Improved Bounds for Online Balanced Graph Re-Partitioning

Rajmohan Rajaraman
Northeastern University, Boston, MA, USA

Omer Wasim
Northeastern University, Boston, MA, USA

Abstract

We study the online balanced graph re-partitioning problem (OBGR) which was introduced by Avin, Bienkowski, Loukas, Pacut, and Schmid [2] and has recently received significant attention [16, 12, 13, 10, 4] owing to potential applications in large-scale, data-intensive distributed computing. In OBGR, we have a set of ℓ clusters, each with k vertices (representing processes or virtual machines), and an online sequence of communication requests, each represented by a pair of vertices. Any request (u, v) incurs unit communication cost if u and v are located in different clusters (and zero otherwise). Any vertex can be migrated from one cluster to another at a migration cost of $\alpha \geq 1$. We consider the objective of minimizing the total communication and migration cost in the competitive analysis framework. The only known algorithms (which run in exponential time) include an $O(k^2\ell^2)$ competitive [2] and an $O(k\ell 2^{O(k)})$ competitive algorithm [4]. A lower bound of $\Omega(k\ell)$ is known [16]. In an effort to bridge the gap, recent results have considered *beyond worst case* analyses including resource augmentation (with augmented cluster capacity [2, 13, 12]) and restricted request sequences (the *learning* model [13, 12, 16]).

In this paper, we give *deterministic, polynomial-time* algorithms for OBGR, which *mildly* exploit resource augmentation (i.e. augmented cluster capacity of $(1 + \varepsilon)k$ for arbitrary $\varepsilon > 0$). We improve beyond $O(k^2\ell^2)$-competitiveness (for general ℓ, k) by first giving a simple algorithm with competitive ratio $O(k\ell^2 \log k)$. Our main result is an algorithm with a significantly improved competitive ratio of $O(k\ell \log k)$. At a high level, we achieve this by employing i) an ILP framework to guide the allocation of large components, ii) a simple "any fit" style assignment of small components and iii) a charging argument which allows us to bound the cost of migrations. Like previous work on OBGR, our algorithm and analysis are phase-based, where each phase solves an independent instance of the learning model. Finally, we give an $\Omega(\alpha k\ell \log k)$ lower bound on the total cost incurred by any algorithm for OBGR under the learning model, which quantifies the limitation of a phase-based approach.

2012 ACM Subject Classification Theory of computation → Online algorithms

Keywords and phrases online algorithms, graph partitioning, competitive analysis

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.83

1 Introduction

Modern data intensive applications which are distributed across data centers or clusters generate a large amount of network traffic [21, 18, 3]. To enable efficient communication among processes or virtual machines that may be dispersed in these clusters, many distributed systems are increasingly re-configurable and demand-aware [5]. Since inter-cluster communication can incur significant cost due to physical distance and limited bandwidth, clusters may strategically migrate processes to reduce the cost of communication, subject to cluster capacity constraints. The online balanced graph re-partitioning (OBGR) problem, introduced by Avin, Bienkowski, Loukas, Pacut, and Schmid [2], is an algorithmic investigation of trade-offs between migration and inter-cluster communication in an environment where the sequence of communication requests is unknown or hard to predict.

In OGBR, we are given ℓ clusters (representing servers or data centers), each holding at most k vertices (representing processes or virtual machines), and an online sequence of edges (representing communication requests). The algorithm maintains a partition of the vertices among the ℓ clusters so that each set of the partition contains at most k vertices. The *communication cost* of serving a request (u, v) is 0 if u and v are in the same cluster and 1, otherwise. Prior to serving any request, an algorithm has the option of migrating any vertex from one cluster to another for a *migration cost* of $\alpha \in \mathbb{Z}^+$. Given an online sequence σ of requests, the cost incurred by an (online) algorithm $\mathcal{A}$, denoted by $c(\mathcal{A}, \sigma)$ is the sum of the communication costs and migration costs over σ. Let $OPT(\sigma)$ denote the cost incurred by an optimal offline algorithm, which knows σ in advance. We measure the performance of the algorithm in terms of the (*strict*) competitive ratio which is the minimum value of $\rho > 0$ such that for any input sequence σ and a fixed constant $\tau > 0$ (independent of the length of σ) we have $c(\mathcal{A}, \sigma) \leq \rho \cdot OPT(\sigma) + \tau$. We usually refer to $OPT(\sigma)$ as OPT when σ is clear from context.

The static version of balanced graph partitioning and its variants are well studied [14, 20, 15, 1]. In this problem, given a weighted graph on a set V of n vertices and an integer ℓ, the goal is to partition V into vertex sets $V_1, ..., V_\ell$ such that the total weight of edges of the form (u, v) where $u \in V_i, v \in V_j, j \neq i$ is minimized. The problem is NP-hard and even hard to approximate within a finite factor. Note that for $k = 2$, this corresponds to maximum matching and for $\ell = 2$, this reduces to the minimum bisection problem which is already NP-hard [11]. Several approximation and bi-criteria approximation algorithms are known [9, 8, 6, 7] (for a discussion of results, see [2]). Since balanced graph partitioning is NP-hard in the static setting, exponential time competitive algorithms have been considered in the online setting [2, 16, 13]. Note that a balanced partition of the graph induced by the entire request sequence may not necessarily correspond to the optimal offline algorithm's strategy since this strategy overlooks the initial assignment of vertices in clusters (and thus, the migration cost required to mimic a balanced partition), the length of the sequence and its evolution over time. On the other hand, there is an approximation-preserving polynomial time reduction from the static version of OBGR to the offline version of OBGR that repeats the edges of the hard static instance sufficiently many times to derive a hard offline instance. Since the offline problem is unlikely to admit any known polynomial time optimal algorithms, beyond worst-case analysis has been employed to study competitiveness and running times of OBGR. We briefly discuss two such settings in which OBGR has been studied.

Resource Augmentation

In the resource augmented setting, an $(1 + \varepsilon)$-*augmented* online algorithm is granted $(1 + \varepsilon)k$ capacity on each cluster for some constant $\varepsilon > 0$, and its performance is compared with the optimal offline algorithm with capacity exactly k per cluster. This is similar in vein to the offline bi-criteria versions of the offline balanced graph partitioning problem [6, 7] where the algorithm is required to partition V into ℓ clusters that minimizes the weighted sum of cut edges, such that the number of excess vertices assigned to any cluster is at most δk for some $\delta > 0$. The cost of an algorithm's obtained partition is compared to the cost of an optimal partition of V in which clusters are assigned exactly k vertices. We note that resource augmentation has been studied extensively in online algorithms (e.g., see [17, 24, 25]), and goes back as far as the earliest work on caching [22].

Constrained Input

A special case of OBGR that has been recently considered is the so-called *learning model*, introduced by Henzinger, Neumann, Räcke, and Schmid [12] and studied later in [4, 16]. In this model, the online sequence satisfies the condition that there exists a feasible assignment of vertices to clusters without any inter-cluster requests in the sequence. Thus, upon executing such an assignment of vertices, any algorithm incurs zero cost. In other words, an online algorithm in this model is required to *learn* an optimal partitioning of V into k clusters with no inter-cluster edges. In contrast to the *general model* (i.e. with an arbitrary request sequence), the learning model focuses only on migration costs.

1.1 Related work

OBGR without resource augmentation

In [2], an $O(k^2\ell^2)$ upper bound and an $\Omega(k)$ lower bound are established on the competitive ratio of any deterministic algorithm for OBGR without resource augmentation. The lower bound has been improved to $\Omega(k\ell)$ in recent work by Pacut, Parham, and Schmid [16]. The special cases of $k = 2$ (online re-matching problem) and $k = 3$ have also been studied [2, 16]. In very recent work, Bienkowski, Böhm, Koutecký, Rothvoß, Sgall, and Veselý [4] give an $O(k\ell 2^{O(k)})$-competitive algorithm for OBGR, which is optimal for constant k.

OBGR with resource augmentation

The $\Omega(k)$ lower bound of [2] holds even when the algorithm is allowed an arbitrary amount of resource augmentation as long as $\ell \geq 2$ and all vertices do not fit into a single cluster. The main result of [2] is an $O(k \log k)$-competitive deterministic algorithm for OBGR with $(2+\varepsilon)k$ augmented cluster capacity for $\varepsilon \in (0, 1)$. Very recently, Forner, Räcke, and Schmid [10] give a *polynomial time* deterministic $O(k \log k)$-competitive algorithm in the same setting.

The learning model

In [16], the authors present a tight $\Theta(k\ell)$ bound for the best deterministic competitive ratio in the learning model without resource augmentation. Moreover, they show that a lower bound of $\Omega(\ell)$ holds even in the $(1 + \varepsilon)$-augmented setting for $\varepsilon < 1/3$. Henzinger, Neumann and Schmid [13] introduced the learning model of OBGR and give a $O((\ell \log \ell \log n)/\varepsilon)$-competitive algorithm and a lower bound of $\Omega(1/\varepsilon + \log n)$ assuming $(1 + \varepsilon)k$ augmented capacity for $\varepsilon \in (0, 1/2)$. In more recent work, [12] establishes tight bounds of $\Theta(\log \ell + \log k)$ and $\Theta(\ell \log k)$ on the best competitive ratio of randomized and deterministic algorithms, respectively, for the learning model with resource augmentation.

Summarizing, for deterministic competitive ratios, the best known upper bound for OBGR is $O(k^2\ell^2)$ without resource augmentation and $O(k \log k)$ with $(2 + \varepsilon)$-augmentation, while the best known lower bound is $\Omega(k\ell)$ without resource augmentation and $\Omega(k + \ell \log k)$ with $(1 + \varepsilon)$-augmentation for $\varepsilon < 1/3$.

1.2 Our results

In this paper, we give online deterministic $(1 + \varepsilon)$-augmented algorithms for OBGR in the general model, for an arbitrary constant $\varepsilon > 0$. We first observe that a ρ-competitive algorithm for OBGR in the learning model can be used to get a ρkl-competitive algorithm in the general model. The proof is deferred to Appendix A.

▶ **Observation 1.** *Any ρ-competitive algorithm for OBGR in the learning model can be transformed to a $O(\rho k\ell)$-competitive algorithm for OBGR in the general model.*

Using the $(1+\varepsilon)$-augmented deterministic $O(\ell \log k)$-competitive algorithm of [12] for the learning model, Observation 1 immediately yields $(1+\varepsilon)$-augmented deterministic $O(k\ell^2 \log k)$-competitive and randomized $O(k\ell(\log k + \log \ell))$-competitive algorithms for the general model. The algorithm of [12] for the learning model is quite sophisticated and relies on an intricate analysis. In Section 3, we give an alternative simpler algorithm for the general model referred to as $\mathcal{A}_S$, which admits a direct analysis and attains the same competitive ratio.

▶ **Theorem 2.** *There exists a deterministic, polynomial time, $(1+\varepsilon)$-augmented $O(k\ell^2 \log k)$-competitive algorithm for OBGR in the general model, for arbitrary constant $\varepsilon > 0$.*

Our main result, given in Section 4, is a polynomial time *deterministic* $(1+\varepsilon)$-augmented $O(k\ell \log k)$-competitive algorithm $\mathcal{A}_G$, for constant $\varepsilon > 0$; the competitive ratio nearly matches the lower bound of $\Omega(k\ell)$ without resource augmentation [16]. Under resource augmentation, our algorithm is optimal for constant k while for constant ℓ it is within a $O(\log k)$ factor of the optimal (following from the lower bound of $\Omega(k+\ell \log k)$ in the resource augmented setting). For many applications in which k is usually large (such as distributed communication between nodes placed in cloud servers), our algorithms have near-linear instead of an exponential [4] or quadratic [2] dependence on k in previous work.

▶ **Theorem 3.** *There exists a deterministic, polynomial time $(1+\varepsilon)$-augmented $O(k\ell \log k)$-competitive algorithm for OBGR in the general mode, for arbitrary constant $\varepsilon > 0$.*

The algorithm of Theorem 3 is a “phase-based” algorithm in which each phase solves OBGR in the learning model. The key component of our proof is an upper bound of $O(\alpha k\ell \log k)$ on the total cost of the algorithm in the learning model, starting from an arbitrary initial assignment of vertices. It is natural to ask whether this bound can be improved since any improvement would also yield an improved competitive ratio for OBGR in the general model. The following lower bound, which can be derived from a lower bound instance of [12], rules this out, thus presenting a limitation of a phase-based analysis approach.

▶ **Theorem 4.** *For any online deterministic (resp., randomized) algorithm with $(1+\varepsilon)$-augmentation for the learning model where $\varepsilon > 0$ is an arbitrary constant, there exists a sequence of requests for which the cost (resp., expected cost) is $\Omega(\alpha k\ell \log k)$.*

1.3 Overview of techniques

We highlight the main techniques we use to get a significantly improved competitive ratio for OBGR in the general model. Our algorithms partition the online sequence of requests into contiguous phases, and keep track of the graph induced by the communication requests within a phase. In particular, the algorithms ensure that during any phase all vertices in a connected component of the graph associated with the phase are assigned to the same cluster. On any request (u, v) where u is in component P_1 and v in P_2, P_1 and P_2 are merged into P_m and subsequently co-located. Components are classified as small or large based on a threshold size Dk where $D = \Theta(\varepsilon^2)$.

For the algorithm $\mathcal{A}_S$, if P_m is large, we solve an ILP to guide the assignment of large components. Small components may also need to be reassigned. If P_m is small, P_1 is migrated to P_2’s cluster as long as there is enough space. If that is not possible, small components are reassigned. We ensure that the maximum assigned volume on any cluster is $(1+\frac{\varepsilon}{4})k$ after the ILP is solved or small components are reassigned. By definition, large component merges happen only $O(1)$ times while at least $\frac{\varepsilon k}{4}$ total volume of small components is successfully migrated between any two small component reassignments. Using a charging argument, we show that every vertex can be charged at most $O(\ell \log k)$ before an optimal offline algorithm incurs a cost of 1 during that phase, yielding Theorem 2.

The approach for algorithm $\mathcal{A}_G$ is as follows. Each small component assigned to a cluster is allocated a volume which is within a $(1+\frac{\varepsilon}{4})$ factor of the component size. Once a large component is created during a phase, successive assignments of large components created by any merge are handled by ILP used in $\mathcal{A}_S$. We note that our ILP is similar to that of [12] and we follow their approach to invoke a result on sensitivity analysis of ILPs [19], which limits the change in assignments when a large component is created. This is not sufficient to establish Theorem 3, however, since small components can be completely displaced leading to high migration cost after every merge. Interestingly, we show that a simple "any fit" strategy for small components coupled with a charging argument is sufficient to bound the total migration cost by $O(k\ell \log k)$.

Finally, to establish the lower bound of Theorem 4, we show that for any competitive algorithm $\mathcal{A}$ there exists a request sequence composed of $\Omega(\log k)$ batches of requests and an initial assignment which is $\Omega(k\ell)$ far apart from $\mathcal{A}$'s assignment such that $\mathcal{A}$ incurs cost at least $\Omega(\alpha k\ell)$ on every batch.

2 Preliminaries

In this section, we present some definitions and high-level structure of our algorithms, which will be useful throughout the paper. Let $[n]$ denote the set of integers $\{1, 2, .., n\}$. Let V denote the set of $n = k\ell$ vertices. Let $\mathcal{C}$ denote the set of ℓ clusters. Each cluster $C \in \mathcal{C}$ is initially assigned exactly k vertices. A request is an unordered pair of vertices (u, v). A connected component P_i induced by a sequence of requests is the maximal set of vertices such that for any $u \in P_i$ there exists $v \in P_i$ s.t. (u, v) was a request in the sequence. The volume of any component P_i is its size $|P_i|$. Our algorithms maintain a set of connected components $\mathcal{P} = \{P_1, P_2, ..P_{|\mathcal{P}|}\}$ where $P_i \subseteq V$ for all i and $\bigcup_{i=1}^{|\mathcal{P}|} P_i = V$. Initially, $\mathcal{P} = \{\{u\} | u \in V\}$ i.e. the set of singleton vertices. We refer to a request (u_t, v_t) with $u_t \in P_1$ and $v_t \in P_2$ as an *inter-cluster request* (between P_1 and P_2) if P_1 and P_2 are assigned to different clusters at the start of time t.

Large and small components

Both our algorithms organize components into classes based on their volumes. A component P is in class i if $|P| \in [(1+\frac{\varepsilon}{4})^{i-1}, (1+\frac{\varepsilon}{4})^{i})$. A component is small if it belongs to a class i where $i \leq c_s = \lfloor \frac{4}{\varepsilon} \ln(\frac{\varepsilon^2 k}{32}) - 2 \rfloor$ where c_s denotes the number of small component classes. Hence, a component is small if it has volume at most Dk where $D < \frac{\varepsilon^2}{32} < \frac{\varepsilon}{4}$ and large otherwise. Note that the number of large component classes, denoted by c_l satisfies $c_l \leq \frac{4+\varepsilon}{\varepsilon} \ln(\frac{1}{D}) + 2 = O(1)$. *A large component P is understood to be in (large) component class i if it is in class $i + c_s$.* We assume $\varepsilon \geq \frac{4}{k}$. For any cluster C, we use $V(C)$, $V_S(C)$, and $V_L(C)$ to denote the total volume of all, small, and large components, assigned to C, respectively.

Phase-based algorithms

Both our algorithms are phase-based: they divide the sequence of requests into phases, and treat each phase as an independent sequence of requests.

▶ **Definition 5** (Phase). *A phase p of a sequence σ of requests is a maximal contiguous subsequence of σ such that there exists a feasible assignment of the set of large components induced by p to clusters in $\mathcal{C}$ satisfying the constraint that the total volume of large components assigned to any cluster is at most $(1+\frac{\varepsilon}{4})k$.*

A request sequence can be naturally partitioned into consecutive phases. Our algorithms begin a phase by setting $\mathcal{P}$ to the set of singletons and an assignment of vertices to clusters such that every cluster $C \in \mathcal{C}$ is assigned exactly k vertices. For all phases p and all $P_i \in \mathcal{P}$ where $\mathcal{P}$ is the set of components induced by p, vertices in P_i are assigned to the same cluster. Note that OPT increases by 1 per phase. For the sake of exposition, we give our algorithms for the case when $\alpha = 1$. In Appendix B, we show that a simple refinement of our algorithms handles the case when $\alpha > 1$, without asymptotically affecting the competitive ratios.

Merge cases

After any request, (u, v) between components P_1 and P_2 (where w.l.o.g., $|P_1| \leq |P_2|$) which are merged to form P_m, our algorithms consider two *merge cases*: *small*, when P_1, P_2 and P_m are small, and *large* when P_m is large. A merge is viewed as a *deletion* of components P_1, P_2 and an *insertion* of P_m.

3 An $O(k\ell^2 \log k)$-competitive algorithm

In this section, we present $\mathcal{A}_S$, an $O(kl^2 \log k)$-competitive algorithm. The algorithmic and analytic techniques developed play a key role in the improved algorithm $\mathcal{A}_G$ of Section 4.

We describe how $\mathcal{A}_S$ executes during any phase. Recall that for any inter-cluster request, our algorithm considers two merge cases. For both the cases, $\mathcal{A}_S$ calls subroutine `Balance-Small` to migrate and re-assign small components. For the large merge case, $\mathcal{A}_S$ calls subroutine `Reassign-Large` to solve an integer linear program (ILP) and guide the placement of large components. The ILP has a constant number of variables and constraints and hence can be solved in constant time. To present the ILP, we first introduce the notion of a signature, which encodes the number of large components of each class assigned to a cluster.

▶ **Definition 6** (Signature). *A signature $\tau = (\tau_1, \tau_2, ..., \tau_{c_l})$ for a cluster $C \in \mathcal{C}$ is a non-negative vector of dimension c_l where τ_i is the number of large components of class i that can be assigned to C such that $Dk \sum_{i=1}^{c_l} (1+\frac{\varepsilon}{4})^{i-1} \tau_i \leq k$.*

▶ **Lemma 7** (Upper bound on number of signatures). *The number of possible signatures for any cluster C is $O((\frac{1}{\varepsilon^2})^{c_l})$.*

Proof. Let τ be a possible signature. Note that $\tau_i \leq \frac{k}{Dk} = O(\frac{1}{\varepsilon^2})$ for all $i \in [c_l]$. Therefore, the total number of different signatures is $O((\frac{1}{\varepsilon^2})^{c_l})$. ◀

3.1 The ILP

We describe the ILP which is agnostic to the assignment of small components. Let $T = \{T_1, T_2, \ldots, \}$ denote the set of all possible signatures where w.l.o.g., T_1 is the all-zeroes vector. Let T_{ij} denote the j^{th} entry of signature T_i. From Lemma 7, $|T| = O(1)$. For each signature, T_i let variable $x_i \in [0, \ell]$ denote the number of clusters assigned a signature T_i. Furthermore, let $\kappa_j \in [0, \lceil \frac{\ell}{D} \rceil]$ denote the total number of class j large components. The ILP is as follows.

$$\sum_{i=1}^{|T|} T_{ij} x_i = \kappa_j \text{ for all } j \qquad \sum_{i=1}^{|T|} x_i = \ell \qquad x_i \in [0, \ell] \text{ for all } i \tag{1}$$

In matrix form, the ILP has $n_r = O(\ln(1/\varepsilon^2)) = O(1)$ rows and $n_c = O(|T|) = O(1)$ columns. Thus, the ILP can be solved in polynomial time. The following lemma shows that the total volume of large components assigned to any cluster never exceeds cluster capacities by more than an $\frac{\varepsilon}{4}$ factor.

▶ **Lemma 8** (Total volume of large components). *Let τ denote the assigned signature to cluster C according to which large components are assigned to C. Then, $V_L(C) < (1 + \frac{\varepsilon}{4})k$.*

Proof. We note that $V_L(C) < (1 + \frac{\varepsilon}{4}) Dk \sum_{i=1}^{c_l} (1 + \frac{\varepsilon}{4})^{i-1} \tau_i \le (1 + \frac{\varepsilon}{4})k$. ◀

Next, we give subroutines `Balance-Small` and `Reassign-Large`.

Algorithm `Balance-Small`.

```
1: for each cluster $C \in \mathcal{C}$ s.t. $V(C) > (1 + \frac{\varepsilon}{4})k$:
2:     while $V(C) > (1 + \frac{\varepsilon}{4})k$:
3:         Migrate a small component $P$ from $C$ to $C_1$ where $C_1 \leftarrow \arg\min_{C_2 \in \mathcal{C}} V(C_2)$.
```

Algorithm `Reassign-Large`.

```
1: Solve ILP (1) to obtain solution $x$.
2: if ILP is infeasible: return NULL.
3: Unmark all clusters $C \in \mathcal{C}$ and all large components in $\mathcal{P}$.
4: for $i \in [|T|]$:
5:     for $r \in [x_i]$:
6:         Assign signature $T_i$ to an unmarked cluster $C$, and mark $C$.
7:         for $j \in [c_l]$:
8:             Assign an unmarked large component $P$ of class $j$ to $C$ and mark $P$.
9:             Migrate $P$, if necessary.
```

If large components are assigned according to the subroutine `Reassign-Large`, then $V_L(C) \le (1 + \frac{\varepsilon}{4})k$ for all $C \in \mathcal{C}$ which follows from Lemma 8. On the other hand, if $V_L(c) \le (1 + \frac{\varepsilon}{4})k$ for all $C \in \mathcal{C}$ and `Balance-Small` is run, $V(C) \le (1 + \frac{\varepsilon}{4})k$ thereafter. The latter follows since $D < \frac{\varepsilon}{4}$ and there always exists a cluster C_1 such that $V(C_1) \le k$.

3.2 The algorithm

For a request (u_t, v_t) where $u_t \in P_1, v_t \in P_2$, the algorithm $\mathcal{A}_S$ proceeds as follows.

Proof Theorem 2. We bound the total migration cost incurred by the algorithm $\mathcal{A}_S$ during a phase. For the large merge case, the migration cost is bounded by $k\ell$. To pay for this cost, we charge each vertex in P_m a cost at most $\frac{\ell}{D}$. Every vertex can be charged $O(c_\ell)$ times in this manner within any phase, since a component size is bounded by k. For all $k\ell$ vertices, this gives a total charge of $O(\frac{k\ell^2}{D}) = O(k\ell^2)$.

Algorithm $\mathcal{A}_S$.

```
Input: Distinct components $P_1$ and $P_2$ in clusters $C_1$ and $C_2$, respectively; $|P_1| \leq |P_2|$
1: Merge $P_1$ and $P_2$ into $P_m$ and update $\mathcal{P}, \mathcal{P}_S$ and $\mathcal{P}_L$ accordingly.
2: if $C_1 \neq C_2$ :
3:     if $P_m$ is small:                                                  ▷ Small merge case
4:         Assign $P_m$ to $C_2$.
5:         if $V(C_2) \leq (1 + \frac{\varepsilon}{2})k$: Migrate all vertices of $P_1$ from $C_1$ to $C_2$.
6:         else: Run Balance-Small.
7:     else:                                                               ▷ Large merge Case
8:         Run Reassign-Large.
9:         if Reassign-Large returns NULL: Start a new phase.
10:        else: Run Balance-Small.
```

For the small merge case, there are two cases. If $V(C_2) \leq (1 + \frac{\varepsilon}{2})k$, then each vertex in P_1 is charged unit cost. Any vertex can be charged at most $O(\log k)$ in this way since $|P_m| \geq 2|P_1|$ yielding a total charge of $O(k\ell \log k)$. If $V(C_2) > (1 + \frac{\varepsilon}{2})k$ the migration cost incurred due to `Balance-Small` is at most $k\ell$. Let X denote the set of vertices that migrated to C_2 since the last invocation of `Balance-Small`. Then, $|X| > \frac{\varepsilon k}{4}$. Each vertex in X is charged $\frac{4\ell}{\varepsilon}$. Note that any vertex can be a vertex can be included in such a set X only $O(\log k)$ times before it is part of a large component. For all $k\ell$ vertices, this charge sums to $O(\frac{k\ell^2 \log k}{\varepsilon})$. Thus, the total amount charged to all vertices during a phase is $O(k\ell^2 \log k)$, completing the proof of the theorem. ◀

4 An $O(k\ell \log k)$-competitive algorithm

In this section, we present algorithm $\mathcal{A}_G$. A major shortcoming of $\mathcal{A}_S$ is that a cost of $\Omega(k\ell^2)$ can be incurred for both small and large merge cases. For a large merge case, $\mathcal{A}_G$ addresses this by ensuring that the total volume $O(k)$ large components migrated is $O(k)$ by employing a sensitivity analysis. The $O(k\ell \log k)$-competitiveness of $\mathcal{A}_G$ crucially hinges on bounding the migration cost of small components after a large merge case by $O(k)$. To this end, we give a simple "any-fit" assignment procedure for small components. Effectively, the algorithm guarantees that the the total migration cost for both merge cases is $O(|P_m|)$, which can be charged to P_m. This yields the desired competitive ratio.

The pseudo code of Algorithm $\mathcal{A}_G$ is given below. The algorithm executes as follows. At any given time, the algorithm maintains the property that the volume assigned to every class i component is given by $(1 + \frac{\varepsilon}{4})^i$. Thus, the total assigned volume for a cluster C overestimates the total volume of components assigned to C by a $(1 + \frac{\varepsilon}{4})$ factor. For the large merge case, an ILP is solved to handle assignment of large components similarly to $\mathcal{A}_S$. The assignment of large components is completely independent of small components. Thus, the reassignment of large components can displace small components. A displacement of small component P is viewed as a deletion and successive (re)insertion of P. In the next section, we give a procedure to handle the large merge case and show that the total volume of large components migrated is $O(k)$ if P_m is large.

4.1 Handling large components

To handle the large merge case, we use ILP (1). Additionally, we employ a well known bound on the sensitivity of optimal ILP solutions.

Algorithm $\mathcal{A}_G$.

```
Input: Components $P_1$ and $P_2$ in clusters $C_1$, $C_2$ of class $i, j$ respectively; $i \leq j$.
 1: Merge $P_1$ and $P_2$ into $P_m$ and update $\mathcal{P}, \mathcal{P}_S$ and $\mathcal{P}_L$ respectively.
 2: if $P_m$ is large                                  ▷ Large merge case (see Section 4.1)
 3:     Solve ILP(1).
 4:     Run algorithm Assign Signatures and let $\mathcal{C}' \subseteq \mathcal{C}$ be the set of clusters whose signatures changed.
 5:     for all $C \in \mathcal{C}'$
 6:         for all $P \in \mathcal{P}_S$ assigned to $C$
 7:             if $U(C) < \lceil |P| \rceil_{(1+\frac{\varepsilon}{4})}$
 8:                 Assign and migrate $P$ to $C_3 \in \mathcal{C}$ where $U(C_3) \geq \lceil |P| \rceil_{(1+\frac{\varepsilon}{4})}$.
 9:                 $U(C_3) \leftarrow U(C_3) - \lceil |P| \rceil_{(1+\frac{\varepsilon}{4})}$.
10:             else
11:                 $U(C) \leftarrow U(C) - \lceil |P| \rceil_{(1+\frac{\varepsilon}{4})}$.   ▷ The assignment of $P$ remains unchanged
12: else                                                ▷ Small merge case (see Section 4.2)
13:     if $(1+\frac{\varepsilon}{4})^j \geq |P_1| + |P_2|$
14:         Migrate vertices of $P_1$ to $C_2$.
15:     else
16:         if $U(C_2) \geq (1+\frac{\varepsilon}{4})^m - (1+\frac{\varepsilon}{4})^j$
17:             Migrate vertices of $P_1$ to $C_2$.
18:             $U(C_2) \leftarrow U(C_2) - (1+\frac{\varepsilon}{4})^m + (1+\frac{\varepsilon}{4})^j$.
19:         else
20:             Migrate vertices of $P_m$ to $C_3$ where $U(C_3) \geq (1+\frac{\varepsilon}{4})^m$.
21:             $U(C_3) \leftarrow U(C_3) - (1+\frac{\varepsilon}{4})^m$.
```

▶ **Theorem 9** (reproduced verbatim from [19]). *Let A be an integral $n_r \times n_c$ matrix, such that each subdeterminant of A is at most Δ in absolute value; let b' and b" be column n_r-vectors, and let c be a row n_c-vector. Suppose $\max\{cx|Ax \leq b' : x \text{ integral}\}$ and $\max\{cx|Ax \leq b" : x \text{ integral}\}$ are finite. Then for each optimum solution z' of the first maximum there exists an optimum solution z" of the second maximum such that $\|z' - z"\|_\infty \leq n_c\Delta(\|b' - b"\|_\infty + 2)$.*

Following the merge, the RHS vector in our ILP changes by at most 1 in the infinity norm. To bound the sub-determinant, we use the Hadamard inequality to derive that $\Delta \leq n_c^{n_c/2} A_{max}^{n_c/2}$, where A_{max} denotes the maximum entry (in absolute value) of the constraint matrix A. Each entry in the constraint matrix of our ILP has value either 1 or T_{ij} so that $A_{max} \leq \frac{k}{Dk} = O(1/\varepsilon^2)$. As a result, $\Delta = O((|T|/\varepsilon^2)^{|T|})$. Thus, the optimal solution to the ILP changes by $O(|T|\Delta)$ in the infinity norm. Since x has dimension $|T|$ the number of signatures which change between any two optimal solutions is $O(|T|^2\Delta)$.

Assigning signatures to clusters

Let $x = (x_1, ..., x_{|T|})$ denote the optimal solution obtained after solving the ILP. The procedure `Assign Signatures` greedily assigns signatures to clusters. Following greedy assignment of signatures, large components are migrated between clusters whose assigned signatures changed to reflect new component assignments. The pseudo code is given as follows.

▶ **Lemma 10.** *The number of clusters whose assigned signatures change whenever a large component is created is $O(|T|^2\Delta) = O(1)$.*

Algorithm `Assign Signatures`.

```
1:  Unmark all clusters $C \in \mathcal{C}$.
2:  $\mathcal{C}' \leftarrow \emptyset$
3:  for $i = 1$ to $|T|$:
4:      $z_i = x_i$.
5:      while $z_i \neq 0$:
6:          if there is an unmarked cluster $C$ which has assigned signature $T_i$
7:              Mark $C$.
8:          else
9:              Pick an arbitrary unmarked cluster $C$, assign it signature $T_i$.
10:             Mark $C$ and set $\mathcal{C}' \leftarrow \mathcal{C}' \cup \{C\}$.
11:         $z_i \leftarrow z_i - 1$.
12: $\mathcal{P}_{\mathcal{C}'} \leftarrow \{P|\ P \in \mathcal{P}_L$ and $P$ is assigned to some $C \in \mathcal{C}'\}$.
13: for $C \in \mathcal{C}'$            ▷ Migrate large components to reflect the change in signature.
14:     $\tau \leftarrow$ assigned signature of $C$.
15:     for $i \in [c_l]$
16:         for $j \in [\tau_i]$
17:             $P \leftarrow$ class $i$ component in $\mathcal{P}_{\mathcal{C}'}$.
18:             Assign $P$ to $C$ and migrate if necessary.
19:             $\mathcal{P}_{\mathcal{C}'} \leftarrow \mathcal{P}_{\mathcal{C}'} \backslash \{P\}$.
        $U(C) \leftarrow (1+\varepsilon)k - A_L(C)$.
```

Proof. The greedy procedure ensures that at most $O(|T|\Delta)$ clusters previously assigned a signature T_i for $i \in [|T|]$ are subsequently assigned a new signature. Thus, at most $O(|T|^2\Delta) = O(1)$ clusters change their assigned signatures. ◀

4.2 Handling small components

In this section, we give a simple procedure to assign small components. This procedure is used for both small and large merge cases. In the latter case, small components may need to be re-assigned due to displacements following a re-assignment of large components. Each small component P of class i is allocated volume exactly $(1+\frac{\epsilon}{4})^i$ on a cluster to which it is assigned, i.e. the allocated volume of a component is equal to $\lceil |P| \rceil_{(1+\frac{\epsilon}{4})}$ where $\lceil x \rceil_{(1+\frac{\epsilon}{4})}$ denotes the value x *rounded up* to the nearest multiple of $(1+\frac{\epsilon}{4})$. We introduce some notation. Let $A_L(C)$ and $A_S(C)$ denote volume allocated to large and small components respectively on a cluster $C \in \mathcal{C}$. Let $\mathcal{P}_S(C), \mathcal{P}_L(C) \subseteq \mathcal{P}_S$ denote the set of small and large components respectively assigned to a cluster C. Note that $A_L(C) = Dk\sum_{i=1}^{c_l} \tau_i(1+\frac{\epsilon}{4})^i$ where τ is the signature assigned to C. If $\mathcal{P}$ does not have any large components, then $A_L(C) = 0$ for all $C \in \mathcal{C}$. Moreover, $A_S(C) = \sum_{P \in \mathcal{P}_S(C)} \lceil |P| \rceil_{1+\frac{\epsilon}{4}}$. We define the unallocated volume $U(C)$ of cluster $C \in \mathcal{C}$ as $U(C) = (1+\epsilon)k - A_L(C) - A_S(C)$.

A small component P of class i which is currently unassigned, is assigned to an arbitrary cluster C whose unallocated volume $U(C)$ is greater than $(1+\frac{\epsilon}{4})^i$. Note that such a cluster C must always exist since otherwise this implies that the total volume of components exceeds kl, a contradiction. Below, we outline the assignment of small components.

Small merge case

Consider the small merge case in which components P_1 and P_2 of class i (resp. j) currently assigned to C_1 (resp. C_2) are merged into P_m of class m. W.l.o.g., let $i \leq j$. If $(1+\frac{\epsilon}{4})^j \geq |P_1| + |P_2|$, vertices of P_1 are migrated to C_2. In this case, $m = j$. On the other hand, if $m \neq j$ there are two cases to consider. If $U(C_2) \geq (1+\frac{\epsilon}{4})^m - (1+\frac{\epsilon}{4})^j$ then vertices of P_1 are migrated to C_2. Else, vertices in $P_1 \cup P_2$ are migrated to cluster C_3 where $U(C_3) \geq (1+\frac{\epsilon}{4})^m$. In all cases, P_m is allocated a volume of $(1+\frac{\epsilon}{4})^m$.

Handling displacements

Consider the large merge case in which re-assignment of large components may displace small components. Each small component P assigned to a cluster C whose signature changes after a large merge is assigned to a cluster C' where $U(C') \geq (1+\frac{\epsilon}{4})^i$. Since only $O(1)$ clusters change signatures, the total volume of small components displaced is bounded by $O(k)$.

Proof of Theorem 3. The migration cost of large and small merge cases is analyzed separately. For the large merge case, it follows by Lemma 10 that the total volume of large components migrated is $O(k)$, since the assigned signatures change for only $O(1)$ clusters. Let $\mathcal{C}' \subseteq \mathcal{C}$ denote the set of clusters whose signatures changed. The total volume of small components assigned to $\mathcal{C}'$ is bounded by $O(k)$. As a result, the total migration cost to reassign both small and large components is $O(k)$ which is charged uniformly to all vertices in P_m. Since P_m is large, each vertex in P_m is charged $O(1)$. Noting that the number of large component classes, $c_\ell = O(1)$, the total amount charged to all vertices during the time they are part of large components is bounded by $O(k\ell)$.

For the small merge case involving components P_1 and P_2 (assigned to C_1 and C_2 respectively), we consider two types of charges. If $U(C_2)$ is sufficient, vertices of the smaller component P_1 are migrated to C_2, and the migration cost of $|P_1|$ is charged to vertices in P_1. Each vertex can be charged $O(\log k)$ many times in this manner before it is part of a large component. For all vertices, this type of charge amounts to $O(k\ell \log k)$. On the other hand, if $U(C_2)$ is insufficient and vertices in $P_1 \cup P_2$ are migrated, the migration cost of $O(|P_m|)$ is charged to all vertices in P_m. However, in this case $m > j$. Since $c_s = O(\log k)$, the total charge of this type for all vertices across the phase is $O(k\ell \log k)$.

As a result, the total migration cost during a phase for both small and large cases during any phase is bounded by $O(k\ell \log k)$. ◀

5 Lower bound for the learning model with arbitrary assignment

In this section, we give a lower bound for any deterministic or randomized algorithm for the learning problem in which the initial assignment of vertices by an offline-optimal algorithm $\mathcal{A}_{OPT}$ and an online algorithm can be arbitrary. Our argument follows an approach implicit in an $\Omega(\log k)$ lower bound established in [12] for randomized OBGR in the learning model.

Let $\Gamma_{\mathcal{A}} = (V_1, V_2, ..., V_\ell)$, where $V_i \subseteq V$ and $|V_i| = k$ for all $i \in [\ell]$ denote an initial assignment of vertices to clusters that an algorithm $\mathcal{A}$ begins with. The initial assignment of vertices that the algorithm $\mathcal{A}_{OPT}$ begins with is analogously defined and denoted by $\Gamma'_{\mathcal{OPT}} = (V'_1, V'_2, ..., V'_\ell)$. Let $\pi : [\ell] \to [\ell]$ denote a permutation of integers in $[\ell]$ and Π denote the set of all such permutations. Define $d(\Gamma_{\mathcal{A}}, \Gamma_{OPT}) = \min_{\pi \in \Pi} \sum_{i=1}^{\ell} |V_{\pi(i)} \backslash V'_{\pi(i)}|$ as the initial distance between vertex assignments that $\mathcal{A}$ and $\mathcal{A}_{OPT}$ begin with respectively. Note that $d(\Gamma_{\mathcal{A}}, \Gamma_{OPT}) \leq kl$. In the learning problem with arbitrary assignments, the initial distance can be arbitrary. We prove the following result.

▶ **Theorem 4.** *For any online deterministic (resp., randomized) algorithm with* $(1+\varepsilon)$*-augmentation for the learning model where* $\varepsilon > 0$ *is an arbitrary constant, there exists a sequence of requests for which the cost (resp., expected cost) is* $\Omega(\alpha k\ell \log k)$*.*

Proof. Let $\mathcal{A}$ denote an algorithm that begins with an initial assignment $\Gamma_{\mathcal{A}}$. We show that there exists an assignment $\Gamma_{\mathcal{OPT}}$ satisfying $d(\Gamma_{\mathcal{A}}, \Gamma_{OPT}) = \Theta(kl)$ such that $\mathcal{A}$ incurs at least $\Omega(kl \log k)$ while $\mathcal{A}_{OPT}$ incurs no cost. The idea is to construct a sequence σ composed of batches B_j of requests for $j = \Omega(\log k)$ such that $\mathcal{A}$ incurs cost $\Omega(kl)$ on each batch. For the sake of the proof, let k be a power of 2. We assume $\epsilon < \ell - 1$ is a constant and $l \geq 2$.

We give some terminology which will be useful. Let $\mathcal{P}_i$ denote the set of components induced by the set of requests $\cup_{j=1}^{i} B_j$. Within any batch, we define a saturating sequence of requests between components P_1 and P_2 as a sequence of requests of the form (u, v) where $u \in P_1, v \in P_2$ for vertices u and v which are not currently co-located by $\mathcal{A}$. By definition a saturating sequence of requests terminates once P_1 and P_2 are co-located by $\mathcal{A}$. Let $C_0 = \{\{u\}|u \in V$ denote the set of singletons before $\mathcal{A}$ services the first request.

For the first batch of requests B_1, each singleton component $\{u\}$ is paired with another component $\{v\}$ such that u and v are not co-located by $\mathcal{A}$ under the initial assignment Γ_A. For all such pairs $\{u\}, \{v\}$, B_1 consists of the union of all saturating sequence of requests between $\{u\}$ and $\{v\}$ until they are co-located. If at any point in time while the current batch of requests is being served, $\mathcal{A}$ does not co-locate any pair of components P_1, P_2, a saturating sequence of requests is issued between P_1 and P_2. Observe that for $\mathcal{A}$ to be competitive, $\mathcal{A}$ must co-locate all request pairs. Moreover, $\mathcal{P}_1$ consists of $\frac{k\ell}{2}$ components of size 2.

For any batch B_j for $j > 1$, we proceed similarly. Each component P of size $\frac{k}{2^{j-1}}$ is paired with another component Q such that P and Q are not co-located by $\mathcal{A}$ before any request in batch B_j is issued. Thereafter, for all pairs of components P and Q, a saturating sequence of requests is issued. Once all pairs have been co-located, the next batch of requests B_{j+1} is served.

Note that since requests are issued between only two components of similar size with size less than k at any given time, there exists an assignment $\Gamma_{\mathcal{OPT}} = (V_1', V_2', ..., V_k')$ which satisfies that for any $u, v \in V_i'$ for all $i \in [\ell]$, no request of the form (u, v) was included in σ. Thus, $\mathcal{A}_{OPT}$ incurs zero cost.

On the other hand, the migration cost incurred by $\mathcal{A}$ on any batch of requests B_j is $\Omega(k\ell)$. To this end, note that for all $j \in [\log k]$, $\mathcal{P}_{j-1}$ consists of exactly $\frac{k\ell}{2^{j-1}}$ components, each of size 2^{j-1}. At any point where batch B_j is issued, $\mathcal{A}$ utilizes at least $\frac{k\ell}{(1+\epsilon)k} = \Omega(\ell)$ clusters to assign components. Thus, there exist $\Omega(\frac{k\ell}{2^{j-1}})$ pairs of components that are not co-located by $\mathcal{A}$ and communication requests during batch B_j necessitate migration of at $\Omega(\frac{k\ell}{2^{j-1}})$ components each of size 2^{j-1}. Thus, the total migration cost incurred by $\mathcal{A}$ to service B_j is $\Omega(\alpha k\ell)$. For all $\Omega(\log k)$ batches, this amounts to $\Omega(\alpha k\ell \log k)$.

A similar approach can be employed to construct a probability distribution over request sequences for which every deterministic algorithm incurs an expected cost of at least $\Omega(\alpha k\ell \log k)$. From Yao's minimax principle [23], this yields a lower bound on the expected cost of any randomized algorithm. The distribution of requests is as follows. As above, the sequence proceeds in batches. The probability distribution for a batch is dependent on the components constructed in the preceding batch. For every batch B_j, two components P and Q of size 2^{j-1} are selected at random. Next, all possible requests are issued between vertices in P (resp. Q) and repeated $\Omega(\alpha)$ times. Then, requests of the form (u, v) where $u \in P, v \in Q$ are issued for all possible u, v and repeated $\Omega(\alpha)$ times. This is repeated for batch B_j until there are no components of size 2^{j-1}. It can be shown that for any batch the expected total cost for any deterministic algorithm is $\Omega(\alpha k\ell)$. Since there are $\Omega(\log k)$ batches, this yields the desired $\Omega(\alpha k\ell \log k)$ lower bound, thus completing the proof. ◀

References

1 Sanjeev Arora, Satish Rao, and Umesh Vazirani. Expander flows, geometric embeddings and graph partitioning. *J. ACM*, 56(2), April 2009. `doi:10.1145/1502793.1502794`.

2 Chen Avin, Marcin Bienkowski, Andreas Loukas, Maciej Pacut, and Stefan Schmid. Dynamic balanced graph partitioning. *SIAM Journal on Discrete Mathematics*, 34(3):1791–1812, 2020. `doi:10.1137/17M1158513`.

3 Theophilus Benson, Aditya Akella, and David A. Maltz. Network traffic characteristics of data centers in the wild. In *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, IMC '10, pages 267–280, New York, NY, USA, 2010. Association for Computing Machinery. `doi:10.1145/1879141.1879175`.

4 Marcin Bienkowski, Martin Böhm, Martin Koutecký, Thomas Rothvoß, Jiří Sgall, and Pavel Veselý. Improved analysis of online balanced clustering. In *Approximation and Online Algorithms: 19th International Workshop, WAOA 2021, Lisbon, Portugal, September 6–10, 2021, Revised Selected Papers*, pages 224–233, Berlin, Heidelberg, 2021. Springer-Verlag. `doi:10.1007/978-3-030-92702-8_14`.

5 Mosharaf Chowdhury, Matei Zaharia, Justin Ma, Michael I. Jordan, and Ion Stoica. Managing data transfers in computer clusters with orchestra. *SIGCOMM Comput. Commun. Rev.*, 41(4):98–109, August 2011. `doi:10.1145/2043164.2018448`.

6 Guy Even, Joseph (Seffi) Naor, Satish Rao, and Baruch Schieber. Fast approximate graph partitioning algorithms. In *Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '97, pages 639–648, USA, 1997. Society for Industrial and Applied Mathematics.

7 Guy Even, Joseph Seffi Naor, Satish Rao, and Baruch Schieber. Divide-and-conquer approximation algorithms via spreading metrics. *J. ACM*, 47(4):585–616, July 2000. `doi:10.1145/347476.347478`.

8 Uriel Feige and Robert Krauthgamer. A polylogarithmic approximation of the minimum bisection. *SIAM Rev.*, 48(1):99–130, January 2006. `doi:10.1137/050640904`.

9 Uriel Feige, Robert Krauthgamer, and Kobbi Nissim. Approximating the minimum bisection size (extended abstract). In *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing*, STOC '00, pages 530–536, New York, NY, USA, 2000. Association for Computing Machinery. `doi:10.1145/335305.335370`.

10 Tobias Forner, Harald Räcke, and Stefan Schmid. Online balanced repartitioning of dynamic communication patterns in polynomial time. In *Symposium on Algorithmic Principles of Computer Systems (APOCS)*, pages 40–54, 2021. `doi:10.1137/1.9781611976489.4`.

11 M. R. Garey, D. S. Johnson, and L. Stockmeyer. Some simplified np-complete problems. In *Proceedings of the Sixth Annual ACM Symposium on Theory of Computing*, STOC '74, pages 47–63, New York, NY, USA, 1974. Association for Computing Machinery. `doi:10.1145/800119.803884`.

12 Monika Henzinger, Stefan Neumann, Harald Räcke, and Stefan Schmid. Tight bounds for online graph partitioning. In *Proceedings of the Thirty-Second Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2799–2818, USA, 2021. Society for Industrial and Applied Mathematics.

13 Monika Henzinger, Stefan Neumann, and Stefan Schmid. Efficient distributed workload (re-)embedding. In *Abstracts of the 2019 SIGMETRICS/Performance Joint International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '19, pages 43–44, New York, NY, USA, 2019. Association for Computing Machinery. `doi:10.1145/3309697.3331503`.

14 Robert Krauthgamer, Joseph (Seffi) Naor, and Roy Schwartz. Partitioning graphs into balanced components. In *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '09, pages 942–949, USA, 2009. Society for Industrial and Applied Mathematics.

15 T. Leighton, F. Makedon, and S.G. Tragoudas. Approximation algorithms for vlsi partition problems. In *IEEE International Symposium on Circuits and Systems*, pages 2865–2868 vol.4, 1990. doi:10.1109/ISCAS.1990.112608.

16 Maciej Pacut, Mahmoud Parham, and Stefan Schmid. Optimal online balanced partitioning. In *INFOCOM 2021*, 2021.

17 Cynthia A. Phillips, Cliff Stein, Eric Torng, and Joel Wein. Optimal time-critical scheduling via resource augmentation (extended abstract). In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, STOC '97, pages 140–149, New York, NY, USA, 1997. Association for Computing Machinery. doi:10.1145/258533.258570.

18 Arjun Roy, Hongyi Zeng, Jasmeet Bagga, George Porter, and Alex C. Snoeren. Inside the social network's (datacenter) network. *SIGCOMM Comput. Commun. Rev.*, 45(4):123–137, August 2015. doi:10.1145/2829988.2787472.

19 A. Schrijver. Theory of linear and integer programming. In *Wiley-Interscience series in discrete mathematics and optimization*, 1999.

20 Horst D. Simon and Shang-Hua Teng. How good is recursive bisection? *SIAM J. Sci. Comput.*, 18(5):1436–1445, September 1997. doi:10.1137/S1064827593255135.

21 Arjun Singh, Joon Ong, Amit Agarwal, Glen Anderson, Ashby Armistead, Roy Bannon, Seb Boving, Gaurav Desai, Bob Felderman, Paulie Germano, Anand Kanagala, Jeff Provost, Jason Simmons, Eiichi Tanda, Jim Wanderer, Urs Hölzle, Stephen Stuart, and Amin Vahdat. Jupiter rising: A decade of clos topologies and centralized control in google's datacenter network. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, SIGCOMM '15, pages 183–197, New York, NY, USA, 2015. Association for Computing Machinery. doi:10.1145/2785956.2787508.

22 Daniel D. Sleator and Robert E. Tarjan. Amortized efficiency of list update and paging rules. *Commun. ACM*, 28(2):202–208, February 1985. doi:10.1145/2786.2793.

23 Andrew Chi-Chin Yao. Probabilistic computations: Toward a unified measure of complexity. In *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*, pages 222–227, 1977. doi:10.1109/SFCS.1977.24.

24 N. Young. Thek-server dual and loose competitiveness for paging. *Algorithmica*, 11(6):525–541, June 1994. doi:10.1007/BF01189992.

25 Neal E. Young. On-line file caching. In *Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '98, pages 82–86, USA, 1998. Society for Industrial and Applied Mathematics.

A From learning to the general model

▶ **Observation 1.** *Any ρ-competitive algorithm for OBGR in the learning model can be transformed to a $O(\rho k\ell)$-competitive algorithm for OBGR in the general model.*

Proof. We give an a ρkl-competitive algorithm $\mathcal{A}$ for OBGR in the general model using the ρ-competitive algorithm $\mathcal{A}_L$ as a subroutine. We say an assignment $\Gamma : V \to \mathcal{C}$ is perfect if every cluster is assigned exactly k vertices. The algorithm partitions the request sequence into phases, and treats each phase as an independent sequence of requests. Here, the definition of a phase is slightly different: phase p of σ is a maximal sub-sequence of requests such that there exists a perfect assignment of vertices which satisfies the property that for all $(u, v) \in p$, $\Gamma(u) = \Gamma(v)$, i.e. u and v are assigned to the same cluster. Before a new phase begins, $\mathcal{A}$ sets $\mathcal{P}$ to the set of singletons and migrates vertices so that every cluster has exactly k vertices. During a phase p, $\mathcal{A}$ simply simulates $\mathcal{A}_L$; $\mathcal{A}_L$ starts with the same assignment of vertices as $\mathcal{A}$ at the beginning of p. Let $\mathcal{A}_{OPT}$ denote an offline-optimal algorithm.

It is easy to observe that the cost incurred by $\mathcal{A}_{OPT}$ increases by at least 1 in every phase. We claim that $\mathcal{A}$ incurs a cost no more than $\rho k\ell$. To this end, suppose $\mathcal{A}$ incurred a cost more than $\rho k\ell$. Consider an algorithm which an identical assignment of vertices as $\mathcal{A}$ at the beginning of phase p and immediately moves to a perfect assignment Γ of vertices such that for any $(u, v) \in p$, $\Gamma(u) = \Gamma(v)$ and incurs no cost thereafter throughout p. The cost of this algorithm is at most $k\ell$ which contradicts that $\mathcal{A}_L$ is ρ-competitive. ◀

B The case of general α

In this section, we show how to adapt our algorithms which were given for $\alpha = 1$ to arbitrary α without a degradation in the asymptotic competitive ratio.

▶ **Theorem 11.** *Let $\mathcal{A} \in \{\mathcal{A}_S, \mathcal{A}_G\}$ denote a $O(\rho)$ competitive algorithm for OBGR for $\alpha = 1$, where $\rho = \Omega(k\ell \log k)$. Then, $\mathcal{A}$ can be modified to an $O(\rho)$ competitive algorithm $\mathcal{A}_M$ to handle the case of arbitrary α.*

Proof. In the case of arbitrary α, merging of two components is beneficial only when sufficient number of requests have been encountered between them. Let $w(P_i, P_j)$ denote the number of requests of the form (u_t, v_t) between components P_i and P_j where $u_t \in P_i, v_t \in P_j$ during a phase. $\mathcal{A}_M$ initializes a phase by setting $\mathcal{P}$ to the set of singletons and $w(\{u\}, \{v\}) = 0$ for all $u, v \in V$. For components P_i and P_j where w.l.o.g. $|P_i| \leq |P_j|$, $\mathcal{A}_M$ merges them into P_m when $w(P_i, P_j) \geq \alpha|P_i|$. For every component $P_r \neq P_i, P_j$, $w(P_m, P_r)$ is set to $w(P_i, P_r) + w(P_j, P_r)$. Due to this reason, it is possible P_r may become eligible to be merged with P_m. A request (u_t, v_t) is *special* if it leads to one or more component merges.

During any phase, $\mathcal{A}_M$ works as follows: on any request (u_t, v_t) between components P_i and P_j it first increments $w(P_i, P_j)$. Next, it determines whether the request is special. If it is special, $\mathcal{A}_M$ simulates $\mathcal{A}$ on this request. Note that if P_i ad P_j are in the same cluster, then nothing needs to be done besides updating data structures and merging P_i and P_j into P_m. However, if this makes a component P_r eligible to be merged with P_m, $\mathcal{A}_M$ creates an artificial request (u^A, v^A) where $u^A \in P_m, v^A \in r$ and simulates the action of $\mathcal{A}$ on (u^A, v^A). Recursive component merges are handled similarly. A phase of $\mathcal{A}_M$ ends whenever a phase of $\mathcal{A}$ ends. Note that requests to $\mathcal{A}$ only consist of special and artificial requests.

We bound the total communication and migration cost incurred by $\mathcal{A}_M$ during a phase. Since $\mathcal{A}$ incurs a cost of $O(\rho)$ per phase, the migration cost of $\mathcal{A}_M$ is bounded by $O(\alpha\rho)$. We claim the communication cost per phase of $\mathcal{A}_M$ is $O(\alpha k\ell \log k)$. For this purpose, consider charging any vertex in a small component P_i a cost of α whenever P_i is merged with P_j. This is sufficient to bound the total communication cost, which is $\alpha|P_i|$ incurred due to communication between vertices in P_i and P_j. Thus, every vertex is charged $O(\alpha \log k)$ per phase yielding a total communication cost of $O(\alpha k\ell \log k)$.

To lower bound the cost of an optimal offline algorithm during the phase, note that either it migrated a vertex or not. If a vertex was migrated during the phase, then $OPT \geq \alpha$. On the other hand, if no vertex was migrated, a communication cost of at least α must have been incurred. To see why, note that at the termination of the phase, the ILP 1 solved by $\mathcal{A}$ determines that no feasible solution exists. Each edge in the graph that $\mathcal{A}$ maintains during the phase corresponds to at least α paid communication requests handled by $\mathcal{A}_M$. Thus, for both cases $OPT \geq \alpha$ per phase.

This yields $O(\rho + k\ell \log k)$ competitiveness. Since $\rho = \Omega(k\ell \log k)$, the theorem follows. ◀