that

$$f(I_i) = f(X_{i-1}) + f(A_i) + f(X_i) \geq \delta(|X_{i-1}| + |S_i| + |X_i|) = \delta|I_i|,$$

for all $1 \leq i \leq m$. Then,

$$\begin{aligned} \delta|I| &= \delta \sum_{i=1}^m (|X_{i-1}| + |A_i|) \\ &= \sum_{i=1}^m \delta(|X_{i-1}| + |A_i| + |X_i|) - \sum_{i=1}^m \delta|X_i| \\ &\leq \sum_{i=1}^m (f(X_{i-1}) + f(A_i) + f(X_i)) - \sum_{i=1}^m \delta|X_i| \\ &= \sum_{i=1}^m (f(X_i) + f(A_i)) + \sum_{i=1}^m (f(X_{i-1}) - \delta|X_i|) \\ &= \sum_{i=1}^m (f(X_i) + f(A_i)) + \sum_{i=1}^m (f(X_i) - \delta|X_i|) \\ &= \sum_{i=1}^m (f(X_i) + f(A_i)) + \sum_{i=1}^m ((1-\delta)f(X_i) + \delta f(X_i) - \delta|X_i|) \\ &\leq \sum_{i=1}^m (f(X_i) + f(A_i)) + (1-\delta) \sum_{i=1}^m f(X_i) \\ &\leq (2-\delta) \sum_{i=1}^m (f(X_i) + f(A_i)) \\ &= (2-\delta)f(I) \end{aligned} \quad \square$$

The next lemma explains the structure of the slice between two adjacent active intervals. Let I be an interval with length $|I| \geq 4$ and let I' denote the interval before its most recent growth step. We say I' is the core of I and denote it by $\text{core}(I) = I'$. Note that initial intervals do not have a core.

Lemma 4.7. *Let $N = [a, b]$ be a segment such that no active interval is intersecting N . Let \mathcal{M} be the set of inactive intervals I with $\text{core}(I) \cap N \neq \emptyset$. Let $M = \bigcup_{I \in \mathcal{M}} I$ and $F = N \setminus M$. Then, there exists a color c and an interval set $\mathcal{U} \subseteq \mathcal{M}$ with the following properties:*

- a) $\bigcup_{I \in \mathcal{U}} I = M$;
- b) each interval in \mathcal{U} is $\bar{\delta}$ -monochromatic for color c ;
- c) each process in F has initial color c ;
- d) the segment $N' = M \cup F$ is $\bar{\delta}/(2-\bar{\delta})$ -monochromatic for color c .

Proof. Property d just follows from the first three properties by simply applying Lemma 4.6. Therefore, we focus on proving properties a, b and c. We prove these by induction over the number of intervals in \mathcal{M} .

Base case ($|\mathcal{M}| = 0$):

If $\mathcal{M} = \emptyset$ we have $F = N \setminus M = N$ and $\mathcal{U} = \emptyset$. Then properties a and b of the lemma are trivially fulfilled. Any initial cut-edge is either inside an active interval or inside the core of some interval. As no active interval and no core intersects N (otw. \mathcal{M} would not be empty) we get that all processes in F must have the same initial color. We choose c as this color. This fulfills Property c.

Induction step:

We call the vertices in F *free vertices* as they are not contained in any set of \mathcal{M} . We distinguish two cases.

- I) First suppose that there exists a free vertex f that has an interval of \mathcal{M} on both sides. Let \mathcal{M}_ℓ denote the set of interval to the left of f and \mathcal{M}_r denote the interval to the right of f . We apply the induction hypothesis on the sub-sequences $N_\ell := [a, f]$ and $N_r := [f, b]$ (with interval sets \mathcal{M}_ℓ and \mathcal{M}_r , respectively). This gives interval sets \mathcal{U}_ℓ and \mathcal{U}_r that are $\bar{\delta}$ -monochromatic for colors c_ℓ and c_r , respectively, and it gives sets of free vertices F_ℓ and F_r that all have color c_ℓ and c_r , respectively.

The union $\mathcal{U} := \mathcal{U}_\ell \cup \mathcal{U}_r$ of the interval sets covers M , which gives Property a. The free vertices are $F = F_\ell \cup F_r$. To show properties b and c we have to argue that $c_\ell = c_r$. But this is immediate because by construction $F_\ell \cap F_r \neq \emptyset$ as it contains vertex f .

- II) Now suppose that $M = \cup_{i \in I} I$ forms a single contiguous segment. Let $H = [h_\ell, h_r]$ with $\text{core}(H) = [c_\ell, c_r]$ be the largest interval in \mathcal{M} (ties broken arbitrarily). Let $N_\ell = [a, c_\ell]$ and $N_r = [c_r, b]$ denote the segments to the left and right of H 's core (but still sharing one vertex with the core), and let $F_\ell := N_\ell - M$ and $F_r := N_r - M$ denote the free vertices in these segments. Further define $\mathcal{M}_\ell = \{I \in \mathcal{M} \mid \text{s.t. } \text{core}(I) \cap N_\ell \neq \emptyset\}$, $\mathcal{M}_r = \{I \in \mathcal{M} \mid \text{s.t. } \text{core}(I) \cap N_r \neq \emptyset\}$, $M_\ell = \bigcup_{I \in \mathcal{M}_\ell} I$, and $M_r = \bigcup_{I \in \mathcal{M}_r} I$.

By construction $M = M_\ell \cup M_r$. We now proceed by constructing interval sets $\mathcal{U}_\ell \subseteq \mathcal{M}_\ell$ and $\mathcal{U}_r \subseteq \mathcal{M}_r$ that cover M_ℓ and M_r , respectively, and that are $\bar{\delta}$ -monochromatic for colors c_ℓ and c_r , respectively. This is sufficient to obtain properties a and b by choosing $\mathcal{U} = \mathcal{U}_\ell \cup \mathcal{U}_r$ because c_ℓ will be equal to c_r . To see this we argue that both \mathcal{U}_ℓ and \mathcal{U}_r must contain the set H . This holds for \mathcal{U}_r because H is the only set in \mathcal{M}_r that contains the vertex $h_\ell \in M_r$. An interval $X \in \mathcal{M}_r$ has a vertex to the right of c_r (including c_r) in its core. To also include h_ℓ would imply $H \subseteq X$. This is not possible as H is one of the largest intervals and no two intervals can have identical borders.

Now, we show how to construct \mathcal{U}_r . The construction for \mathcal{U}_ℓ is analogous. We distinguish two sub-cases.

- A) First suppose that h_r is the rightmost vertex in any interval of \mathcal{M} . Then

any interval in \mathcal{M}_r is completely contained in H , which means we can choose $\mathcal{U}_r = \{H\}$.

B) Now, let $J \neq H$ denote the interval that contains the rightmost vertex of M . First observe that J is $\bar{\delta}$ -monochromatic because if not there would need to be a strictly larger $\bar{\delta}$ -monochromatic interval that completely contains J in its core. Then J could not contain the rightmost vertex of M .

a) If H and J have intersecting cores then the whole segment $[c_r, h_r]$ is contained in both of them. This means that their intersection is at least $\frac{1}{4} \max\{|H|, |J|\}$. Since $\bar{\delta}/(2 - \bar{\delta}) \geq \frac{7}{8}$ we can apply Lemma 4.5 and conclude that H and J have the same majority color. Hence, we can choose $\mathcal{U}_r = \{H, J\}$ to cover M_r .

b) Suppose the cores do not intersect. Let $J = [j_\ell, j_r]$ and $\text{core}(j) = [d_\ell, d_r]$. We apply the induction hypothesis to the two sub-sequences $A = [c_r, d_\ell - 1]$ and $B = [c_r + 1, d_\ell]$. We can do this because both have at most $|\mathcal{M}| - 1$ intersecting cores. We get two $\bar{\delta}/(2 - \bar{\delta})$ -monochromatic sequences A' and B' , and interval subsets \mathcal{U}_A and \mathcal{U}_B .

$A' = [h_\ell, \gamma]$, where $\gamma \geq \max\{h_r, d_\ell - 1\}$ because A' must cover H and A . $B' = [\gamma, j_r]$, where $\gamma \leq \min\{c_\ell + 1, j_\ell\}$ because B' must cover J and B .

The intersection between A' and B' has size at least $\frac{1}{4} \max\{|A'|, |B'|\}$, which by Lemma 4.5 gives that they need to have the same majority color. Also the intervals in \mathcal{U}_A and \mathcal{U}_B have this majority color. Hence, we can choose $\mathcal{U}_r = \mathcal{U}_A \cup \mathcal{U}_B$.

So far all intervals in $\mathcal{U} = \mathcal{U}_\ell \cup \mathcal{U}_r$ have the same majority color c . It remains to show that also all in F_ℓ and F_r have this color. This then gives Property c of the lemma. Again we only do the argument for F_r .

Let $X = [x_\ell, x_r]$ denote the interval that contains the rightmost vertex of M . If $b \geq x_r$ then $F_r = \emptyset$ and there is nothing to prove. Otherwise, $F_r = [x_r + 1, b]$. We first argue that the leftmost vertex $x_r + 1$ in F_r has color c . This implies the statement because the region $[x_r + 1, b]$ cannot contain any initial cut-edges as these would be either inside an active interval or inside the core of an inactive interval, and there are no cores intersecting $[x_r + 1, b]$.

We apply the induction hypothesis to the segment $S := [c_r + 1, h_r + 1]$, which we can do because $\text{core}(H)$ does not intersect S (hence, at most $|\mathcal{M}| - 1$ intervals have a core intersecting S). By Property d this returns a segment $S' = M_r \cup F_S \supseteq S$ that is $\bar{\delta}/(2 - \bar{\delta})$ -monochromatic for some color c_S , and all vertices in $F_S = S \setminus M_r$ have color c_S .

The intersection between S' and H is at least $\frac{1}{4} \max\{|H|, |S'|\}$. Since $\bar{\delta}/(2 - \bar{\delta}) \geq \frac{7}{8}$ we can apply Lemma 4.5 and conclude that H and S' have the same majority color, i.e., $c_S = c$. In addition we get that the vertex $x_r + 1$ has color c , as well, as it is contained in F_S .

This finishes the proof of Case II of the induction step. \square

Lemma 4.8. *Let $L = [a, b]$ and $R = [c, d]$ be two non-intersecting intervals, such that no active interval intersects the segment $N = [b + 1, c - 1]$. Let $e_\ell = (\ell, \ell + 1)$ and $e_r = (r, r + 1)$ be the centers of L and R , respectively, and let $S = [a, d]$ be the segment containing both L and R . Let \mathcal{M} be the set of non-initial intervals with centers in $[\ell + 1, r]$ and let $M = \sum_{I \in \mathcal{M}} |I|$.*

If $|S| \geq \frac{1}{1-\delta}(M + |L| + |R|)$, then each segment $S' = [i, j]$ with $i \in L$ and $j \in R$ is δ -monochromatic.

Proof. We show that if $|S| \geq \frac{1}{1-\delta}(M + |L| + |R|)$, then the segment N contains many processes with same initial color, which implies the lemma, since every segment S' contains N .

Let \mathcal{M}' be the set of intervals I such that $\text{core}(I) \cap N \neq \emptyset$. Then, $\mathcal{M}' \subseteq \mathcal{M}$. Otherwise, w.l.o.g. there exists an interval I with center $e_i < e_\ell$ and $\text{core}(I) \cap N \neq \emptyset$. But, then $L \subseteq \text{core}(I)$, which means L should have been dominated at the time when I was active.

We apply Lemma 4.7 on N and receive a set of $\bar{\delta}$ -monochromatic (non-initial) intervals $\mathcal{U} \subseteq \mathcal{M}'$, such that $U = \bigcup_{I \in \mathcal{U}} I$ and the processes in $F = N \setminus U$ have the same initial color. Then,

$$|F| = |N| - |U| = |S| - (|U| + |L| + |R|) .$$

Since $\mathcal{U} \subseteq \mathcal{M}' \subseteq \mathcal{M}$, and therefore $|U| \leq M$, we conclude:

$$|F| \geq |N| - (M + |L| + |R|) \geq |S| - (1 - \delta)|S| = \delta|S| \geq \delta|S'|$$

Then, there are at least $\delta|S'|$ processes of the same color which means that segment S' is δ -monochromatic. \square

4.4. Correctness

In this section we show that at any time step, each server contains at most $(3 + 2\epsilon)k$ processes. A crucial ingredient for this is, that each cluster has a bounded size.

Lemma 4.9. *Let A and B be two adjacent active intervals with cut-edges e_a and e_b , respectively. The size of the slice S between e_a and e_b is at most $|A| + |B| - 2 + (2 - \bar{\delta})/\bar{\delta} \cdot k$.*

Proof. W.l.o.g. let $A = [a_\ell, a_r]$ and $B = [b_\ell, b_r]$ with $0 \leq a_\ell < b_\ell < n$. If the intervals intersect ($a_r > b_\ell$) we are done. Otherwise we analyze the segment $N = [a_r + 1, b_\ell - 1]$. Since N does not intersect any active interval we can apply Lemma 4.7, which states, that there is a $\frac{\bar{\delta}}{2-\bar{\delta}}$ -monochromatic segment N' with $N \subseteq N'$. Thus, N' consists of at most k processes of some color c and at most $(1 - \frac{\bar{\delta}}{2-\bar{\delta}})k$ processes of other color. Hence, $|N| \leq |N'| \leq k + (1 - \frac{\bar{\delta}}{2-\bar{\delta}})|N'| \leq \frac{2-\bar{\delta}}{\bar{\delta}}k$. Then, the slice between e_a and e_b contains N , at most $|A| - 1$ processes of A and at most $|B| - 1$ processes of B , which is at most $|A| + |B| - 2 + \frac{2-\bar{\delta}}{\bar{\delta}}k$. \square

Corollary 4.10. *A singleton cluster has size at most $(3 + 2(1 - \bar{\delta})/\bar{\delta}) \cdot k$.*

Proof. Since the size of each interval is at most $k + 1$, we get that each singleton slice has size at most $2k + (2 - \bar{\delta})/\bar{\delta} \cdot k = (3 + 2(1 - \bar{\delta})/\bar{\delta}) \cdot k$. \square

Observation 4.11. *Let S and T be $\frac{3}{4}$ -monochromatic segments with majority color c . Then, S and T are both contained in the color c cluster.*

Lemma 4.12. *The size of a color c cluster is at most $2k$.*

Proof. A color c cluster contains at most k processes of color c . Since the slices in this cluster are all at least $1/2$ -monochromatic, the number of processes of other colors in this cluster is at most k . Thus, the overall number of processes in a color c cluster is at most $2k$. \square

Lemma 4.13. *The scheduling algorithm produces an assignment of processes to servers, such that the load of each server is at most $(3 + 2\epsilon)k$, for $\bar{\delta} \geq 2/(2 + \epsilon)$.*

Proof. By construction, the scheduling algorithm ensures that no server has load greater than $(D + \epsilon)k$, where Dk is the maximum slice size. Due to Corollary 4.10 and Lemma 4.12 we know that $D \leq 3 + 2(1 - \bar{\delta})/\bar{\delta}$, which is at most $3 + \epsilon$ for $\bar{\delta} \geq 2/(2 + \epsilon)$. \square

4.5. Cost Analysis

4.5.1. Cost of Intervals

We view parts of the cost of the online algorithm as associated with intervals, as follows. For an interval I we use $\text{cost}_{\text{hit}}(I)$ to denote the communication cost that the online algorithm experiences on the cut-edge maintained by the interval, we use $\text{cost}_{\text{move}}(I)$ to denote the cost for moving the cut-edge within the interval, and define $\text{cost}(I) := \text{cost}_{\text{hit}}(I) + \text{cost}_{\text{move}}(I)$ to be the *cost of the interval*. Observe that the cost of merging neighboring slices when deactivating an interval is not counted in $\text{cost}(I)$.

Observation 4.14. *For an initial interval $\text{cost}(I) = 0$.*

We use $\text{OPT}(I)$ to denote the cost that the optimum algorithm experiences for migrating processes of I , or for communicating along edges in I .

Lemma 4.15. *We have the following lower bounds on the optimum cost of an interval:*

1. *A non-initial interval I fulfills $\text{OPT}(I) \geq \frac{1}{2}(1 - \bar{\delta})|I|$.*
2. *Active intervals fulfill $\text{OPT}(I) \geq \frac{1}{2} \min_e x_e$, where x_e is the current request vector.*
3. *Inactive intervals fulfill $\text{OPT}(I) \geq \frac{1}{2} \min_e x'_e$, where x'_e is the request vector at the time I became inactive.*

Proof. For the first part I is non-initial, which means that it has grown before. Let I' denote the interval I before the most recent growth step. The interval I' was grown because we had $\min_{e \in I'} x_e \geq (1 - \bar{\delta})|I'|$. This means if OPT has a cut edge inside I' it experiences cost at least $\min_{e \in I'} x_e \geq (1 - \bar{\delta})|I'|$; if not it had to recolor at least $(1 - \bar{\delta})|I'|$ processes in I' because the interval I' is not $\bar{\delta}$ -monochromatic. Consequently we get

$$\text{OPT}(I) \geq \text{OPT}(I') \geq (1 - \bar{\delta})|I'| \geq \frac{1}{2}(1 - \bar{\delta})|I|, \quad (1)$$

where the last step follows because the growth step at most doubles the length of an interval.

The second part trivially holds for initial intervals, as the left hand side is 0 in this case. Intervals that are non-final (i.e., they did not reach the maximum size $k + 1$ yet), fulfill $\min_{e \in I} x_e \leq (1 - \bar{\delta})|I|$, as otherwise we would grow the interval. Then Equation 1 directly implies the lemma. Final intervals have size $k + 1$ and therefore the optimum algorithm must have a cut-edge inside. Hence, $\text{OPT}(I) \geq \min_{e \in I} x_e$ for such intervals. The third part immediately follows from Part 2. \square

Lemma 4.16. *For any interval $\text{cost}(I) \leq \mathcal{O}(1/(1 - \bar{\delta}) \cdot \log k) \text{OPT}(I)$.*

Proof. Clearly, the lemma holds, if I is an initial interval, since $\text{cost}(I) = 0$. Let I be a non-initial interval and let x be either the current request vector (if I is still active) or the request vector at the time I was deactivated. Due to Lemma 4.3 we know that I has cost at most $E[\text{cost}(I)] \leq \mathcal{O}(1) \min_e(x_e) + \mathcal{O}(\log |I|) |I|$. Using Lemma 4.15 we derive that $E[\text{cost}(I)] \leq \mathcal{O}(\frac{1}{1-\bar{\delta}} \log k) \text{OPT}(I)$. \square

4.5.2. Cost of the Algorithm

The overall cost of the algorithm consist of five parts. The first part is the *communication cost*, which we denote by cost_{hit} .

The next three parts are migration costs due to the clustering algorithm. Whenever a process is moved to an existing cluster the scheduling algorithm *may* have to also move this process in order to guarantee that all processes in a group/cluster are scheduled on the same server. Therefore we define the following costs.

- *Moving cost* $\text{cost}_{\text{move}}$: when an interval moves a cut-edge by a distance of d , d processes switch the slice that they belong to. This increases the *moving cost* of the clustering algorithm by d .
- *Merging cost* $\text{cost}_{\text{merge}}$: when two slices S_s and S_ℓ with $|S_s| \leq |S_\ell|$ are merged the processes in the smaller slice are moved to the cluster of the larger slice, which results in a cost of at most $|S_s|$.
- *Monochromatic cost* $\text{cost}_{\text{mono}}$: when a slice S becomes $\frac{3}{4}$ -monochromatic for some color c (and it was not $\frac{3}{4}$ -monochromatic before) we move it to the color c cluster at a cost of $|S|$.

Observe that there is no cost if we remove a slice from a color c cluster to form a singleton cluster, as this does not put a constraint on the scheduling algorithm to move any processes.

The final part of the cost is the migration cost caused by direct actions of the scheduling procedure to enforce capacity constraints. We call this the *rebalancing cost* cost_{bal} .

Observation 4.17. *The communication cost and moving cost are given by the interval cost, this means $\text{cost}_{\text{hit}} = \sum_I \text{cost}_{\text{hit}}(I)$, and $\text{cost}_{\text{move}} = \sum_I \text{cost}_{\text{move}}(I)$*

Lemma 4.18. *The merge cost $\text{cost}_{\text{merge}}$ of the clustering algorithm is at most $\mathcal{O}(1/(1 - \bar{\delta}) \cdot \log k) \sum_{I \in \mathcal{I}} \text{OPT}(I)$.*

Proof. For each non-initial interval I we create an account $\text{cost}_{\text{merge}}(I)$. We show how to distribute the merge costs among these accounts such that

$$\text{cost}_{\text{merge}} \leq \sum_{I \in \mathcal{I}} \text{cost}_{\text{merge}}(I) \leq \sum_{I \in \mathcal{I}} \mathcal{O}(\log k)|I| .$$

Since for every non-initial interval $\text{OPT}(I) \geq \frac{1}{2}(1 - \bar{\delta})|I|$ according to Lemma 4.15, the lemma follows.

Merging costs arise only if we deactivate an interval and remove its cut-edge. This could happen in two cases.

I) First, during a growth step of an interval I , some intervals might become dominated by I , i.e., they are completely inside I . We deactivate these intervals and remove their corresponding cut edges. However, an interval that gets dominated by I must have its cut edge inside I . But then, the cost for merging the slices due to the deactivation of dominated (by I) intervals is at most $|I|$. We charge I the cost $|I|$. Let I_1, I_2, \dots, I_m denote the growth phases of I , i.e., $I_m = I$ and $|I_i| = 2^i$ (unless of course I is a final interval, where $|I| = |I_m| = k + 1$). We derive, that an interval is charged for Case I at most $\sum_i |I_i| \leq \mathcal{O}(1)|I|$.

II) Second, during a growth step of an interval J , it may become $\bar{\delta}$ -monochromatic. We deactivate J and remove its cut-edge. We show how to distribute the cost for the slice merge.

Let $L = [a, b]$, $J = [c, d]$ and $R = [e, f]$ be adjacent active intervals with cut edges e_ℓ , e_j and edges e_r , respectively. Let S' be the slice between e_ℓ and e_j , and T' the slice between e_j and e_r . Furthermore, let $S = [a, d]$ and $T = [c, f]$. Clearly, $|S'| \leq |S|$ and $|T'| \leq |T|$. Assume interval J becomes $\bar{\delta}$ -monochromatic, i.e., we have to eliminate the cut-edge e_j and merge the incident slices S' and T' . If both S' and T' are $\frac{3}{4}$ -monochromatic, then due to Observation 4.11 the slices already reside in the same cluster, such that no cost is incurred. Otherwise, we move the smaller slice to the cluster of the bigger slice and pay $X = \min\{|S'|, |T'|\}$. We assume that either S' or T' is not $\frac{3}{4}$ -monochromatic.

We introduce additional notation. For adjacent active intervals I and J with centers $e_i = (i, i + 1)$ and $e_j = (j, j + 1)$, respectively, we define the *center slice* between I and J as the segment $C = [i + 1, j]$. We say that the active intervals I and J are *incident* to the center slice C . Furthermore, we say an inactive interval I with its center in C is also incident to C . Notice, that an inactive interval is incident to exactly one center slice and an active interval is incident to exactly two center slices.

Let S_c be the center slice between L and J , and let T_c be the center slice between J and R , respectively. We distribute the cost X to the intervals incident to the center slices S_c and T_c .

We show that each interval I is charged at most $\mathcal{O}(\log k)|I|$. The basic argument is that whenever an interval I is charged a cost $\mathcal{O}(|I|)$, one of its incident center slices doubles its size. Observe, that due to Corollary 4.10 and Lemma 4.12 the size of a center slice is at most $\mathcal{O}(k)$. Furthermore, an interval is incident to at most two center slices. Hence, if an interval is charged $\mathcal{O}(\log k)|I|$ merge cost, then its incident center slices must reach the maximum possible size.

Observe, that J cannot be an initial interval, since it became $\bar{\delta}$ -monochromatic. However, if L or R are initial intervals, we cannot charge them. Instead we charge J the additional cost (which is at most constant).

Assume, w.l.o.g that $|S_c| \leq |T_c|$. We distinguish between two cases:

- A) Assume that S' is not $\frac{3}{4}$ -monochromatic or $L \cap J \neq \emptyset$. Let \mathcal{M}_S be the set of non-initial intervals with centers in S and let $M = \sum_{I \in \mathcal{M}_S} |I|$. Clearly, $X \leq |S'| \leq |S|$. If $L \cap J \neq \emptyset$, then $X \leq |S| \leq |L| + |J|$. If S' is not $\frac{3}{4}$ -monochromatic and $L \cap J = \emptyset$, then we apply Lemma 4.8 on L and J and derive that $X \leq |S| < 4(M + |J| + |L|)$.

We distribute the cost X among the intervals in $\mathcal{M}_S \cup \{J, L\}$. We charge each interval $I \in \mathcal{M}_S \cup \{J, L\}$ the cost $4|I|$. If L is an initial interval, then we charge instead J the additional constant cost. Then, $4(M + |J| + |L|) \geq X$, i.e., we distributed all cost X .

Now, we fix an interval $I \in \mathcal{M}_S \cup \{J, L\}$ and assume that I does not grow during future time steps (i.e., we analyze a fixed growth phase of an interval). After the deactivation of J , the interval $I \in \mathcal{M}_S \cup \{J, L\}$ is incident to the center slice $Z = T_c \cup S_c$ with $|Z| \geq 2|S_c|$. Thus, for I one of its incident center slices at least doubles its size. Hence, if we charge I $\mathcal{O}(\log k)$ times (each time $|I|$) for Case A, its incident center slices reach their maximum size. We conclude that I pays at most $\mathcal{O}(\log k)|I|$ for Case A.

- B) Now, assume that S' is $\frac{3}{4}$ -monochromatic and $L \cap J = \emptyset$. Since S' is $\frac{3}{4}$ -monochromatic, then T' must be non- $\frac{3}{4}$ -monochromatic. Let \mathcal{M}_T be the set of non-initial intervals with centers in T and let $M = \sum_{I \in \mathcal{M}_T} |I|$. If $J \cap R \neq \emptyset$, then clearly, $X \leq |T| \leq |J| + |R|$. Otherwise, we apply Lemma 4.8 on J and R and derive that $X \leq |T| < 4(M + |R| + |J|)$.

We distribute the cost X among the intervals in $\mathcal{M}_T \cup \{J, R\}$. We charge each interval $I \in \mathcal{M}_T \cup \{J, R\}$ the cost $4\frac{|I|}{|T|}X$. If R is an initial interval, we charge instead J the additional constant cost. Then, $\sum_I 4\frac{|I|}{|T|}X = \frac{4}{|T|}(M + |J| + |R|)X \geq X$, i.e., we distributed all cost X .

Now, we fix an interval $I' \in \mathcal{M}_T \cup \{J, R\}$ and assume that I' does not grow during future time steps (i.e., we analyze a fixed growth phase of an interval). After the deactivation of J , I' is incident to the center slice $Z = T_c \cup S_c$. Since $L \cap J = \emptyset$, then $X \leq |S| \leq 2|S_c|$, i.e., the center slice Z increases its size by at least $\frac{1}{2}X$. Now, assume that I' is charged repeatedly $4\frac{|I'|}{|T_1|}X_1, 4\frac{|I'|}{|T_2|}X_2, \dots$ for Case B. Clearly, $|T| = |T_1| \leq |T_2| \leq \dots$, since due to merges the segment T only grows. Assume, that at some point I' experiences at least $8|I'|$ cost due

to Case B. Then, $8|I'| \leq \sum_i 4 \frac{|I'|}{|T_i|} X_i \leq 8 \frac{|I'|}{|T|} \sum_i \frac{1}{2} X_i$, i.e., $|T| \leq \sum_i \frac{1}{2} X_i := Y$. Then, we know that the center slice Z grows by at least $Y \geq |T| \geq |T_c|$, which means that $|Z| \geq 2|T_c|$, i.e., the size of the center slice Z doubles, while we charge interval I' at most $\mathcal{O}(1)|I'|$. Hence, if we charge I' overall $\mathcal{O}(\log k)|I'|$ merging cost, its incident center slices reach their maximum size. We conclude, that interval I' pays for Case B at most $\mathcal{O}(\log k)|I'|$.

Let I be an interval and let I_1, I_2, \dots, I_m denote the growth phases of I , i.e., $I_m = I$ and $|I_i| = 2^i$ (unless of course I is a final interval, where $|I| = |I_m| = k + 1$). From the above two cases (A and B) we derive that during each growth phase I_i is charged at most $\mathcal{O}(\log k)|I_i|$. Thus, the overall cost that we charge I is at most $\text{cost}_{\text{merge}}(I) \leq \sum_i \mathcal{O}(\log k)|I_i| \leq \mathcal{O}(\log k)|I|$.

Summarizing over all cases we conclude that for each non-initial interval I the merge cost $\text{cost}_{\text{merge}}(I)$ is at most $\mathcal{O}(\log k)|I|$. \square

Lemma 4.19. *The monochromatic cost $\text{cost}_{\text{mono}}$ of the clustering algorithm is at most $\mathcal{O}(1)(\text{cost}_{\text{move}} + \text{cost}_{\text{merge}})$.*

Proof. At the beginning, every slice is a 1-monochromatic segment and therefore contained in a colored cluster. Assume, at some point a slice S becomes $\frac{3}{4}$ -monochromatic. For S to leave a colored cluster and become $\frac{3}{4}$ -monochromatic again, S should have experienced at least $(\frac{3}{4} - \frac{1}{2})|S|$ change in size (due to moves and merges) since it left a colored cluster. The cost for migrating S to the colored cluster is $|S|$. Then, the cost for all "colored" migrations is at most $\mathcal{O}(1)$ times the overall moving and merge cost. \square

Lemma 4.20. *The rebalancing cost cost_{bal} of the algorithm is at most $\mathcal{O}(\frac{1}{\epsilon})(\text{cost}_{\text{move}} + \text{cost}_{\text{merge}} + \text{cost}_{\text{mono}})$.*

Proof. Initially, each server has load k . Assume, that after the clustering step server s has load L with $L > (D + \epsilon)k$, i.e., server s becomes imbalanced. The costs of the cluster migrations due to the rebalancing procedure are at most L . We know, that after the previous rebalancing step for server s the corresponding server had load at most D . Hence, in the time between the previous and the current rebalancing, server s observed at least $L - D$ migrations. These migrations were due to the clustering algorithm, since the rebalancing procedure do not place more than D processes on a server. Since $\frac{L}{L-D} \leq \frac{D+\epsilon}{\epsilon} \leq \frac{D+1}{\epsilon}$ (for $\epsilon \leq 1$), we conclude that the cost of the rebalancing procedure is at most $\mathcal{O}(\frac{1}{\epsilon})(\text{cost}_{\text{move}} + \text{cost}_{\text{merge}} + \text{cost}_{\text{mono}})$ (since D is constant). \square

Lower Bound

Now, we show a lower bound on the cost of an optimum algorithm. Lemma 4.15 already gives us a lower bound for the cost of OPT on an interval I . The following lemma shows that a process is not contained in too many intervals, which allows us to extend the lower bound.

Lemma 4.21. *A process p is contained in at most $\mathcal{O}(\log k)$ intervals.*

Proof. We say an interval I has rank r , if it has performed $(r - 1)$ growth steps. Note that its length is then usually $|I| = 2^r$, unless of course I is a final interval (then its length is $k + 1$). A process p is contained in at most 2 intervals of rank 1 and at most 4 intervals of rank 2. Let I and J be two active intervals in \mathcal{I} with rank $r \geq 3$. Let I' and J' denote the intervals I and J before their most recent growth step, respectively. Assume the distance between I' 's center and J' 's center is less than $\frac{|I|}{4}$. Then $I' \subseteq J$ and $J' \subseteq I$. But then, during the last growing phase of either I or J , one of them should have dominated the other one, a contradiction. In case I or J is a monochromatic or dominated interval, we can argue the same argument about the previous growing phases of I and J , where both of them were active. Then, the centers of I and J are at least at a distance of $\frac{|I|}{8}$ apart.

This means a process p is contained in at most 8 intervals I of rank $r \geq 3$. Since there are at most $\log k + 1$ ranks, the lemma follows. \square

Lemma 4.22. *The cost of the optimal static algorithm OPT is at least $\frac{1}{\mathcal{O}(\log k)} \sum_I \text{OPT}(I)$.*

Proof. For a process p we denote by $\text{OPT}(p)$ the cost that the optimal algorithm experiences for migrating process p . Further, for an edge e we use $\text{OPT}(e)$ to denote the cost that the optimal algorithm experiences for communicating along e . Then, $\text{OPT} = \sum_p \text{OPT}(p) + \sum_e \text{OPT}(e)$.

$$\begin{aligned} \sum_I \text{OPT}(I) &= \sum_I (\sum_{p \in I} \text{OPT}(p) + \sum_{e \in I} \text{OPT}(e)) \\ &= \sum_p |\{I : p \in I\}| \text{OPT}(p) + \sum_e |\{I : e \in I\}| \text{OPT}(e) \\ &\leq \mathcal{O}(\log k) \sum_p \text{OPT}(p) + \mathcal{O}(\log k) \sum_e \text{OPT}(e) \\ &= \mathcal{O}(\log k) \text{OPT} \end{aligned}$$

The inequality is due to Lemma 4.21 \square

Now, we summarize the overall cost of our online algorithm.

Theorem 2.2. *There is a randomized algorithm that solves the dynamic balanced graph partitioning problem for ring demands with expected cost $\mathcal{O}(\frac{1}{\epsilon^2} \log^2 k) \text{OPT}$ and resource augmentation $3 + \epsilon$, where OPT is the cost of an optimal static algorithm.*

Proof. Let $\epsilon' := \min\{\frac{\epsilon}{2}, 1\}$ and $\bar{\delta} := \max\{\frac{2}{2+\epsilon}, \frac{14}{15}\}$. We execute the slicing procedure with $\bar{\delta}$ and the scheduling procedure with ϵ' parameters. Due to Observation 4.17 and Lemma 4.15 we derive that $\text{cost}_{\text{hit}} + \text{cost}_{\text{move}} \leq \sum_I \text{cost}(I) \leq \frac{2}{1-\bar{\delta}} \sum_I \text{OPT}(I)$. Furthermore, due to Lemma 4.18 we know that $\text{cost}_{\text{merge}}$ is at most $\mathcal{O}((1-\bar{\delta})^{-1} \log k) \sum_I \text{OPT}(I)$. Lemmas 4.20 and 4.19 state that $\text{cost}_{\text{mono}} + \text{cost}_{\text{bal}} \leq \mathcal{O}(1/\epsilon)(\text{cost}_{\text{move}} + \text{cost}_{\text{merge}})$. Let cost_{onl} denote the overall costs of the online algorithm. Then,

$$\begin{aligned} \text{cost}_{\text{onl}} &= \text{cost}_{\text{hit}} + \text{cost}_{\text{move}} + \text{cost}_{\text{merge}} + \text{cost}_{\text{mono}} + \text{cost}_{\text{mig}} \\ &\leq \mathcal{O}((\epsilon(1-\bar{\delta})^{-1} \log k) \sum_{I \in \mathcal{I}} \text{OPT}(I) \\ &\leq \mathcal{O}((\epsilon^{-2} \log k) \sum_{I \in \mathcal{I}} \text{OPT}(I), \quad \text{for } \bar{\delta} \geq 2/2 + \epsilon' . \end{aligned}$$

Applying Lemma 4.22 ($\sum_I \text{OPT}(I) \leq \mathcal{O}(\log k)\text{OPT}$), we derive that our costs are at most $\mathcal{O}(\log^2(k)/\epsilon^2)\text{OPT}$.

Lemma 4.13 states that for $\bar{\delta} \geq 2/(2 + \epsilon')$ the scheduling algorithm places at most $3 + 2\epsilon' = 3 + \epsilon$ processes on each server, which concludes the proof. \square

5. Conclusion

We presented the first polylogarithmically-competitive online algorithms for the balanced graph partitioning problem for a scenario where the communication pattern inherently and continuously requires inter-server communications and/or migrations. In particular, we described two different approaches, one for the static model and one for the dynamic one, and proved their competitiveness accordingly.

Our work opens several interesting directions for future research. In particular, it would be interesting to study lower bounds on the achievable competitive ratio in the two models. The main open question regards whether polylogarithmic-competitive algorithms can also be achieved under more general communication patterns.

Acknowledgements

Research supported by German Research Foundation (DFG), grant 470029389 (FlexNets), 2021-2024 and Federal Ministry of Education and Research (BMBF), grant 16KISK020K (6G-RIC), 2021-2025.

References

- [1] Jeffrey C Mogul and Lucian Popa. What we talk about when we talk about cloud network performance. *ACM SIGCOMM Computer Communication Review*, 42(5):44–48, 2012.
- [2] William M Mellette, Rob McGuinness, Arjun Roy, Alex Forencich, George Papen, Alex C Snoeren, and George Porter. Rotornet: A scalable, low-complexity, optical datacenter network. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 267–280, 2017.
- [3] Chen Griner, Johannes Zerwas, Andreas Blenk, Manya Ghobadi, Stefan Schmid, and Chen Avin. Cerberus: The power of choices in datacenter topology design—a throughput perspective. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 5(3):1–33, 2021.
- [4] Monika Henzinger, Stefan Neumann, Harald Raecke, and Stefan Schmid. Tight bounds for online graph partitioning. In *Proc. ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2021.

- [5] Tobias Forner, Harald Raecke, and Stefan Schmid. Online balanced repartitioning of dynamic communication patterns in polynomial time. In *Proc. SIAM Symposium on Algorithmic Principles of Computer Systems (APOCS)*, 2021.
- [6] Chen Avin, Marcin Bienkowski, Andreas Loukas, Maciej Pacut, and Stefan Schmid. Dynamic balanced graph partitioning. In *SIAM J. Discrete Math (SIDMA)*, 2019.
- [7] Maciej Pacut, Mahmoud Parham, and Stefan Schmid. Optimal online balanced graph partitioning. In *Proc. IEEE INFOCOM*, 2021.
- [8] Chen Avin, Andreas Loukas, Maciej Pacut, and Stefan Schmid. Online balanced repartitioning. In *Proc. 30th International Symposium on Distributed Computing (DISC)*, 2016.
- [9] Chen Avin, Louis Cohen, Mahmoud Parham, and Stefan Schmid. Competitive clustering of stochastic communication patterns on a ring. In *Journal of Computing*, 2018.
- [10] Monika Henzinger, Stefan Neumann, and Stefan Schmid. Efficient distributed workload (re-)embedding. In *Proc. ACM SIGMETRICS*, 2019.
- [11] Chen Avin, Louis Cohen, and Stefan Schmid. Competitive clustering of stochastic communication patterns on the ring. In *Proc. 5th International Conference on Networked Systems (NETYS)*, 2017.
- [12] Felix Hohne, Soren Schmitt, and Rob van Stee. Algorithms column 38: 2021 in review. *SIGACT News 52*, 2021.
- [13] Alexander Sergeev and Mike Del Balso. Horovod: fast and easy distributed deep learning in tensorflow. *CoRR*, abs/1802.05799, 2018.
- [14] Ahmad Faraj, Pitch Patarasuk, and Xin Yuan. A study of process arrival patterns for mpi collective operations. *International Journal of Parallel Programming*, 36(6):543–570, 2008.
- [15] Carl Yang. Tree-based allreduce communication on mxnet. 2018.
- [16] Harald Räcké, Stefan Schmid, and Ruslan Zabrodin. Approximate dynamic balanced graph partitioning. In *Proceedings of the 34th ACM Symposium on Parallelism in Algorithms and Architectures*, pages 401–409, 2022.
- [17] Chen Avin, Louis Cohen, Mahmoud Parham, and Stefan Schmid. Competitive clustering of stochastic communication patterns on a ring. In *Journal of Computing*, 2018.
- [18] Dorit S Hochbaum and David B Shmoys. Using dual approximation algorithms for scheduling problems theoretical and practical results. *Journal of the ACM (JACM)*, 34(1):144–162, 1987.

- [19] Peter Sanders, Naveen Sivadasan, and Martin Skutella. Online scheduling with bounded migration. *Math. Oper. Res.*, 34(2):481–498, 2009.
- [20] Björn Feldkord, Matthias Feldotto, Anupam Gupta, Guru Guruganesh, Amit Kumar, Sören Riechers, and David Wajc. Fully-dynamic bin packing with little repacking. In *ICALP*, pages 51:1–51:24, 2018.
- [21] Allan Borodin, Nathan Linial, and Michael E. Saks. An optimal on-line algorithm for metrical task system. *J. ACM*, 39(4):745–763, oct 1992.
- [22] Yair Bartal, Avrim Blum, Carl Burch, and Andrew Tomkins. A polylog(n)-competitive algorithm for metrical task systems. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing, STOC '97*, page 711–719, New York, NY, USA, 1997. Association for Computing Machinery.
- [23] Amos Fiat and Manor Mendel. Better algorithms for unfair metrical task systems and applications. *SIAM Journal on Computing*, 32(6):1403–1422, 2003.
- [24] Yair Bartal, Béla Bollobás, and Manor Mendel. Ramsey-type theorems for metric spaces with applications to online problems. *Journal of Computer and System Sciences*, 72(5):890–921, 2006. Special Issue on FOCS 2001.
- [25] Sébastien Bubeck, Michael B. Cohen, James R. Lee, and Yin Tat Lee. Metrical task systems on trees via mirror descent and unfair gluing. *SIAM Journal on Computing*, 50(3):909–923, 2021.
- [26] Sébastien Bubeck, Christian Coester, and Yuval Rabani. The randomized k -server conjecture is false!, 2022.
- [27] A. Blum, C. Burch, and A. Kalai. Finely-competitive paging. In *40th Annual Symposium on Foundations of Computer Science (Cat. No.99CB37039)*, pages 450–457, 1999.

A. Smooth Minimum Approximation

Let $x = (x_1, x_2, \dots, x_n)^T \in \mathbb{R}^n$ be an n dimensional vector. The function $\text{smin}(x) := -\ln(\sum_i e^{-x_i})$ smoothly approximates the minimum function $\min(x) = \min(x_1, x_2, \dots, x_n)$.

Fact A.1.

(i) The function $\text{smin}(x)$ approximates the minimum up to an additive term:

$$\min(x) - \ln n \leq \text{smin}(x) \leq \min(x)$$

(ii) The gradient $\nabla \text{smin}(x)$ is a probability distribution.

Proof.

(i) Let $x_* = \min(x)$. Using facts $\sum_i e^{-x_i} \geq e^{-x_*}$ and $\sum_i e^{-x_i} \leq ne^{-x_*}$, we get

$$x_* - \ln n = -\ln(ne^{-x_*}) \leq -\ln(\sum_i e^{-x_i}) \leq -\ln(e^{-x_*}) = x_*$$

(ii) The i -th component of the gradient is $\nabla \text{smin}(x)_i = \frac{e^{-x_i}}{\sum_i e^{-x_i}} \geq 0$. The sum of all components is exactly 1. \square

The change of the smin function is well approximated by its gradient. Formally:

Lemma A.2.

(i) For all vectors $x, \ell \geq 0$, $\ell_i \leq 1$,

$$\text{smin}(x + \ell) - \text{smin}(x) \geq \frac{1}{2} \nabla \text{smin}(x)^T \ell$$

(ii) For all vectors $x, \ell \geq 0$,

$$\|\nabla \text{smin}(x + \ell) - \nabla \text{smin}(x)\|_1 \leq 2 \nabla \text{smin}(x)^T \ell$$

Proof.

(i) Let $A = \sum_{i=1}^n e^{-x_i}$. Then,

$$\begin{aligned} \text{smin}(x + \ell) - \text{smin}(x) &= -\ln\left(\frac{1}{A} \sum_i e^{-(x_i + \ell_i)}\right) \\ &= -\ln\left(\frac{1}{A} \sum_i e^{-x_i} + e^{-x_i}(e^{-\ell_i} - 1)\right) \\ &= -\ln\left(1 + \frac{1}{A} \sum_i e^{-x_i}(e^{-\ell_i} - 1)\right) \\ &\geq -\frac{1}{A} \sum_i e^{-x_i}(e^{-\ell_i} - 1) \\ &= \frac{1}{A} \sum_i e^{-x_i}(1 - e^{-\ell_i}) \\ &\geq \frac{1}{A} \sum_i e^{-x_i} \frac{\ell_i}{2} = \frac{1}{2} \nabla \text{smin}(x)^T \ell \end{aligned}$$

The first inequality follows by applying the fact $\ln(1+z) \leq z$. The second inequality follows because $1 - e^{-z} \geq \frac{z}{2}$, for $0 \leq z \leq 1$.

- (ii) Since both $\nabla \text{smin}(x + \ell)$ and $\nabla \text{smin}(x)$ are probability distributions, the sum of their components is 1. Consider the change of the components going from x to $x + \ell$. The sum of increasing components must equal the sum of decreasing components. Now, let I be the set of indices that are decreasing. Consider such a component $i \in I$. Then,

$$\begin{aligned} \nabla \text{smin}(x)_i - \nabla \text{smin}(x + \ell)_i &= \frac{e^{-x_i}}{\sum_j e^{-x_j}} - \frac{e^{-(x_i + \ell_i)}}{\sum_j e^{-(x_j + \ell_j)}} \\ &\leq \frac{e^{-x_i} - e^{-(x_i + \ell_i)}}{\sum_j e^{-x_j}} \\ &= \frac{e^{-x_i}(1 - e^{-\ell_i})}{\sum_j e^{-x_j}} \\ &\leq \frac{e^{-x_i} \ell_i}{\sum_j e^{-x_j}} \end{aligned}$$

In the first inequality we used that since $\ell \geq 0$, $\sum_j e^{-(x_j + \ell_j)} \leq \sum_j e^{-x_j}$ must hold. In the last inequality we used the fact that $1 - e^{-\ell_i} \leq \ell_i$. Then,

$$\begin{aligned} \|\nabla \text{smin}(x + \ell) - \nabla \text{smin}(x)\|_1 &= 2 \sum_{i \in I} \nabla \text{smin}(x)_i - \nabla \text{smin}(x + \ell)_i \\ &\leq 2 \sum_{i \in I} \frac{e^{-x_i} \ell_i}{\sum_j e^{-x_j}} \\ &\leq 2 \sum_{i=1}^n \frac{e^{-x_i} \ell_i}{\sum_j e^{-x_j}} \\ &= 2 \nabla \text{smin}(x)^T \ell \end{aligned}$$

□

Now we generalize the smin function. We define a function

$$\text{smin}_c(x) := c \cdot \text{smin}\left(\frac{1}{c}x\right).$$

for a constant $c \geq 1$. The advantage of the $\text{smin}_c(x)$ function over $\text{smin}(x)$ is that we can control the gradients change using the constant c . On the other side the approximation of the minimum becomes worse. The following properties apply to $\text{smin}_c(x)$:

Lemma A.3.

- (i) The function $\text{smin}_c(x)$ approximates the minimum up to an additive term:

$$\min(x) - c \ln n \leq \text{smin}_c(x) \leq \min(x)$$

(ii) $\nabla \text{smin}_c(x)$ is a probability distribution. Furthermore,

$$\nabla \text{smin}_c(x) = \nabla \text{smin}\left(\frac{1}{c}x\right) .$$

(iii) For all vectors $x, \ell \geq 0$, $\ell_i \leq 1$,

$$\text{smin}_c(x + \ell) - \text{smin}_c(x) \geq \frac{1}{2} \nabla \text{smin}_c(x)^T \ell$$

(iv) For any vectors $x, \ell \geq 0$,

$$\|\nabla \text{smin}_c(x + \ell) - \nabla \text{smin}_c(x)\|_1 \leq \frac{2}{c} \nabla \text{smin}_c(x)^T \ell$$

Proof. Let $x' = \frac{1}{c}x$ and $\ell' = \frac{1}{c}\ell$. Since $c \geq 1$, we have that $x', \ell' \geq 0$ and $\ell_i \leq 1$.

(i) Using Fact A.1 we obtain:

$$\begin{aligned} \text{smin}_c(x) &= c \text{smin}\left(\frac{1}{c}x\right) \leq n \min\left(\frac{1}{c}x\right) = \min(x) \\ \text{smin}_c(x) &= c \text{smin}\left(\frac{1}{c}x\right) \geq n(\min\left(\frac{1}{c}x\right) - \ln n) = \min(x) - c \ln n \end{aligned}$$

(ii) Using Fact A.1 we obtain:

$$\begin{aligned} \nabla \text{smin}_c(x)_i &= \frac{\partial \text{smin}_c(x)}{\partial x_i} \\ &= c \frac{\partial \text{smin}(x')}{\partial x'_i} \frac{\partial x'_i}{\partial x_i} \\ &= \frac{\partial \text{smin}(x')}{\partial x'_i} \\ &= \nabla \text{smin}\left(\frac{1}{c}x\right)_i \end{aligned}$$

(iii) Using Lemma A.2 (i) we obtain:

$$\begin{aligned} \text{smin}_c(x + \ell) - \text{smin}_c(x) &= c(\text{smin}(x' + \ell') - \text{smin}(x')) \\ &\geq c \frac{1}{2} \nabla \text{smin}(x')^T \ell' \\ &= \frac{1}{2} \nabla \text{smin}_c(x)^T \ell \end{aligned}$$

(iv) Using Lemma A.2 (ii) we obtain:

$$\begin{aligned} \|\nabla \text{smin}_c(x + \ell) - \nabla \text{smin}_c(x)\|_1 &= \|\nabla \text{smin}(x' + \ell') - \nabla \text{smin}(x')\|_1 \\ &\leq 2 \nabla \text{smin}(x')^T \ell' \\ &= \frac{2}{c} \nabla \text{smin}_c(x)^T \ell \end{aligned}$$

□