# Approximate Dynamic Balanced Graph Partitioning

Harald Räcke
Technical University of Munich
Munich, Germany
raecke@in.tum.de

Stefan Schmid
Technical University of Berlin
Berlin, Germany
University of Vienna
Vienna, Austria
stefan.schmid@tu-berlin.de

Ruslan Zabrodin
Technical University of Munich
Munich, Germany
zabrodin@in.tum.de

## ABSTRACT

Networked systems are increasingly flexible and reconfigurable. This enables demand-aware infrastructures whose resources can be adjusted according to the traffic pattern they currently serve.

This paper revisits the dynamic balanced graph partitioning problem, a generalization of the classic balanced graph partitioning problem. We are given a set $P$ of $n = k\ell$ processes which communicate over time according to a given request sequence $\sigma$. The processes are assigned to $\ell$ servers (each of capacity $k$), and a scheduler can change this assignment dynamically to reduce communication costs, at cost $\alpha$ per node move. Avin et al. showed an $\Omega(k)$ lower bound on the competitive ratio of any deterministic online algorithm, even in a model with resource augmentation, and presented an $O(k \log k)$-competitive online algorithm. We study the offline version of this problem where $\sigma$ is known to the algorithm.

Our main contribution is a polynomial-time algorithm which provides an $O(\log n)$-approximation with resource augmentation. Our algorithm relies on an integer linear program formulation in a metric space with spreading constraints. We relax the formulation to a linear program and employ Bartal's clustering algorithm in a novel way to round it.

## CCS CONCEPTS

• **Theory of computation → Scheduling algorithms**.

## KEYWORDS

Approximation algorithms; LP rounding; clustering; graph partitioning

**ACM Reference Format:**
Harald Räcke, Stefan Schmid, and Ruslan Zabrodin. 2022. Approximate Dynamic Balanced Graph Partitioning. In *Proceedings of the 34th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA '22), July 11–14, 2022, Philadelphia, PA, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3490148.3538563

## 1 INTRODUCTION

Distributed cloud applications often generate a large amount of network traffic [9]. In order to reduce communication costs and optimally serve such applications, it can make sense to allocate frequently communicating nodes closer to each other, e.g., by collocating them on the same server or in the same rack in the datacenter. Such demand-aware optimizations are enabled by the increasing resource allocation flexibilities available in modern virtualized infrastructures, and are empirically supported by the rich spatial and temporal structure featured by communication patterns of data-intensive applications [5].

This paper is motivated by the fundamental question how such dynamic optimizations can be performed while minimizing reconfiguration costs. In particular, we revisit the dynamic balanced graph partitioning problem introduced by Avin et al. at DISC 2016 [6] and recently reviewed at SIGACT News [21]. In a nutshell, in this problem, a set $P$ of $n$ processes is distributed across $\ell$ servers, each of size $k$: without augmentation, each server can host up to $k$ processes. It holds that $n = k\ell$. A sequence $\sigma$ describes the demand, that is, the communication pairs over time. If we view the communications requests as the edges of a graph that appear over time, then this can be seen as a dynamic graph partitioning problem.

The goal is to devise a scheduling algorithm which strikes a balance between the benefits and the costs of process migrations. The cost model is as follows: if a communication request is served remotely, i.e., between nodes mapped to different servers, it incurs a communication cost of 1; communication requests between nodes located on the same server are free of cost. Before the cost for the current request is paid, an algorithm has the option to migrate nodes at a cost of $\alpha > 1$ for each node move. The problem can hence be seen as a symmetric version of caching: two nodes can be "cached together" on any server.

Prior work [2, 6, 27] focused on an online setting where $\sigma$ is revealed to the algorithm one request at a time. In particular, it has been shown that the competitive ratio of any deterministic online algorithm is at least linear: in the worst case, the online algorithm pays at least $\Omega(k)$ times more than the offline algorithm (even if server capacities are augmented by a constant factor). This lower bound even holds in a scenario with augmentation, that is, if the online algorithm can use larger servers than the offline algorithm. Today, it is still an open question whether an $O(k)$-competitive algorithm exists. The best-known online algorithm is $O(k \log k)$ [2, 6, 27] competitive.

This paper initiates the study of offline algorithms for this problem: the design of efficient approximation algorithms which know the entire sequence $\sigma$. Besides being a fundamental theoretical research question on its own right, our study of offline algorithms is also motivated by the fact that communication patterns in datacenters are often fairly predictable [22].


*SPAA '22, July 11–14, 2022, Philadelphia, PA, USA*

ACM ISBN 978-1-4503-9146-7/22/07…$15.00
https://doi.org/10.1145/3490148.3538563

## 1.1 Our Contributions

Our main contribution is a polynomial-time algorithm which provides an $O(\frac{1}{\epsilon^2}\log(n))$ approximation with resource augmentation factor $2+\epsilon$.

The proposed algorithm is randomized and formulates the optimization problem as an integer linear program in a metric space, using spreading constraints. This integer linear program is relaxed to a linear program (LP), for which we obtain a separation oracle, allowing us to solve the LP in polynomial time. We interpret the LP solutions as trajectories in the metric space. We subsequently round the LP solutions efficiently, using Bartal's clustering algorithm in a novel way. Unfortunately, we cannot apply Bartal's algorithm directly because the number of vertices in our graph depends on the number of requests $T$. Hence, a direct application would introduce a logarithmic dependency on $T$ into the approximation guarantee. In order to avoid this we partition our graph into subgraphs in an appropriate way, execute Bartal's clustering algorithm on each subgraph and combine the results, achieving a competitive ratio that is not dependent on the number of requests.

## 1.2 Novelty and Putting Things into Perspective

Our problem can be seen as a generalization of a static graph partitioning problem: As our model does not assume initial locations for the processes, for $\alpha \to \infty$ (moving processes is very expensive), the problem turns into a balanced graph partitioning problem [1]. Even in this static version the problem does not allow for a finite approximation ratio without augmentation [1]. Therefore we clearly require some form of augmentation in our more general model to achieve any reasonable guarantees.

Existing algorithms for the balanced graph partitioning problem are usually based on one of the following two algorithmic approaches: spreading metrics or dynamic programming. Algorithms that use spreading metrics typically require augmentation at least 2 [11, 12], whereas algorithms that use dynamic programming achieve augmentation $1+\epsilon$ [1, 14]. Our techniques in this paper are based on spreading metrics, hence the achieved augmentation of roughly 2 is natural. For the balanced graph partitioning problem there exists an $O(\log n)$-approximation algorithm for $1+\epsilon$ augmentation [14] as well as an $O(\sqrt{\log n \log k})$-approximation algorithm for augmentation 2 [24]. For comparison, our algorithm for the more general model obtains an $O(\log n)$ approximation with an augmentation of $2+\epsilon$.

A natural open problem is whether the violation of our algorithm could be improved from $2+\epsilon$ to $1+\epsilon$ or the approximation guarantee could be improved from $O(\log n)$ to $O(\sqrt{\log n \log k})$ by using one of the aforementioned techniques for balanced graph partitioning. However, transferring either of these approaches to our more general problem seems to be very challenging. See Section A in the appendix for a detailed discussion.

At first glance our problem may also look similar to the k-server problem [23, 25, 26] because we can model the processes as "servers" traveling in a metric space. But unlike the traditional k-server model we are not allowed to have many servers close to each other. This additional constraint completely changes the problem, and existing k-server algorithmic techniques are not applicable. Furthermore, the static version of the k-server problem can be solved optimally in polynomial time, which is clearly not the case for our problem.

## 1.3 Additional Related Work

The dynamic balanced graph partitioning problem has been introduced by Avin et al. [2, 6]. The authors studied an online version of the problem and presented a lower bound of $\Omega(k)$ for deterministic algorithms, even in a resource augmentation model, and described a deterministic online algorithm which achieves a competitive ratio of $O(k\log k)$. The problem can be seen as a symmetric version of paging [16]. While the online algorithms in [2, 6] relies on expensive repartitioning operations and has a super-polynomial runtime, Forner et al. [17] later showed that an $O(k\log k)$ competitive ratio 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. [27] further described an $O(\ell)$-competitive online algorithm for a scenario without resource augmentation and the case where $k=3$ [10].

There also exist several results on restricted versions of the problem. In particular, Henzinger et al. [19] initiated the study of a model where it is guaranteed that the communication requests can be perfectly partitioned: that is, they are drawn from a graph whose connected components can be assigned to servers such that no connected component is split across multiple servers. As the graph is fixed and partitionable, this model is sometimes referred to as the *learning variant* of the general dynamic balanced graph partitioning problem. The authors present a deterministic exponential-time algorithm with competitive ratio $O(\ell\log\ell\log k)$ and complement their result with 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 a follow-up work, Henzinger et al. [18] improve upon these results and present deterministic and randomized algorithms which achieve (almost) tight bounds for the learning variant. In particular, they present a polynomial-time randomized algorithm achieving a polylogarithmic competitive ratio of $O(\log\ell+\log k)$, and prove 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 [20] and Sanders et al. [28]. For the deterministic learning variant without resource augmentation, Pacut et al. showed a tight bound of $\Theta(k\cdot\ell)$.

Another restricted version of the dynamic balanced graph partitioning problem has been studied in [3, 4]: here, the adversary needs to generate the communication sequence from a random distribution. That is, while the adversary can choose the distribution (and knows the deterministic online algorithm), it needs to sample communication requests from this distribution in an *i.i.d.* manner.

More generally, our model is also related to dynamic bin packing problems which allow for limited *repacking* [13]: 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 [13] do not extend to our problem.

To the best of our knowledge there are currently no results on approximation algorithms for the dynamic balanced graph partitioning problem.

### 1.4 Organization

The remainder of this paper is organized as follows. After introducing our formal model in Section 2, we first present a linear programming formulation of our problem in Section 3 and then describe our randomized rounding approximation algorithm in Section 4. We conclude our contribution in Section 5 and discuss directions for future research.

## 2 MODEL AND NOTATION

We are given a set of $n$ processes together with a sequence $\sigma = (\sigma_1, \ldots, \sigma_T)$ of communication requests between these processes, where $\sigma_t = \{p, q\}$ specifies a communication request between processes $p$ and $q$. The processes have to be scheduled on a set of $\ell$ servers, where each server has capacity $k$, i.e., each server can hold at most $k$ processes at any point in time. Let $P$ be the set of processes with $|P| = n = k\ell$.

A communication request $\sigma_t = \{p, q\}$ incurs cost of exactly 1, if $p$ and $q$ are located on different servers. We call this the *communication cost* and say that request $\sigma_t$ is served remotely. If the processes are on the same server, no cost is incurred. A migration of a process to another server induces cost $\alpha > 1$. We call this the *migration cost*. The goal is to find a mapping of processes to servers for each time step that minimizes the sum of migration and communication cost and obeys the capacity constraints.

Assigning the processes to the servers such that capacity constraints are exactly obeyed seems very challenging. Thus, we use a model with augmentation and allow to place up to $(2+\epsilon)k$ processes on a server at any point in time. (We assume w.l.o.g. that $1/k \leq \epsilon$; in addition, for technical reasons, we assume that $\epsilon \leq 1$.) A solution with augmentation is then compared to the optimal solution without augmentation.

We use the following notation for graphs throughout the paper. Let $V$ be a vertex set and $d$ a distance function for the vertices in $V$. For $v \in V$ we use $B(v, r) = \{u \in V \mid d(v, u) \leq r\}$ to denote the ball with radius $r$ around $v$, i.e., the set of vertices that are at most at a distance $r$ away from $v$. Let $G = (V, E)$ be a graph and $\{V_1, V_2, \ldots, V_z\}$ a partition of $V$, i.e., $V_i \cap V_j = \emptyset$, $i \neq j$, and $\bigcup_i V_i = V$. We say an edge $e \in E$ is *cut*, if it connects two different vertex sets, i.e., $e = \{u, v\}$ with $u \in V_i, v \in V_j, i \neq j$.

## 3 LINEAR PROGRAMMING FORMULATION

As the first step we formulate our problem statement as an integer linear program (ILP). For this purpose we view the servers $s_1, \ldots, s_\ell$ as some abstract points in a metric space, where the distance between two servers is exactly 1. Additionally, for every process $p$ and time step $t$ we introduce a point $p^t \in \{s_1, \ldots, s_\ell\}$ with the meaning that if $p^t = s_i$ then process $p$ is located at the $i$th server at time step $t$. As a consequence, the migration cost for a process $p$ at time step $t$ is $\alpha$ times the distance between $p^t$ and $p^{t-1}$ and the communication cost at time $t$ is the distance between $p^t$ and $q^t$ where $\sigma_t = \{p, q\}$ is the request at time $t$.

$$
\begin{aligned}
\min \quad & \alpha \sum_{\substack{t \geq 1 \\ p \in P}} d(p^{t-1}, p^t) + \sum_{\substack{t \geq 1 \\ \sigma_t = \{p,q\}}} d(p^t, q^t) \\
\forall p^t, q^\tau, r^\rho : \quad & d(p^t, q^\tau) \leq d(p^t, r^\rho) + d(r^\rho, q^\tau) && \text{(T)} \\
\forall t, s_i : \quad & \textstyle\sum_{p \in P} d(p^t, s_i) \geq n - k && \text{(S)} \\
\forall t, p : \quad & \textstyle\sum_{i=1}^{\ell} d(p^t, s_i) = \ell - 1 && \text{(M)} \\
\forall t, \tau, q, p : \quad & d(p^t, q^\tau) \in \{0, 1\} && \text{(I)}
\end{aligned}
$$

**Figure 1: ILP1: Integer linear program that exactly captures the model**

$$
\begin{aligned}
\min \quad & \alpha \sum_{\substack{t \geq 1 \\ p \in P}} d(p^{t-1}, p^t) + \sum_{\substack{t \geq 1 \\ \sigma_t = \{p,q\}}} d(p^t, q^t) \\
\forall p^t, q^\tau, r^\rho : \quad & d(p^t, q^\tau) \leq d(p^t, r^\rho) + d(r^\rho, q^\tau) && \text{(T)} \\
\forall S \subseteq P, p \in S, t : \quad & \textstyle\sum_{q \in S} d(p^t, q^t) \geq |S| - k && \text{(S)} \\
\forall t, \tau, q, p : \quad & d(p^t, q^\tau) \in \{0, 1\} && \text{(I)}
\end{aligned}
$$

**Figure 2: ILP2: The relaxed integer linear program**

Next, note that we are not interested in the exact metric points. We need only the distances between them. We denote by $d(p^t, q^\tau)$ the distance between points $p^t$ and $q^\tau$. All variables in our ILP will be of the form $d(p^t, q^\tau) \in \{0, 1\}$. Actually, there is only one variable $d(\{p^t, q^\tau\})$ in the ILP instead of $d(p^t, q^\tau)$ and $d(q^\tau, p^t)$, which implies symmetry by design, but for ease of notation we write $d(p^t, q^\tau)$.

ILP1 captures exactly our problem statement. We are minimizing over the traveled distance of all processes multiplied with $\alpha$ (migration cost) and the distance between communication partners (the number of requests served remotely). Because of the triangle inequality constraints (T) and symmetry (which holds by construction) we obtain a valid pseudometric, which is sufficient for our purpose. The spreading constraints (S) ensure that server capacity constraints are met. This means that only $k$ processes can reside on the same server simultaneously. The mapping constraints (M) ensure that each process is assigned to exactly one server in each time step $t$.

Notice, that not enforcing the mapping constraints can only lower the cost of the optimal solution. We eliminate server points $s_1, \ldots, s_l$ together with the mapping constraints (M) and let the ILP decide where to place processes within the metric space. Doing so we also need to adjust the spreading constraints (S). We formulate the spreading constraints in a way that ensures that there are not too many points within a ball of a fixed radius. Let ILP2 denote the relaxed integer linear program.

Consider the optimal solution of ILP1. Spreading constraints together with the mapping constraints in ILP1 imply that the spreading constraints in ILP2 are also met. Thus, this is a valid solution of ILP2. It follows, that the optimal solution cost of ILP2 is a lower bound on the optimal solution cost of ILP1. Next, we relax the integrality constraints (I) in ILP2 by replacing them with non-negativity

constraints which again might only decrease the cost of the optimal solution. We denote the resulting linear program by LP. Lemma 3.1 states that LP is computable in polynomial time.

Lemma 3.1. *An optimal solution to LP can be computed in polynomial time.*

Proof. There is a polynomial number of variables and triangle inequality constraints. For each time step $t$ and process $p \in P$ we can verify in polynomial time whether there exists a spreading constraint involving $p^t$ that is not satisfied. Consider the sorted sequence $a = (p_1, p_2, \ldots, p_n)$ of processes $p_i \in P$ that are ordered according to the increasing distance to $p$ at time step $t$. For each $S \subseteq P, |S| = m$ and $p \in S$ the expression $\sum_{q \in S} d(p^t, q^t) \geq |S| - k$ is minimized if $S$ contains the first $m$ processes from the sequence $a$. Hence, for each $p$ and $t$ we need to verify the spreading constraints at most for $n$ sets $S \subseteq P$. Thus, we obtain a separation oracle. □

# 4 ROUNDING THE LP SOLUTION

The linear program formulated in the previous section can be solved in polynomial time, but the solution we obtain is fractional. We need to round it in order to achieve a feasible integral solution. Recall that the solution of the LP provides us with distances $d(p^t, q^\tau)$ for some abstract points $p^t$ and $q^\tau$, which correspond to the positions of the processes $p$ and $q$ in metric space at time steps $t$ and $\tau$, respectively.

We first transform the LP rounding problem into a graph problem as follows (see Figure 3 for a visualization). We refer to the sequence $(p^0, p^1, p^2, \ldots, p^T)$ of points as the *trajectory* of process $p$. Thus, each trajectory consists of $T+1$ vertices and $T$ edges $e = \{p^{t-1}, p^t\}$ between consecutive trajectory vertices. We call these edges *migration edges*. Let $E_m = \{\{p^{t-1}, p^t\} \mid p \in P, 1 \leq t \leq T\}$ be the set of migration edges. Furthermore, for each communication request $\sigma_t = \{p, q\}$ we create an edge $e = \{p^t, q^t\}$. We refer to these edges as *communication edges*. Let $E_c = \{\{p^t, q^t\} \mid 1 \leq t \leq T, \sigma_t = \{p, q\}\}$ be the set of communication edges. Define the length $d(e)$ of an edge $e = \{u, w\} \in E_m \cup E_c$ as the distance $d(u, w)$. Let the cost of a communication edge be 1 and the cost of a migration edge $\alpha$. We obtain a graph $G = (V, E, d)$ with $V = \{p^t \mid p \in P, t \leq T\}$ and $E = E_m \cup E_c$. We call vertices of the form $p^t$ for $p \in P$ vertices with timestamp $t$. Let $V(t)$ denote the set of vertices with timestamp $t$, $0 \leq t \leq T$. We say the *cost of graph* $G$ is $cost(G) = \alpha \sum_{e \in E_m} d(e) + \sum_{e \in E_c} d(e)$. Note that $cost(G)$ is exactly the cost of the fractional solution of our linear program. Let $OPT$ be the cost of the optimal integral solution. Clearly, $cost(G) \leq OPT$.

We would like to partition the graph into exactly $\ell$ pieces $V_1$, $V_2$, …, $V_\ell$ such that $|V_i \cap V(t)| = k, \forall i, t$. Then, we could simply assign each piece to a server. Unfortunately, this seems very difficult. Instead we proceed in two steps. As the first step, we partition the graph into many small pieces $C = \{V_1, V_2, \ldots\}$ with the property that $|V_i \cap V(t)| \leq (1+\epsilon)k, \forall i, t$ for a fixed constant $\epsilon$. We call these pieces *clusters* and a partition of the graph $G$ into clusters such that the above property holds a *valid* clustering $C$. Let $W_c$ and $W_m$ be the number of communication edges and migration edges, respectively, that are cut by the clustering $C$. We define the *cost of clustering* $C$ by $cost_G(C) := \alpha W_m + W_c$. In the following we just write $cost(C)$ if the graph is clear from the context. We refer to this first step as the graph clustering step.

The second step is the scheduling step. The aim of this step is to decide which cluster is assigned to which server such that capacity constraints are maintained. However, this is not trivial and there might be no good way to assign the clusters from the graph clustering step. Therefore, the scheduling step first partitions the clusters from the clustering step into smaller pieces and then assigns the pieces to servers.

## 4.1 Graph Clustering Step

Given $\epsilon > 0$, our first goal is to partition the graph $G$ into clusters $V_1, V_2, \ldots$, such that $\forall i, t : |V_i \cap V(t)| \leq (1+\epsilon)k$. We begin with a lemma that states that capacity constraints are violated at most by a factor of $(1+\epsilon)$, if the diameter of each cluster is bounded.

Lemma 4.1. *Let $C$ be a cluster and $\delta$ its diameter. If $\delta \leq \frac{\epsilon}{1+\epsilon}$ then $|C \cap V(t)| \leq (1+\epsilon)k \ \forall t$.*

Proof. The set of vertices in $C$ with timestamp $t$ is exactly $C \cap V(t)$. The crucial observation is that due to the spreading constraints for each $u \in C \cap V(t)$ we have

$$\delta |C \cap V(t)| \geq \sum_{v \in C \cap V(t)} d(u, v) \geq |C \cap V(t)| - k.$$

Rearranging, we get $|C \cap V(t)| \leq \frac{k}{1-\delta}$ for $\delta < 1$, which we will ensure. We want to obtain $|C \cap V(t)| \leq (1+\epsilon)k$. We achieve this by setting $\frac{k}{1-\delta} \leq (1+\epsilon)k$. Solving for $\delta$ gives $\delta \leq \frac{\epsilon}{1+\epsilon} < 1$. □

As a consequence, an assignment of one such cluster to a server would violate the capacity constraints by at most $(1+\epsilon)k$ (although we do not know at the moment how to schedule all the clusters). A further issue is that the clustered graph should incur little cost, i.e., the number of migration and communication edges between different clusters should be as small as possible. A candidate algorithm for the graph clustering step is Bartal's well known algorithm for low diameter decomposition of graphs [7, 8]. However, if we apply Bartal's algorithm directly, our approximation ratio would depend on $\log N$, where $N$ is the number of vertices in our graph. Because $N = n(T+1)$ in our case and we are aiming for a factor $O(\log n)$ approximation, we need to modify Bartal's algorithm in order to achieve our goal. In particular, we will enforce a lower bound on the diameter of each cluster, which in turn will give us an upper bound on the number of clusters.

Denote by $M = \sum_{p \in P, t \geq 1} d(p^{t-1}, p^t)$ the overall distance traveled by the processes in the linear programming solution. Next, we derive an upper bound on the number of clusters.

Lemma 4.2. *Let $V_1, V_2, \ldots, V_z$ be a clustering of graph $G$. Assume that every cluster $V_i$ has a center vertex $v_i$ and the distance between two distinct center vertices is at least $r$. Then the number of clusters $z$ is at most $\frac{1}{r}M + n$.*

Proof. Consider process $p$ and its trajectory $(p^0, p^1, \ldots, p^T)$. Let $\text{len}(p) = \sum_{t=1}^{T} d(p^{t-1}, p^t)$ be the length of the trajectory. Then, there are at most $\lceil \frac{1}{r} \text{len}(p) \rceil$ cluster centers among these trajectory vertices, as otherwise the distance between two center vertices would be strictly lower than $r$. Summing over all process trajectories we obtain:

$$z \leq \sum_{p \in P} \lceil \tfrac{1}{r} \text{len}(p) \rceil \leq \tfrac{1}{r} \sum_{p \in P} \text{len}(p) + n \leq \tfrac{1}{r} M + n,$$

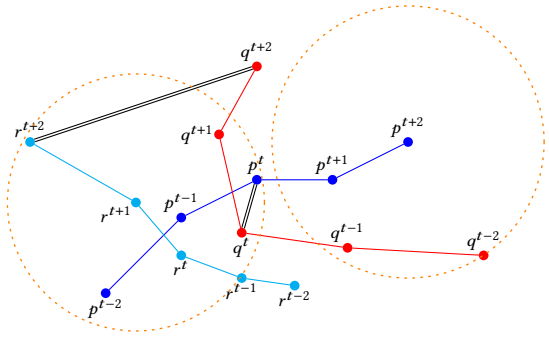


**Figure 3: Transformation of the LP solution into a graph: The behavior of every process (here $r$, $p$ and $q$) is represented by a trajectory of $T+1$ points, which are connected by *migration edges*. The communication requests (here between $q$ and $p$ at time $t$, and $r$ and $q$ at time $t+2$) are represented by *communication edges* (drawn as a double edge). The dashed circles are clusters with centers $r^{t+1}$ and $p^{t+2}$, respectively.**

which proves the lemma. □

Let $r = \frac{\epsilon}{4(1+\epsilon)}$ and $Z = \frac{1}{r}M + n$. We want to ensure that the radius of each cluster is between $r$ and $2r$. The graph clustering algorithm works as follows. During iteration $i$ we arbitrarily select a still unassigned vertex $v$ as the new cluster center. Then, we choose $x$ from interval $[0, r]$ according to the probability density $\rho(x) = \frac{Z}{Z-1}\frac{\ln Z}{r}e^{-\ln Z\frac{x}{r}}$ and set radius $r_i$ to $r+x$. Subsequently, we create a new cluster $V_i$ of unassigned vertices that are at distance at most $r_i$ from $v$ and remove it from the graph $G$. The procedure is repeated until there are no unassigned vertices left. The algorithm is described more formally in Procedure CreateClusters (see Figure 4).

The proof of Lemma 4.3 follows essentially the same steps as the proof in [7]. We slightly modify the analysis to account for the fact that the radius we choose in each iteration is now from the range $[r, 2r]$ and not $[0, r]$ as in [7]. This gives us an approximation factor that is dependent on the number of clusters and not on the number of vertices in the graph.

Lemma 4.3. *Algorithm CreateClusters applied to a graph $G$ produces a valid clustering $C$ with expected cost $E[cost(C)]$ at most $\frac{8}{\epsilon}(1+\epsilon)\ln(Z)cost(G)$.*

Proof. By construction, the diameter of each cluster is at most $4r$. According to Lemma 4.1 the maximum number of vertices corresponding to some time step $t$ inside each cluster is at most $(1+\epsilon)k$. Hence, the clustering is valid.

Next, we derive a bound on the probability that an edge $e = \{u, w\}$ with length $d(e)$ is cut. For ease of analysis let $r' = r/\ln(Z)$. Then, $x$ is chosen from the interval $[0, r'\ln(Z)]$ according to the probability density $\rho(x) = \frac{Z}{Z-1}\frac{1}{r'}e^{-\frac{x}{r'}}$. We fix some iteration $t$. Let $v$ be the vertex that was chosen as the cluster center in this iteration. Assuming that $u$ and $w$ are still unassigned, we derive a bound on the probability that they will be separated during the next iterations. W.l.o.g. let $d(v,u) \le d(v,w)$. We define the following events:

**Input:** Graph $G = (V, E, d)$, $\epsilon$, $M$
**Output:** Set of clusters $C$

**procedure** CreateClusters($G, \epsilon, M$)
  $r \leftarrow \frac{\epsilon}{4(1+\epsilon)}$
  $Z \leftarrow \frac{1}{r}M + n$
  $C \leftarrow \emptyset$
  $i \leftarrow 1$
  **while** $V \neq \emptyset$ **do**
    Choose arbitrary unassigned vertex $v$
    Choose $x$ from $[0, r]$ according to probability density
      $\rho(x) = \frac{Z}{Z-1}\frac{\ln Z}{r}e^{-\ln Z\frac{x}{r}}$
    $r_i \leftarrow r + x$
    $V_i \leftarrow B(v, r_i)$
    $V \leftarrow V - V_i$
    $C \leftarrow C \cup \{V_i\}$
    $i \leftarrow i + 1$
  **end while**
  return $C$
**end procedure**

**Figure 4: Algorithm for graph clustering**

- $U_t$: vertices $u$ and $w$ are still unassigned at the beginning of iteration $t$
- $C_t$: $d(v,u) \le r + x \le d(v,w)$ given $U_t$, i.e., cluster $V_t$ cuts $e$
- $N_t$: $r + x < d(v,u)$ given $U_t$, i.e., both of $u$ and $w$ remain unassigned
- $X_t$: edge $e$ will be cut by some cluster $V_j$, $j \ge t$, given $U_t$

Trivially, $P(U_0) = 1$. Eventually, we want to bound $P(X_0)$. Notice that the edge could be either cut in the current iteration or the edge "survives" and it will be cut in some of the following iterations. We state it as follows:

$$P(X_t) = P(C_t) + P(N_t)P(X_{t+1})$$

Let $a = \max\{d(v,u) - r, 0\}$ and $b = \max\{d(v,w) - r, 0\}$. Then, we obtain:

$$\begin{aligned} P(C_t) &= \int_a^b \rho(x)dx \\ &= \frac{Z}{Z-1}(e^{-\frac{a}{r'}} - e^{-\frac{b}{r'}}) \\ &= \frac{Z}{Z-1}(1 - e^{-\frac{b-a}{r'}})e^{-\frac{a}{r'}} \\ &\le \frac{Z}{Z-1}\frac{d(u,w)}{r'}e^{-\frac{a}{r'}}, \end{aligned}$$

where we used $(1 - e^{-x}) \le x$ and $b - a \le d(u,w)$ in the last step. For $P(N_t)$ we obtain

$$P(N_t) = \int_0^a \rho(x)dx = \frac{Z}{Z-1}(1 - e^{-\frac{a}{r'}}) \ .$$

We want to prove by induction that $P(X_t) \le (2 - \frac{t}{Z-1})\frac{d(u,w)}{r'})$. Due to Lemma 4.2 and by construction of algorithm CreateClusters we know that the number of clusters is at most $Z$. Consider the last iteration $t' \le Z$, i.e., the base case. Then, $P(X_{t'}) = 0$. Otherwise the

edge would be cut, there would be still unassigned vertices left and $t'$ would not be the last iteration, so the bound holds.

Let $P(X_{t+1}) \leq (2-\frac{t+1}{Z-1})\frac{d(u,w)}{r'}$. Notice, that if $d(v,u) > 2r$, then trivially $P(C_t) = 0$ and $P(N_t) = 1$, leading to $P(X_t) = P(X_{t+1}) \leq (2-\frac{t+1}{Z-1})\frac{d(u,w)}{r'} \leq (2-\frac{t}{Z-1})\frac{d(u,w)}{r'}$. Therefore, we can assume that $d(v,u) \leq 2r$. Then:

$$\begin{aligned}
P(X_t) &= P(C_t) + P(N_t)P(X_{t+1}) \\
&\leq \frac{Z}{Z-1}\frac{d(u,w)}{r'}e^{-\frac{a}{r'}} \\
&\quad + \frac{Z}{Z-1}(1-e^{-\frac{a}{r'}})(2-\tfrac{t+1}{Z-1})\frac{d(u,w)}{r'} \\
&= \frac{Z}{Z-1}\left[e^{-\frac{a}{r'}} + (1-e^{-\frac{a}{r'}})(2-\tfrac{t+1}{Z-1})\right]\frac{d(u,w)}{r'} \\
&= \frac{Z}{Z-1}\left[1 + (1-e^{-\frac{a}{r'}}) - (1-e^{-\frac{a}{r'}})\tfrac{t+1}{Z-1}\right]\frac{d(u,w)}{r'} \\
&= \frac{Z}{Z-1}\left[1 + (1-\tfrac{t+1}{Z-1})(1-e^{-\frac{a}{r'}})\right]\frac{d(u,w)}{r'} \\
&\leq \frac{Z}{Z-1}\left[1 + (1-\tfrac{t+1}{Z-1})(1-\tfrac{1}{Z})\right]\frac{d(u,w)}{r'} \\
&= \left[2 - \frac{t}{Z-1}\right]\frac{d(u,w)}{r'}
\end{aligned}$$

The last inequality holds since $d(v,u) - r \leq r = r'\ln(Z)$ and therefore $a \leq r'\ln(Z)$ which leads to $e^{-\frac{a}{r'}} \geq \frac{1}{Z}$. By plugging in $r' = r/\ln Z$ we obtain $P(X_0) \leq \frac{2}{r}\ln(Z)d(u,w)$.

Let $C$ be the clustering produced by algorithm CreateClusters and let $W_c$ and $W_m$ be the number of communication edges and migration edges, respectively, that are cut by it. The cost of clustering $C$ is $cost(C) = \alpha W_m + W_c$. Then, by linearity of expectation

$$\begin{aligned}
E[cost(C)] &= \alpha E[W_m] + E[W_c] \\
&= \alpha \textstyle\sum_{t\geq 1, p\in P} P[e = \{p^{t-1}, p^t\} \text{ is cut}] \\
&\quad + \textstyle\sum_{t\geq 1, \sigma_t = \{p,q\}} P[e = \{p^t, q^t\} \text{ is cut}] \\
&\leq \tfrac{2}{r}\ln(Z)\big(\alpha \textstyle\sum_{t\geq 1, p\in P} d(p^{t-1}, p^t) \\
&\qquad + \textstyle\sum_{t\geq 1, \sigma_t = \{p,q\}} d(p^t, q^t)\big) \\
&= \tfrac{2}{r}\ln(Z)cost(G) \\
&= \tfrac{8}{\epsilon}(1+\epsilon)\ln(Z)cost(G)
\end{aligned}$$

□

## 4.2 Splitting the Input Sequence

We managed to partition the graph into clusters, but our approximation factor still depends on the maximum number of clusters $Z = \frac{4}{\epsilon}(1+\epsilon)M + n$, where $M$ is the total length of migration edges. To eliminate this dependency we apply the following procedure: We divide our input sequence into phases that incur migration cost $\Theta(\alpha n)$. In particular, we divide the graph obtained from the LP into subgraphs $G_0 = (V_0, E_0, d), G_1 = (V_1, E_1, d), \ldots$ such that each subgraph $G_i$ has migration cost between $\alpha n$ and $2\alpha n$ and contains vertices with timestamps from interval $[t_i, t_{i+1} - 1]$. The time intervals are not overlapping. We denote by $\mathcal{V} = \{V_0, V_1, \ldots\}$ the clustering of graph $G$ into subgraphs $G_i$. We use algorithm CreateClusters to generate clusters $C_i$ for each subgraph $G_i$ independently and combine the results.

Let $C = \bigcup_i C_i$ be the combined clustering. Then, $cost(C)$ is exactly $\sum_{i\geq 0} cost(C_i) + cost(\mathcal{V})$. Clearly, $cost(\mathcal{V}) \leq |\mathcal{V}|\alpha n$, since by construction the number of migration edges that connect subgraph $G_i$ with subgraph $G_{i+1}$ is $n$ (only consecutive subgraphs are connected by edges). Let $M_i$ denote the total length of migration edges in subgraph $G_i$. By construction $\alpha M_i \leq 2\alpha n$. Hence, the number of clusters $Z_i = |C_i|$ in each subgraph $G_i$ is at most $\frac{(1+\epsilon)}{\epsilon}8n + n$. Since $\epsilon \geq \frac{1}{k} \geq \frac{1}{n}$ and the function $\frac{1+\epsilon}{\epsilon} \leq n+1$ for $\epsilon \geq \frac{1}{n}$ we obtain: $Z_i \leq 8n^2 + 9n \leq 16n^2$, for $n \geq 2$. Then, by linearity of expectation

$$\begin{aligned}
E[cost(C)] &\leq \textstyle\sum_{i\geq 0} E[cost_{G_i}(C_i)] + E[cost_G(\mathcal{V})] \\
&\leq \textstyle\sum_{i\geq 0} \frac{8}{\epsilon}(1+\epsilon)\log(Z_i)cost(G_i) + |\mathcal{V}|\alpha n \\
&\leq \tfrac{32}{\epsilon}\log(4n)\textstyle\sum_{i\geq 0} cost(G_i) + |\mathcal{V}|\alpha n \\
&\leq \tfrac{32}{\epsilon}\log(4n)\textstyle\sum_{i\geq 0}(cost(G_i) + \alpha n) \\
&\leq \tfrac{64}{\epsilon}\log(4n)\textstyle\sum_{i\geq 0} cost(G_i) \\
&\leq O(\tfrac{1}{\epsilon}\log n)cost(G)
\end{aligned}$$

The second inequation is due to Lemma 4.3. The third inequation holds because $Z_i \leq 16n^2$ and $\epsilon \leq 1$. The fifth inequation holds because $\alpha n \leq cost(G_i)$ for all $i$.

## 4.3 Scheduling Step

In the graph clustering step we partitioned the graph into clusters such that the cost of cut edges is $O(\frac{1}{\epsilon}\log n)cost(G)$ and capacity constraints are violated only by a factor of $(1+\epsilon)$. Now we want to assign clusters to servers. An assignment of a cluster $C$ to a server $s$ means that for all points $p^t \in C$ we place the process $p$ at time step $t$ on the server $s$. A migration edge between clusters corresponds to a process that may be migrating to another server (depending on the assignment of clusters to servers). A communication edge between clusters corresponds to a communication request that may be served remotely. If we assign the clusters directly to the servers, then the cost of our solution would correspond to the cost of cut edges, i.e., cost of migrations and communication requests served remotely. Still, an assignment of several clusters to one server could result in large capacity constraints violation.

Until now we established an upper bound on the number of vertices in each time step in the clusters. In order to schedule the processes without large capacity constraints violation we also need a lower bound. We split the clusters from the graph clustering step into subclusters, such that the cost increase is only a constant factor (dependent on $\epsilon$) and an assignment of subclusters to servers involves only small capacity constraint violation. We create subclusters by choosing consecutive non-overlapping time intervals and assign vertices with timestamps from the same time interval to a subcluster. For example, all vertices of cluster $C$ with timestamps in $[t_0, t_1]$ form a subcluster $S_0 \subseteq C$, the vertices with timestamps in $[t_1 + 1, t_2]$ form subcluster $S_1 \subseteq C$ and so on. In other words, we are "cutting" the cluster at time steps $t_0, t_1, t_2, \ldots$.

We say the *duration* $I(S)$ of a subcluster $S$ is the time interval $[t_b, t_e]$ such that $t_b$ is the smallest timestamp of all vertices in $S$ and $t_e$ the largest timestamp of all vertices in $S$. Let us refer to $t_b$ as the

**Input:** Set of clusters $C$
**Output:** Set of subclusters $\mathcal{S}$

**procedure** CreateSubclusters($C$)
  $\mathcal{S} = \emptyset$
  **for** $C \in C$ **do**
    $t \leftarrow 0$
    **while** $t \leq T$ **do**
      $t_b \leftarrow t$ ▷ start time of current subcluster
      $x \leftarrow |C \cap V(t)|$
      **while** $t \leq T$ and $|C \cap V(t)| \leq (1 + \frac{\epsilon}{4})x$ and $|C \cap V(t)| \geq (1 - \frac{\epsilon}{4})x$ **do**
        $t \leftarrow t + 1$
      **end while**
      $S \leftarrow \bigcup_{i=t_b}^{t-1} (C \cap V(i))$ ▷ the new subcluster
      $I(S) \leftarrow [t_b, t-1]$ ▷ duration with start time $t_b$, end time $t-1$
      $w(S) \leftarrow \min_{i \in I(S)} |C \cap V(i)|$ ▷ width of $S$
      $\mathcal{S} \leftarrow \mathcal{S} \cup \{S\}$
    **end while**
  **end for**
  return $\mathcal{S}$
**end procedure**

**Figure 5: Algorithm for subcluster creation**

*start time* and $t_e$ as the *end time* of subcluster $S$. We say a subcluster is *active* at time $t$ if $t \in I(S)$. For each subcluster we want that $|S \cap V(t)|$ remains approximately the same for all $t \in I(S)$. The idea is as follows. If for every subcluster $S$, $|S \cap V(t)|$ is between $w(S)$ and $(1+\epsilon)w(S)$ for all $t \in I(S)$, then we can assign the subclusters to the servers using a simple greedy algorithm. We call $w(S)$ the *width* of the subcluster $S$. Algorithm CreateSubclusters (see Figure 5) describes how the subclusters are generated (see also Figure 6 for a visualization).

Lemma 4.4. *The set $\mathcal{S}$ of subclusters returned by algorithm CreateSubclusters when applied to a clustering $C$ fulfills the following properties:*

- *every subcluster has approximately the same number of processes throughout its duration, i.e., $\forall t \in I(S)$: $|S \cap V(t)| \geq w(S)$ and $|S \cap V(t)| \leq (1+\epsilon)w(S)$*
- *$cost(\mathcal{S}) \leq O(\frac{1}{\epsilon})cost(C)$*

Proof. Let $I(S) = [t_b, t_e]$ and $x = |S \cap V(t_b)|$. By construction $w(S) = \min_{t \in I(S)} |S \cap V(t)|$. Since $w(S) \geq (1 - \frac{\epsilon}{4})x$ we obtain that $(1+\epsilon)w(S) \geq (1+\epsilon)(1-\frac{\epsilon}{4})x \geq (1+\frac{\epsilon}{2})x \geq (1+\frac{\epsilon}{4})x$ ($\epsilon^2 \leq \epsilon$ for $\epsilon \leq 1$) which is by construction an upper bound on $|S \cap V(t)|, \forall t \in I(S)$ .

Each subcluster induces a cut to its cluster. We are cutting the processes at each subclusters end time. At that time there are at most $(1 + \frac{\epsilon}{4})x$ processes, but since the beginning of the subclusters start time we already observed at least $\frac{\epsilon}{4}x$ migration edges that are cut, otherwise we would have continued the duration $I(S)$ of subluster $S$. Each time we observe at least $\alpha\frac{\epsilon}{4}x$ cost and increase the migration cost by at most $\alpha(1 + \frac{\epsilon}{4})x$. This corresponds to a factor of $O(\frac{1}{\epsilon})$ increase of the cost. □

As the next step, we schedule the subclusters using a greedy algorithm. We say the *load* of a server $s$ at time $t$ is the total width of active subclusters (at time $t$) that are assigned to $s$. We process the subclusters in the order of increasing start time and assign the current subcluster $S$ to the server that has minimum load at the start time of this cluster.

Lemma 4.5. *The greedy algorithm produces a scheduling such that the number of processes on each server is at most $2(1+\epsilon)k$ for all time steps.*

Proof. Let $S$ be the current subcluster we want to assign to a server. By an averaging argument the least loaded server $s$ must have load at most $k$ at this time (i.e., *before* scheduling $S$). Otherwise every server would have load greater that $k$ which results in the overall number of processes greater than $n$, a contradiction. By a monotonicity argument the current load of $s$ can also be at most $k$ for future time steps. This holds because we are processing the subclusters in the order of increasing start time.

Since $w(S) \leq k$, the load on server $s$ is at most $2k$ for all time steps (right *after* scheduling $S$). Furthermore, no subcluster will be assigned to $s$ as long as its load at the start time of the corresponding subcluster is greater that $k$. Hence, after the greedy algorithm finishes, the load on each server does not exceed $2k$ at any time step. According to Lemma 4.4 the actual number of processes on each server is at most $2(1+\epsilon)k$ for all time steps. □

Let $W$ be the cost of process migrations and remotely served requests incurred by scheduling subclusters $\mathcal{S}$. Clearly, $W \leq cost(\mathcal{S})$. Then

$$E[W] \leq O(\tfrac{1}{\epsilon^2} \log n)OPT$$

By setting $\epsilon' = \frac{\epsilon}{2}$ and applying the algorithm described in the previous sections with $\epsilon'$ as parameter, we can state the following theorem:

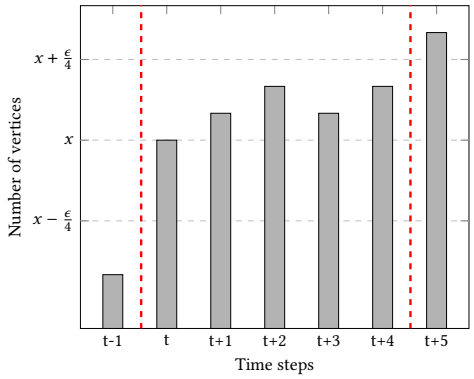


**Figure 6: Subdivision of a cluster $S$ into subclusters: Prior to time step $t$ we are "cutting" the cluster $S$ (left red line). Then, $x$ is the number of vertices in $S$ with timestamp $t$. In the next time step such that the number of vertices with the corresponding timestamp is greater than $x + \frac{\epsilon}{4}$ or lower than $x - \frac{\epsilon}{4}$ (which is $t+5$ in this example) we are cutting the cluster again (right red line). A subcluster with start time $t$ and end time $t + 4$ is created. We proceed in the same way with the remaining time steps.**

Theorem 4.6. *The dynamic balanced graph partitioning problem can be solved in polynomial time with approximation factor $O(\frac{1}{\epsilon^2}\log(n))$ and resource augmentation $2 + \epsilon$.*

## 5 CONCLUSION

Motivated by the vision of demand-aware networked systems, we revisited the dynamic balanced graph partitioning problem. Our main contribution is the first polynomial-time algorithm which achieves a polylogarithmic approximation ratio.

Our work leaves open several interesting avenues for future research. In particular, it remains to provide an exact characterization of the achievable approximation ratio under different allowed augmentation factors. It would also be interesting to see whether similar approximation ratios can be achieved using online algorithms as well, or whether there are lower bounds which show a separation.

## ACKNOWLEDGMENTS

Research supported by the German Research Foundation (DFG) project, Algorithms for Flexible Networks (FlexNets), 2021-2024, grant agreement 470029389.

## REFERENCES

[1] Konstantin Andreev and Harald Räcke. 2006. Balanced graph partitioning. *Theory of Computing Systems* 39, 6 (2006), 929–939.
[2] Chen Avin, Marcin Bienkowski, Andreas Loukas, Maciej Pacut, and Stefan Schmid. 2019. Dynamic Balanced Graph Partitioning. In *SIAM J. Discrete Math (SIDMA)*.
[3] Chen Avin, Louis Cohen, Mahmoud Parham, and Stefan Schmid. 2018. Competitive Clustering of Stochastic Communication Patterns on a Ring. In *Journal of Computing*.
[4] Chen Avin, Louis Cohen, and Stefan Schmid. 2017. Competitive Clustering of Stochastic Communication Patterns on the Ring. In *Proc. 5th International Conference on Networked Systems (NETYS)*.
[5] Chen Avin, Manya Ghobadi, Chen Griner, and Stefan Schmid. 2020. On the Complexity of Traffic Traces and Implications. In *Proc. ACM SIGMETRICS*.
[6] Chen Avin, Andreas Loukas, Maciej Pacut, and Stefan Schmid. 2016. Online Balanced Repartitioning. In *Proc. 30th International Symposium on Distributed Computing (DISC)*.
[7] Y. Bartal. 1996. Probabilistic approximation of metric spaces and its algorithmic applications. In *Proceedings of 37th Conference on Foundations of Computer Science*. 184–193. https://doi.org/10.1109/SFCS.1996.548477
[8] Yair Bartal. 1998. On Approximating Arbitrary Metrices by Tree Metrics. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing* (Dallas, Texas, USA) *(STOC '98)*. Association for Computing Machinery, New York, NY, USA, 161–168. https://doi.org/10.1145/276698.276725
[9] Theophilus Benson, Ashok Anand, Aditya Akella, and Ming Zhang. 2010. Understanding data center traffic characteristics. *ACM SIGCOMM Computer Communication Review* 40, 1 (2010), 92–99.
[10] Marcin Bienkowski, Martin Böhm, Martin Koutecký, Thomas Rothvoß, Jiří Sgall, and Pavel Veselý. 2021. Improved Analysis of Online Balanced Clustering. arXiv:2107.00145 [cs.DS]
[11] Guy Even, Joseph (Seffi) Naor, Satish Rao, and Baruch Schieber. 1999. Fast Approximate Graph Partitioning Algorithms. *SIAM J. Comput.* 28, 6 (1999), 2187–2214. https://doi.org/10.1137/S0097539796308217 arXiv:https://doi.org/10.1137/S0097539796308217
[12] Guy Even, Joseph Seffi Naor, Satish Rao, and Baruch Schieber. 2000. Divide-and-Conquer Approximation Algorithms via Spreading Metrics. *J. ACM* 47, 4 (July 2000), 585–616. https://doi.org/10.1145/347476.347478
[13] Björn Feldkord, Matthias Feldotto, Anupam Gupta, Guru Guruganesh, Amit Kumar, Sören Riechers, and David Wajc. 2018. Fully-Dynamic Bin Packing with Little Repacking. In *ICALP*. 51:1–51:24.
[14] Andreas Feldmann and Luca Foschini. 2012. Balanced Partitions of Trees and Applications. *Algorithmica* 71, 100–111. https://doi.org/10.1007/s00453-013-9802-3
[15] Andreas Emil Feldmann and Luca Foschini. 2015. Balanced partitions of trees and applications. *Algorithmica* 71, 2 (2015), 354–376.
[16] Amos Fiat, Richard M Karp, Michael Luby, Lyle A McGeoch, Daniel D Sleator, and Neal E Young. 1991. Competitive paging algorithms. *Journal of Algorithms* 12, 4 (1991), 685–699.
[17] Tobias Forner, Harald Räcke, and Stefan Schmid. 2021. Online Balanced Repartitioning of Dynamic Communication Patterns in Polynomial Time. In *Proc. SIAM Symposium on Algorithmic Principles of Computer Systems (APOCS)*.
[18] Monika Henzinger, Stefan Neumann, Harald Räcke, and Stefan Schmid. 2021. Tight Bounds for Online Graph Partitioning. In *Proc. ACM-SIAM Symposium on Discrete Algorithms (SODA)*.
[19] Monika Henzinger, Stefan Neumann, and Stefan Schmid. 2019. Efficient Distributed Workload (Re-)Embedding. In *Proc. ACM SIGMETRICS*.
[20] Dorit S Hochbaum and David B Shmoys. 1987. Using dual approximation algorithms for scheduling problems theoretical and practical results. *Journal of the ACM (JACM)* 34, 1 (1987), 144–162.
[21] Felix Hohne, Soren Schmitt, and Rob van Stee. 2022. SIGACT News Online Algorithms Column 38: 2021 in Review. *SIGACT News* 52, 4 (jan 2022), 80–96. https://doi.org/10.1145/3510382.3510396
[22] Srikanth Kandula, Sudipta Sengupta, Albert Greenberg, Parveen Patel, and Ronnie Chaiken. 2009. The nature of data center traffic: measurements & analysis. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement*. 202–208.
[23] Elias Koutsoupias and Christos H. Papadimitriou. 1995. On the k-Server Conjecture. *J. ACM* 42, 5 (Sept. 1995), 971–983. https://doi.org/10.1145/210118.210128
[24] Robert Krauthgamer, Joseph Naor, and Roy Schwartz. 2009. Partitioning graphs into balanced components. In *Proceedings of the twentieth annual ACM-SIAM symposium on Discrete algorithms*. SIAM, 942–949.
[25] Mark Manasse, Lyle McGeoch, and Daniel Sleator. 1988. Competitive Algorithms for On-Line Problems. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing* (Chicago, Illinois, USA) *(STOC '88)*. Association for Computing Machinery, New York, NY, USA, 322–333. https://doi.org/10.1145/62212.62243
[26] Mark S Manasse, Lyle A McGeoch, and Daniel D Sleator. 1990. Competitive algorithms for server problems. *Journal of Algorithms* 11, 2 (1990), 208–230. https://doi.org/10.1016/0196-6774(90)90003-W
[27] Maciej Pacut, Mahmoud Parham, and Stefan Schmid. 2021. Optimal Online Balanced Graph Partitioning. In *Proc. IEEE INFOCOM*.
[28] Peter Sanders, Naveen Sivadasan, and Martin Skutella. 2009. Online Scheduling with Bounded Migration. *Math. Oper. Res.* 34, 2 (2009), 481–498.

# A LIMITATION OF OTHER TECHNIQUES

In this section we discuss the problems arising from transferring techniques from [15, 24] to our more general problem.

In [15] the authors achieve violation $1 + \epsilon$ on trees by using dynamic programming. The idea is to characterize the partition of the tree into connected components by a *signature* vector, that describes the approximate sizes of components within the partition. Crucially, it is shown that there are only a small number of different signatures (polynomial in $n$) and that the best possible partition for a given signature can be computed in polynomial time via dynamic programming. This gives the aforementioned guaranty.

However, if you want to adapt this technique to our scenario the signature needs to contain the component sizes for each time step. This would result in a number of signatures that is exponential in the number of time steps.

Another open problem is whether the approximation guarantee could be improved from $O(\log n)$ to $O(\sqrt{\log n \log k})$ using techniques from [24]. The algorithm there consists of two steps. In a first step the $k$-balanced partitioning problem is formulated using a semidefinite program with an $\ell_2^2$ spreading metric, which is subsequently mapped into $\ell_2$. In a second step the components are created by repeatedly cutting out pieces from the graph using a random projection procedure. However, the size of such a piece could be too large. In this case the algorithm simply rejects the piece. With constant probability a piece is not too large, which bounds the expected number of times the cutting procedure is performed.

If you want to adapt this approach to our scenario, then you need to reject all pieces that contain too many vertices from one of the time steps. However, in this case we cannot argue that with constant probability the piece is not too large, as for an arbitrary number of time steps the probability bound will not hold. Therefore, the running time of the algorithm would be exponential in the number of time steps.