Online Dynamic B-Matching*

With Applications to Reconfigurable Datacenter Networks

Marcin Bienkowski
University of Wrocław, Poland
marcin.bienkowski@cs.uni.wroc.pl

David Fuchssteiner
University of Vienna, Austria
david.alexander.fuchssteiner@univie.ac.at

Jan Marcinkowski
University of Wrocław, Poland
jan.marcinkowski@cs.uni.wroc.pl

Stefan Schmid
University of Vienna, Austria
stefan_schmid@univie.ac.at

ABSTRACT

This paper initiates the study of online algorithms for the maximum weight b-matching problem, a generalization of maximum weight matching where each node has at most $b \geq 1$ adjacent matching edges. The problem is motivated by emerging optical technologies which allow to enhance datacenter networks with reconfigurable matchings, providing direct connectivity between frequently communicating racks. These additional links may improve network performance, by leveraging spatial and temporal structure in the workload. We show that the underlying algorithmic problem features an intriguing connection to online paging (a.k.a. caching), but introduces a novel challenge. Our main contribution is an online algorithm which is $O(b)$-competitive; we also prove that this is asymptotically optimal. We complement our theoretical results with extensive trace-driven simulations, based on real-world datacenter workloads as well as synthetic traffic traces.

Keywords

online algorithms, competitive analysis, b-matching, reconfigurable networks, demand-aware networks

1. INTRODUCTION

1.1 Motivation: Reconfigurable Datacenters

The popularity of distributed data-centric applications related to machine learning and AI has led to an explosive growth of datacenter traffic, and researchers are hence making great efforts to design more efficient datacenter networks, providing a high throughput at low cost. Indeed, over the last years, much progress has been made in the design of innovative datacenter interconnects, based on fat-tree topologies [2, 44], hypercubes [34, 66], expanders [42] or random graphs [61], among many others [38, 67]. All these networks have in common that their topology is static and fixed.

An emerging intriguing alternative to these static datacenter networks are reconfigurable networks [50, 51, 64, 13, 31, 40, 32, 43, 26, 68, 11, 65]: networks whose topology can be changed *dynamically*. In particular, novel optical technologies allow to provide "short cuts", i.e., direct connectivity between top-of-rack switches, based on dynamic matchings. First empirical studies demonstrate the potential of such reconfigurable networks, which can deliver very high bandwidth efficiency at low cost.

The matchings provided by reconfigurable networks are either periodic (e.g., [51, 50, 14]) or demand-aware (e.g., [32, 29, 26, 6, 12, 13, 8]). The latter is attractive as it allows to leverage structure in the demand: datacenter traffic is known to be highly structured, e.g., traffic matrices are typically sparse and some flows (sometimes called elephant flows) much larger than others [57, 15]. This may be exploited: in principle, demand-aware datacenter networks allow to directly match racks which communicate more frequently, leveraging spatial and temporal locality in the workload [5]. These reconfigurable matchings are usually assumed to enhance a given fixed datacenter topology based on traditional electric switches [32]: the remaining traffic (e.g., mice flows) can be routed along the fixed network, e.g., using classic shortest path control planes such as ECMP.

The advent of such hybrid static-dynamic datacenter networks introduces an online optimization problem: how to enhance a given fixed topology with a set of additional short-cut "demand-aware" edges, such that the current demand is served optimally (e.g., large flows are routed along short paths, minimizing the "bandwidth tax" [51]), while at the same time reconfiguration costs are kept minimal (reconfigurations take time and can temporarily lead to throughput loss).

1.2 Problem in a Nutshell

The above problem can be modeled as an online dynamic version of the classic b-matching problem [4] (where b is the number of optical switches). In this problem, each node can be connected with at most b other nodes (using optical links), which results in a b-matching.

Interestingly, while the offline version of the b-matching problem has been studied intensively in the past (e.g., in the context of matching applicants to posts) [60], we are not aware of any work on the dynamic online variant. As the problem is fundamental and finds applications beyond

the reconfigurable datacenter design problem, we present it in the following abstract form.

Input. We are given an arbitrary (undirected) static weighted and connected network on the set of nodes V connected by a set of non-configurable links F: the fixed network. Let V^2 be the set of all possible unordered pairs of nodes from V. For any pair $\tau = \{u, v\} \in V^2$, let ℓ_τ denote the length of a shortest path between nodes u and v in graph $G = (V, F)$. Note that u and v are not necessarily directly connected in F.

The fixed network can be enhanced with reconfigurable links, providing a matching of degree b: Any node pair from V^2 may become a *matching edge* (such an edge corresponds to a reconfigurable optical link), but the number of matching edges adjacent to any node has to be at most b, for a given integer $b \geq 1$.

The demand is modelled as a sequence of communication requests[1] $\sigma = \{s_1, t_1\}, \{s_2, t_2\}, \ldots$ revealed over time, where $\{s_i, t_i\} \in V^2$.

Output and Objective. The goal is to schedule the reconfigurable links over time, that is, to maintain a dynamically changing b-matching $M \subseteq V^2$. Each node pair from M is called a *matching edge* and we require that each node has at most b adjacent matching edges. We aim to jointly minimize routing and reconfiguration costs, defined below.

Costs. The routing cost for a request $\tau = \{s, t\}$ depends on whether s and t are connected by a matching edge. In this model, a given request can either only take the fixed network or a direct matching edge (i.e., routing is segregated [32]). If $\tau \notin M$, the requests is routed exclusively on the fixed network, and the corresponding cost is ℓ_τ (shorter paths imply smaller resource costs, i.e., lower "bandwidth tax" [51]). If $\tau \in M$, the request is served by the matching edge, and the routing costs 0 (note that this is the most challenging cost function: our result only improves if this cost is larger).

Once the request is served, an algorithm may modify the set of matching edges: reconfiguration costs α per each node pair added or removed from the matching M. (The reconfiguration cost and time can be assumed to be independent of the specific edge.)

Online algorithms. An algorithm ON is *online* if it has to take decisions without knowing the future requests (in our case, e.g., which edge to include next in the matching and which to evict). Such an algorithm is said to be ρ-competitive [18] if there exists a constant β such that for any input instance $\mathcal{I}$, it holds that

$$\text{COST}(\text{ON}, \mathcal{I}) \leq \rho \cdot \text{COST}(\text{OPT}, \mathcal{I}) + \beta,$$

where $\text{COST}(\text{OPT}, \mathcal{I})$ is the cost of the optimal (offline) solution for $\mathcal{I}$. It is worth noting that β can depend on the parameters of the network, such as the number of nodes, but has to be independent of the actual sequence of requests. Hence, in the long run, this additive term β becomes negligible in comparison to the actual cost of online algorithm ON.

---

[1] A request could either be an individual packet or a certain amount of data transferred. This model of a request sequence is often considered in the literature and is more fine-grained than, e.g., a sequence of traffic matrices.

1.3 Our Contributions

This paper initiates the study of a natural problem, online dynamic b-matching. For example, this problem finds applications in the context of emerging reconfigurable datacenter networks.

We make the following contributions:

- We show that the online dynamic b-matching problem features an interesting connection to online paging problems, however, with a twist, introducing a new challenge.

- We present an $O((1 + \ell_{\max}/\alpha) \cdot b)$-competitive deterministic algorithm, where $\ell_{\max} = \max_{e \in V^2} \ell_e$. Note that in all relevant practical applications $\ell_{\max} \ll \alpha$ (i.e., the cost of routing between any two nodes is much smaller than the cost of reconfiguration, and hence the term $\ell_{\max}/\alpha$ is negligible).

- We derive a lower bound which shows that no deterministic algorithm can achieve a competitive ratio better than b.

- We verify our approach experimentally, performing extensive trace-driven simulations, based on real datacenter workloads as well as synthetic traffic traces.

1.4 Challenges, Technical Novelty, Scope

At the heart of our approach lies the observation that online b-matching is similar to online paging [28, 62, 1, 46]: each node in the network can manage its reconfigurable edges in a cache of size b. However, making a direct reduction to caching seems impossible as reconfigurable edges involve *both* incident nodes, which introduces non-trivial dependencies. Without accounting for these dependencies, the competitive ratio would be in the order of the total number of reconfigurable edges in the network, whereas we in this paper derive results which only depend on the number of per-node edges. Our algorithms hence combine "per-node caches" in a clever way.

Generally, we believe that the notion of link caching has interesting implications for reconfigurable network designs beyond the model considered in this paper. In particular, caching strategies can typically be implemented locally, and hence may allow to overcome centralized control overheads, similar to the stable matching algorithms proposed in the literature [32]. However, we leave the discussion of of such decentralized schedulers to future work.

1.5 Organization

The remainder of this paper is organized as follows. Our online algorithm is described and analyzed in Section 2, and the lower bound is presented in Section 3. We report on our simulation results in Section 4. After reviewing related literature in Section 5, we conclude our contribution in Section 6.

2. ALGORITHM BMA

This section introduces our online b-matching algorithm, together with a competitive analysis. As described above, in our case study of reconfigurable networks, the matching links may for example describe the reconfigurable links provided by optical circuit switches, offering shortcuts between datacenters racks.

Algorithm 1 Algorithm BMA

1: **Initialization:**   ▷ *Matching is empty and counters are zero*
2:      $M \leftarrow \emptyset$
3:      **for** each edge e **do**
4:          $h_e \leftarrow 0$
5:
6: **Request** $\tau = \{u, v\}$ **arrives:**
7:      **if** $\tau \notin M$ **then**
8:          $h_\tau \leftarrow h_\tau + 1$
9:          **if** $h_\tau = T_\tau$ **then**   ▷ *If τ becomes saturated,*
10:              Execute FIXSATURATION(u, τ)
11:              Execute FIXSATURATION(v, τ)
12:              **if** $h_\tau = T_\tau$ **then**   ▷ *and if no desaturation event occured,*
13:                  Execute FIXMATCHING(u)
14:                  Execute FIXMATCHING(v)
15:                  $M \leftarrow M \cup \{\tau\}$ ▷ *add τ to the matching.*
16:
17: **Routine** FIXSATURATION(w, τ):
18:      $E'_w = E_w \setminus \{\tau\}$
19:      **if** $|E'_w \cap \{e : h_e = T_e\}| \geq b$ **then**   ▷ *If the number of saturated node pairs from E'_w is at least b,*
20:          **for** each edge $e \in E_w$ **do**   ▷ *reset counters of all node pairs from E_w (desaturation event at w).*
21:              $h_e \leftarrow 0$
22:
23: **Routine** FIXMATCHING(w):
24:      **if** $|M \cap E_w| = b$ **then**   ▷ *If there are already b incident matching edges,*
25:          Pick any $e^* \in M \cap E_w$ such that $h_{e^*} < T_{e^*}$ ▷ *remove any unsaturated edge e^* from the matching.*
26:          $M \leftarrow M \setminus \{e^*\}$

Before we present the algorithm and its analysis in details, let us first provide some intuition of our approach and the underlying challenges. To this end, let us for now assume that the fixed network $G = (V, F)$ is a complete unweighted graph (e.g., capturing the distances in a datacenter network), that $\alpha = 1$, and that all requests are node pairs involving one chosen node w. Also recall that each node can have at most b incident matching edges.

In this simplified scenario, we can observe that the choice of an appropriate set of matching edges becomes essentially a variant of online caching (more precisely, a variant of online paging with bypassing [25]). That is, an algorithm maintains a set of at most b edges incident to w that are in the matching. These edges can be thought as "cached": subsequent requests to matched (cached) edges do not incur further cost. Thus, from the perspective of a single node, the question is roughly equivalent to maintaining a cache of at most b items (which leads to the typical algorithmic questions such as which item to cache or evict next).

However, if we simply run independent paging algorithms at all nodes, local perspectives of particular nodes might not be coherent with each other: one endpoint of a node may want to keep an edge in the matching, while the other may want to evict it from the matching. This is practically undesirable, as transmitters and receivers typically need to be aligned and coordinated [51, 32]. To illustrate this issue, assume that node w wants to add a new matching edge $\{w, w'\}$, but it already has b incident matched edges. To accommodate a new matching edge, w removes edge $\{w, w''\}$ from the matching. This however removes the edge not only from the "cache" of node w, but also from the "cache" of node w''. Handling this coherence issue with a low overall

cost, constitutes a main technical challenge that we need to tackle in our algorithm.

2.1 Algorithm Definition

Our algorithm BMA is defined as follows. For each node pair $e \in V^2$, BMA keeps a counter h_e, initially equal to zero. The value of h_e will always be a lower bound for the number of times e has been requested since it was removed from the matching the last time (or from the beginning of the input sequence if e was never in the matching). For each node pair $e \in V^2$ we define a threshold

$$T_e = 2 \cdot \lceil \alpha / \ell_e \rceil.$$

Once the counter h_e reaches T_e, and additional certain conditions are fulfilled, edge e will be added to the matching. Otherwise, the counter will be reset to zero. A node pair $e \in V^2$ whose counter value is equal to T_e is called *saturated*; our algorithm always keeps all saturated node pairs in the matching. Note that these T_e requests to node pair e induced a total cost of $T_e \cdot \ell_e \in [2\alpha, 2(\alpha / \ell_e + 1) \cdot \ell_e] = [2\alpha, 2\alpha + 2\ell_e]$.

For any node w, we define $E_w = \{\{w, v\} : v \in V\}$, i.e., $E_w \subseteq V^2$ is the set of all node pairs, with one node equal to w. Recall that, at any time, $M \subseteq V^2$ denotes the set of matching edges.

BMA is designed to preserve three invariants:

Counter invariant: $0 \leq h_e \leq T_e$.

Saturation invariant: If $h_e = T_e$, then $e \in M$.

Matching invariant: If $e \in M$, then $h_e = T_e$ or $h_e = 0$.

plus an invariant for any node w:

Saturation degree invariant: $|\{e \in E_w : h_e = T_e\}| \le b$.

A pseudo-code for our algorithm BMA is given in Algorithm 1. In the next section, we will explain it in more details, and prove that it never violates the invariants.

2.2 Maintaining Invariants

At the beginning, the matching M is empty and the counters of all edges are zero, and thus all invariants hold. Below we describe what happens upon serving a communication request τ by BMA and how BMA ensures that all invariants are preserved.

BMA first verifies whether τ is a matching edge. If so, then such a request incurs no cost and BMA does nothing. Otherwise, by the saturation invariant, $h_\tau \le T_e - 1$. In this case, BMA pays for the communication request and increments counter h_τ. The increment preserves the counter invariant. The matching invariant holds emptily as $\tau \notin M$. If h_τ is still below T_e, then the remaining invariants also hold and BMA does not execute any further actions.

However, if the value of h_τ reaches T_τ (τ becomes saturated), the saturation invariant becomes violated (τ should be a matching edge) and also the saturation degree invariants may become violated at u and v. We now explain how BMA handles these issues.

BMA first ensures that the saturation degree invariant is satisfied at both endpoints of τ. To this end, BMA executes the FixSaturation routine at u and v. If at any node $w \in \{u, v\}$ the number of saturated node pairs from E_w different from τ is already b, all edges of E_w, including τ, have their counters reset to zero. In this case, we say that a *desaturation event* occurred at the respective endpoint w. Note that all four cases are possible: there can be no desaturation event, a desaturation event can occur at u, at v, or at both of u and v. The execution of the FixSaturation routines reestablishes the saturation degree invariants, and preserves counter, saturation and matching invariants at all edges different from τ.

For edge τ, the corresponding counter and matching invariants clearly hold. If any desaturation event occurs, then it also fixes the saturation invariant for τ, and BMA need not do anything more. Otherwise (no desaturation event occurs, that is, h_τ is still equal to T_τ), the saturation invariant is violated: τ has to be added to the matching. However, if any of τ endpoints already has b incident matching edges, one such edge has to be removed from the matching before. This is achieved by the FixMatching routines executed at both u and v: if necessary, they remove one incident non-saturated matching edge. It remains to show that such a matching edge indeed exists.

LEMMA 1. *A non-saturated matching edge e^* chosen at Line 25 of the algorithm BMA (in routine FixMatching(w)) always exists. Moreover, when it is removed, $h_{e^*} = 0$.*

PROOF. Assume that FixMatching(w) is executed (for a node $w \in \{u, v\}$). Let $S_w = E_w \cap \{e : h_e = T_e\}$ be the set of saturated node pairs from E_w. Note that FixMatching(w) is preceded by the execution of FixSaturation(w, τ): it ensures that $|S_w| \le b$ (S_w contains τ and at most $b - 1$ other edges).

Let $M_w = M \cap E_w$ be the set of matching edges incident to w. The condition at Line 24 ensures that $|M_w| = b$.

Now, observe that the set $S_w \setminus M_w$ is non-empty as it contains the requested node pair τ. However, as $|M_w| \ge |S_w|$, the set $M_w \setminus S_w$ is non-empty either, and any of its edges is a viable candidate for e^*.

Finally, by the matching invariant, the counter of a matching edge $e \in M_w$ is equal either to T_e or 0. Hence, the counters of all matching edges in $M_w \setminus S_w$ are zero, and thus $h_{e^*} = 0$. □

2.3 Desaturation Events

Fix any node w. For the analysis of BMA, a natural approach would be to estimate the number of paid requests to all node pairs of E_w between two desaturation events at w. This number corresponds to the total increase of all counters corresponding to these node pairs in the considered time interval. However, such an approach fails as these counters may be reset multiple times because of desaturation events at other nodes. In particular, it is possible that a node pair $\{w, u\}$ from E_w in included multiple times in the matching between two desaturation events at w. Therefore, we develop a more complicated accounting scheme.

First, not only we track counters, but for any node pair e we keep track of a set H_e of requests paid by BMA (that caused the increase of the counter h_e, i.e., $|H_e| = h_e$). When the counter h_e is reset, the set H_e is emptied.

When requests paid by BMA become removed from sets H_e, we map them to the corresponding desaturation events: for any desaturation event d, we create a set of requests A_d, so that all these sets are disjoint. Requests that still belong to the current contents of sets H_e are not (yet) mapped. More precisely, note that when a desaturation event at a node w occurs, we empty all sets H_e for node pairs $e \in E_w$. If a request $\tau = \{u, v\}$ triggers a single desaturation event d_u at u, then we simply set $A_{d_u} = \biguplus_{e \in E_u} H_e$, i.e., we map all requests corresponding to counters that were reset by d_u. If, however, a request $\tau = \{u, v\}$ triggers desaturation events d_u and d_v both at u and v, we want requests from H_τ to be mapped (partially) to both desaturation events. Thus, we partition T_τ requests from H_τ arbitrarily into two subsets H_τ^v and H_τ^u, each of cardinality $T_\tau/2 = \lceil \alpha/\ell_e \rceil$, and set $A_{d_u} = H_\tau^u \cup \biguplus_{e \in E_u \setminus \tau} H_e$ and $A_{d_v} = H_\tau^v \cup \biguplus_{e \in E_v \setminus \tau} H_e$.

For any request set P, let $\ell(P) = \sum_{e \in P} \ell_e$, i.e., $\ell(P)$ is the cost of serving all requests from P without using matching edges. For any desaturation event d and a node pair e, let $A_d(e)$ be the requests of A_d to node pair e. The following observation follows immediately by the definition of BMA and sets A_d.

OBSERVATION 1. *Fix any desaturation event d at any node w. Then, the following properties hold:*

1. *For any node pair $e \in E_w$, it holds that $|A_d(e)| \le T_e$.*

2. *There exists a set $P \subseteq E_w$ of cardinality $b + 1$, such that $|A_d(e)| \ge T_e/2$ for each $e \in P$.*

2.4 Competitive Ratio of BMA

We now use sets A_d to estimate the costs of BMA and OPT. We do not aim at optimizing the constants, but rather at the simplicity of the argument.

LEMMA 2. *Let $D(\mathcal{I})$ be the set of all desaturation events that occurred during input $\mathcal{I}$. Then,*

$$\text{COST}(\text{BMA}, \mathcal{I}) \le 4 \cdot |V^2| \cdot (\alpha + \ell_{\max}) + 2 \sum_{d \in D(\mathcal{I})} \ell(A_d).$$

PROOF. Within this proof, we consider contents of sets H_e right after BMA processes the whole input $\mathcal{I}$. For any node pair $e \in V^2$, the set H_e contains at most T_e edges, and therefore

$$\ell(H_e) \leq T_e \cdot \ell_e \leq 2 \cdot \alpha + 2 \cdot \ell_e \leq 2 \cdot (\alpha + \ell_{\max}).$$

Any request to a node pair e paid by BMA is either in set H_e or it was already assigned to a set A_d for some desaturation event $d \in D(\mathcal{I})$. Thus, the cost of serving all requests by BMA is at most

$$\sum_{e \in V^2} \ell(H_e) + \sum_{d \in D(\mathcal{I})} \ell(A_d) \leq 2 \cdot |V^2| \cdot (\alpha + \ell_{\max}) + \sum_{d \in D(\mathcal{I})} \ell(A_d).$$

To bound the cost of matching changes, we observe that by the definition of BMA, only a saturated node pair e may become included in the matching. If e becomes removed from the matching later, then by Lemma 1, the counter of e must have dropped to zero in the meantime. Therefore, any addition of e to the matching can be mapped to the unique T_e paid requests to e. As the cost of such T_e requests is $T_e \cdot \ell_e \geq 2 \cdot \alpha$, the total cost of including e in the matching is dominated by the half of the cost of serving requests to e. Furthermore, as the number of removals from the matching cannot be larger than the number of additions, the total cost of excluding e from the matching is also dominated by the same amount. Summing up, the matching reconfiguration cost of BMA is not larger than its request serving cost, i.e.,

$$\text{COST}(\text{BMA}, \mathcal{I}) \leq 4 \cdot |V^2| \cdot (\alpha + \ell_{\max}) + 2 \sum_{d \in D(\mathcal{I})} \ell(A_d),$$

which concludes the lemma. $\square$

LEMMA 3. *Let $D(\mathcal{I})$ be the set of all desaturation events that occurred during input $\mathcal{I}$. Then*

$$\text{COST}(\text{OPT}, \mathcal{I}) \geq \frac{1}{3 \cdot (b+1) \cdot (1 + \ell_{\max}/\alpha)} \sum_{d \in D(\mathcal{I})} \ell(A_d).$$

PROOF. To estimate the cost of OPT, it is more convenient to think that its cost is not associated with node pairs but with nodes. That is, we distribute the cost of OPT pertaining to node pairs (paying for a request, including an edge in the matching or removing an edge from the matching) equally between the endpoints: When OPT pays ℓ_τ for a request at node pair $\tau = \{u, v\}$, we account cost $\ell_\tau/2$ for node u and cost $\ell_\tau/2$ for node v. When OPT pays α for including node pair $\{u, v\}$ into the matching or excluding it from the matching, we associate cost $\alpha/2$ with node u and $\alpha/2$ with node v.

Now, fix a desaturation event d at a node w. Let d_0 be the previous desaturation event at node w. (If d is the first desaturation event at w, then d_0 is the beginning of the input $\mathcal{I}$.) Note that all requests of A_d appeared between d_0 and d.

Let the node cost (in OPT's solution) of w between d_0 and d be denoted $\text{COST}(\text{OPT}, d)$. As each node-cost paid by OPT is covered by at most one term $\text{COST}(\text{OPT}, d)$, it holds that $\text{COST}(\text{OPT}, \mathcal{I}) \geq \sum_{d \in D(\mathcal{I})} \text{COST}(\text{OPT}, d)$. Hence, our goal is to lower bound the value of $\text{COST}(\text{OPT}, d)$ for any desaturation event d (at some node w).

We sort the edges from E_w by the cost of serving them between d_0 and d. That is, let $E_w = \{e_1, e_2, \ldots, e_{|V|-1}\}$ and

$$\ell(A_d(e_1)) \geq \ell(A_d(e_2)) \geq \cdots \geq \ell(A_d(e_{|V|-1})).$$

Let k be the number of node pairs from E_w that OPT added to the matching between d_0 and d. The corresponding node cost of w due to matching changes is then at least $k \cdot \alpha/2$. Then, the total number of all node pairs from E_w that OPT may have in the matching at some time between d_0 and d is at most $b + k$. Therefore, OPT pays for requests from A_d to all node pairs but at most $b + k$ node pairs, i.e.,

$$\text{COST}(\text{OPT}, d) \geq k \cdot \alpha/2 + \ell(A_d) - \sum_{j=1}^{b+k} \ell(A_d(e_j)).$$

To lower-bound this amount, we first observe that for any $k \geq 0$, it holds that

$$3 \cdot (b+1) \cdot [\ell(A_d(e_{b+k+1})) + k \cdot \alpha/2] \geq (b+k+1) \cdot \alpha \quad (1)$$

Indeed, if $k = 0$, then by Property 2 of Observation 1), $\ell(A_d(e_{b+1})) \geq \alpha/2$, and thus (1) follows. If $k \geq 1$, then $(3/2) \cdot (b+1) \cdot k \geq b + k + 1$ holds for any $b \geq 1$, which implies (1).

Second, by Property 1 of Observation 1, for each node pair $e \in E_w$, it holds that

$$\ell(A_d(e)) \leq T_e \cdot \ell_e \leq 2 \cdot (\alpha + \ell_{\max}) = 2\alpha \cdot (1 + \ell_{\max}/\alpha). \quad (2)$$

Therefore, using (1) and (2), we obtain that

$$\text{COST}(\text{OPT}, d) \geq k \cdot \alpha/2 + \sum_{j=b+k+1}^{|V|-1} \ell(A_d(e_j))$$

$$= k \cdot \alpha/2 + \ell(A_d(e_{b+k+1})) + \sum_{j=b+k+2}^{|V|-1} \ell(A_d(e_j))$$

$$\geq \frac{(b+k+1) \cdot \alpha}{3 \cdot (b+1)} + \sum_{j=b+k+2}^{|V|-1} \ell(A_d(e_j))$$

$$\geq \frac{\sum_{j=1}^{b+k+1} \ell(A_d(e_j))}{6 \cdot (b+1) \cdot (1 + \ell_{\max}/\alpha)} + \sum_{j=b+k+2}^{|V|-1} \ell(A_d(e_j))$$

$$\geq \frac{1}{6 \cdot (b+1) \cdot (1 + \ell_{\max}/\alpha)} \cdot \sum_{j=1}^{|V|-1} \ell(A_d(e_j))$$

$$= \frac{\ell(A_d)}{6 \cdot (b+1) \cdot (1 + \ell_{\max}/\alpha)}.$$

Summing this relation over all desaturation events from the input $\mathcal{I}$ and using the relation

$$\text{COST}(\text{OPT}, \mathcal{I}) \geq \sum_{d \in D(\mathcal{I})} \text{COST}(\text{OPT}, d)$$

yields the lemma. $\square$

THEOREM 1. BMA *is $O((1 + \ell_{\max}/\alpha) \cdot b)$-competitive.*

PROOF. Fix any input instance I and let $D(\mathcal{I})$ be the number of desaturation events that occurred when BMA was executed on $\mathcal{I}$. By Lemmas 2 and 3, we immediately obtain that $\text{COST}(\text{BMA}, \mathcal{I}) \leq 12 \cdot (b+1) \cdot (1 + \ell_{\max}/\alpha) \cdot \text{COST}(\text{OPT}, \mathcal{I}) + 4 \cdot |V^2| \cdot (\alpha + \ell_{\max})$, i.e., the competitive ratio is at most $12 \cdot (b+1) \cdot (1 + \ell_{\max}/\alpha) = O((1 + \ell_{\max}/\alpha) \cdot b)$. $\square$

3. LOWER BOUND

THEOREM 2. *The competitive ratio of any deterministic algorithm DET is at least b.*

PROOF. Let our graph be a star of $b+2$ nodes $v_0, v_1, \ldots,$ v_b, v_{b+1} and non-reconfigurable edge set

$$F = \{ (v_0, v_1), (v_0, v_2), \ldots, (v_0, v_{b+1}) \}.$$

Each edge of F has length 1. We start with any matching that connects v_0 to b leaves. At any time, the adversary chooses v_i which is not currently matched with v_0, and requests a node pair (v_0, v_i) for α times. These α requests constitute one chunk.

For each chunk, DET pays at least α: either for modifying the matching or for bypassing all α requests. An offline algorithm OFF (that knows the entire input sequence) could however make a smarter selection of an edge to remove from the matching: OFF chooses the one which is not going to be requested in the nearest b rounds. Hence, OFF pays at most $\alpha \cdot \lceil k/b \rceil$ for k chunks of the input. For growing k, the ratio between the costs of DET and OFF becomes arbitrarily close to b, and hence the lemma follows. □

4. SIMULATIONS

In order to complement our theoretical contribution and analytical results on the competitive ratio in the worst case, we conducted extensive simulations, evaluating our algorithms on real-world traffic traces. In the following, we report on our main results.

4.1 Methodology

All our algorithms are implemented in Python (3.7.3), using the graph library NetworkX (2.3.2). All simulations were conducted on a machine with two Intel Xeon E5-2697V3 processors with 2.6 GHz, 128 GB RAM, and 14 cores each.

Our simulations are based on the following workloads:

- *Facebook [57]:* We use the batch processing trace (Hadoop) from one of Facebook datacenters, as well as traces from one of Facebook's database clusters.

- *Microsoft [32]:* This data set is simply a probability distribution, describing the rack-to-rack communication (a traffic matrix). In order to generate a trace, we sample from this distribution *i.i.d.* Hence, this trace does not contain any temporal structure by design (e.g., is not bursty) [5]. However, it is known that it contains significant spatial structure (i.e., is skewed).

- *pFabric [3]:* This is a synthetic trace and we run the NS2 simulation script obtained from the authors of the paper to generate a trace.

In order to evaluate our algorithm, we are comparing four different scenarios in our simulations:

- *Oblivious:* The network topology is fixed and not optimized towards the workload by adding reconfigurable links.

- *Static:* The network topology is enhanced with an optimal static b-matching, computed with the perfect knowledge of the workload ahead of time.

- *Online BMA:* The online algorithm described in this paper.

- *LRU BMA:* Like online BMA, however, the cache is now managed according to a least-recently used (LRU)

strategy. In other words, when a link needs to be cached and the cache is full, the least recently used link in the cache is evicted.

For all simulations, we assume a Clos-like datacenter topology [2], connecting 100 servers (leaf nodes of the Clos topology). In addition, the number of requests for each of our simulations depends on the actual trace, therefore the simulations on the Facebook cluster have a slightly different amount of requests than e.g., the Microsoft trace data. Each test run was performed with six different request counts. The simulations were repeated 5 times, each time with a different subset of the whole data set to account for certain variance in the data; the presented results are averaged over these simulation runs. We evaluated our algorithms with several values for $b \in \{4, 8, 12\}$ and $\alpha = 6$. Note that for larger b, less traffic will be routed over the static network, given our cost function. Given this, and the fact that reconfigurable links require space, we will be particularly interested in relatively small values of b: only a small fraction of all possible $n \cdot (n-1)$ links is actually used. Evaluating the effectiveness of small values for b is hence not only more interesting, but also more realistic.

4.2 Results

In order to study to which extent the *Online BMA* algorithm can leverage the temporal locality available in traffic traces, we first consider the effectiveness of the link cache, as a microbenchmark. Figure 1 shows a comparison of the hit ratio of Facebook's database traces (left), pFabric traces (right) and Microsoft traces. We can observe that in the case of the pFabric and Microsoft traces, a relatively high hit ratio is obtained after a short warm-up period, especially if a least-recently-used (LRU BMA) caching strategy is used. We can also observe that our online algorithm performs better under the pFabric and Microsoft traces, which is expected: empirical studies have already shown that these traces feature more structure than the batch processing traces [5]. We also find that the results naturally depend on the cache size, see Figure 2 (left). An important remark is, that the degree b need to be understood relative to the total number of switch ports, i.e., similar results are obtained for relatively larger b values.

It is interesting to compare the results of our online algorithms to demand-oblivious topologies as well as to static topologies. Figure 2 gives a comprehensive overview of our algorithms performance in terms of route lengths (left and right plot) and also regarding the cache hit ratio (middle plot) for different cache sizes for the Facebook Hadoop cluster. Notably, Figure 2 (left and middle plot) gives insights into our algorithm's performance over all 5 test runs, illustrating the average result, as well as the maximum and minimum result (shaded areas).

As expected, *Oblivious* always performs worse than *Static*, *Online BMA* and *LRU BMA*. We further observe that the performance of *Online BMA* comes close to the performance of *Static*, which knows the demands *ahead of time* (but is fixed). We expect that under longer request sequences, when larger shifts in the communication patterns are likely to appear, the online approach will outperform the static offline algorithm. To investigate this, however, the publicly available traffic traces are not sufficient.

While the Microsoft trace does not contain temporal structure as it is sampled i.i.d., it can still be exploited to-
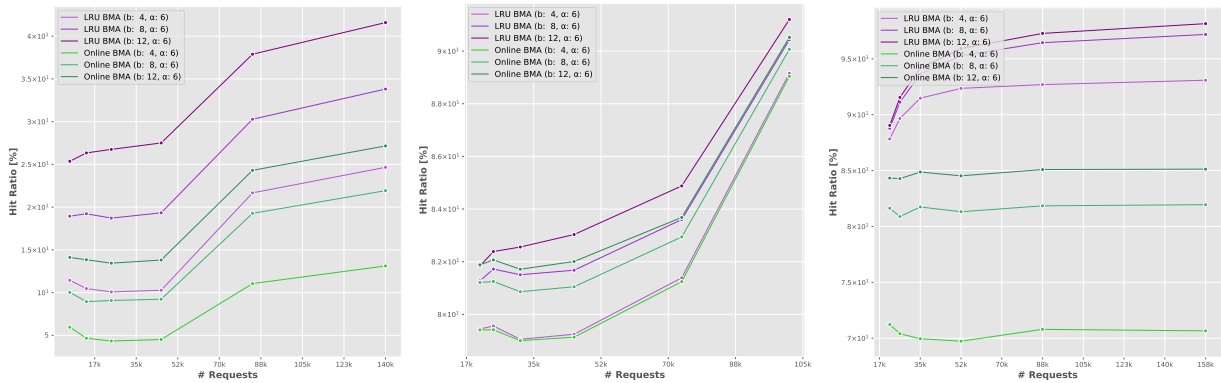
Figure 1: *Left: Hit ratio for Facebook database cluster trace (with lower temporal locality), Middle: pFabric trace (with high temporal locality), Right: Microsoft trace (with high spatial locality).*
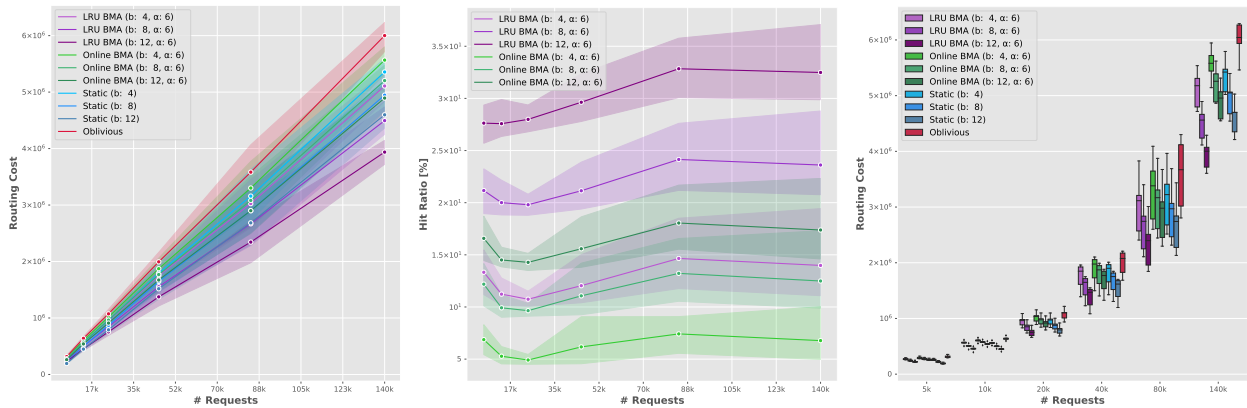

Figure 2: *Left and Right: Facebook Hadoop cluster: routing costs. Middle: Facebook Hadoop cluster: hit ratio for different cache sizes.*

ward a more efficient routing and yield a very high cache hit ratio, due to its spatial structure, i.e., the skewed traffic matrix. See Figure 3.

In conclusion, while our main contribution in this paper concerns the theoretical result, we observe that our online algorithm performs fairly well under real-world workloads, even without further optimizations (besides an improved cache eviction strategy).

5.   RELATED WORK

Reconfigurable networks based on optical circuit switches, 60 GHz wireless, and free-space optics, have received much attention over the last years [32, 43, 26, 13, 68, 63, 12], see also the recent survey for an overview [31]. For an overview of the theoretical foundations of demand-aware networks, we refer the reader to [11]. It has been shown empirically that reconfigurable networks can achieve a performance similar to a demand-oblivious full-bisection bandwidth network at significantly lower cost [13, 32]. Furthermore, the study of reconfigurable networks is not limited to datacenters and interesting use cases also arise in the context of wide-area networks [36, 37] and overlays [58, 56].

Our paper is primarily concerned with the *algorithmic* problems introduced by such technologies. The basic prob-

lem of how to optimally enhance a given fixed datacenter topology with reconfigurable links has already been studied in the literature, both in the static setting, e.g.., [27, 30, 29], as well as in the dynamic setting [12, 24, 64]. For example, Avin et al. [12] presented a statically optimal dynamic network, ReNets, which finds an optimal tradeoff between the benefits and costs of reconfigurations. The static problem is related to *graph augmentation* models, which consider the problem of adding edges to a given graph, so that path lengths are reduced. For example, Meyerson and Tagiku [52] study how to add "shortcut edges" to minimize the average shortest path distances, Bilò et al. [16] and Demaine and Zadimoghaddam [21] study how to augment a network to reduce its diameter, and there are several interesting results on how to add "ghost edges" to a graph such that it becomes (more) "small world" [54, 55, 33]. However, these edge additions can be optimized globally and in a biased manner, and hence do not form a matching. In particular, it is impractical (and does not scale) to add many flexible links per node in practice. The dynamic setting is related to classic switch scheduling problems [47, 20].

Another line of related works considers the design of demand-aware networks from scratch [8, 35, 7, 9], ignoring the fixed topology which is available in current architectures (and in the near future). Also for this model, dynamic ap-
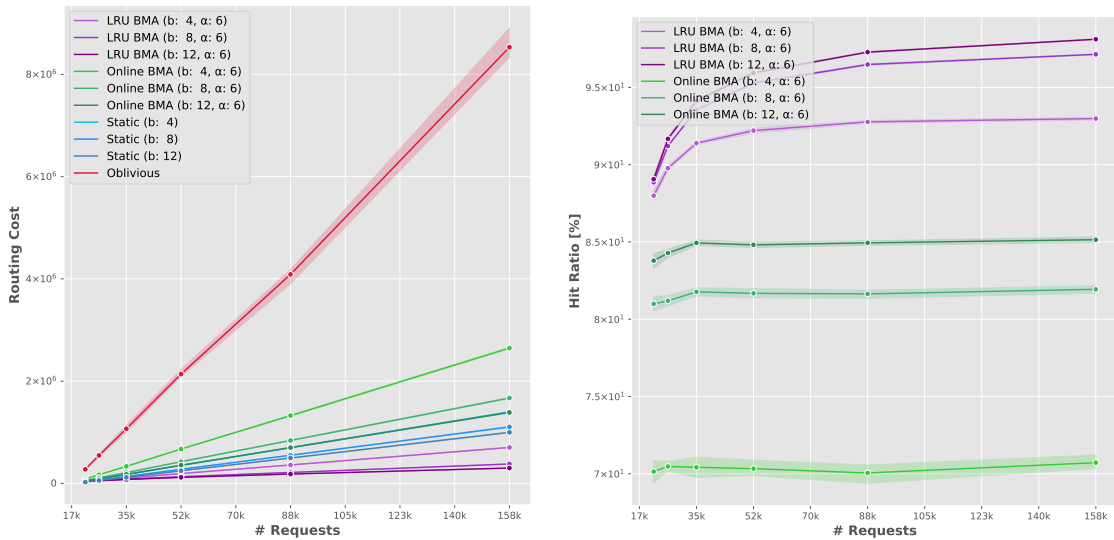
Figure 3: *Left:* Microsoft ProjecToR: routing costs. *Right:* Microsoft ProjecToR: hit ratio.

proaches which come with provable guarantees, have been proposed in the literature, e.g., SplayNets [59] and Push-Down Trees [10].

In this paper, we initiated the study of an online version of the dynamic b-matching problem. A polynomial-time algorithm for the static version of this problem has already been presented over 30 years ago [60, 4], and the problem still receives attention today due to its numerous applications, for example in settings where customers in a market need to be matched to a cardinality-constrained set of items (e.g., matching children to schools, reviewers to papers, or donor organs to patients), but also in protein structure alignment, computer vision, estimating text similarity, etc.

Note that there is a line of papers studying (bipartite) online matching variants [41, 17, 19, 22, 23, 45, 49, 53]. This problem attracted significant attention in the last decade because of its connection to online auctions and the AdWords problem [48]. Despite similarity in names (e.g., the bipartite (static) b-matching variant was considered in [39]), this model is fundamentally different from ours. That is, it considers bipartite graphs in which nodes and (weighted) edges appear in time and the algorithm has to choose a subset of edges being a matching. In our scenario, the (non-bipartite) graph is given a priori, and the algorithm has to *maintain a dynamic* matching. One way of looking at our scenario is to consider the case where edges weights can change over time and the matching maintained by an algorithm needs to catch up with such changes.

6. CONCLUSION

Motivated by emerging reconfigurable datacenter networks whose topology can be dynamically optimized toward the workload, we initiated the study of a fundamental problem, online b-matching. In particular, we presented competitive online algorithms which find an optimal trade-off between the benefits and costs of reconfiguring the matching. While our main contribution concerns the derived theoretical results (i.e., the competitive online algorithm and the lower bound), we believe that our approach has several in-

teresting practical implications: our algorithm is simple to implement, has a low runtime and, as we have shown, performs fairly well also under different real-world workloads and synthetic traffic traces.

Our work opens several interesting avenues for future research. In particular, we have so far focused on deterministic algorithms, and it would be interesting to explore randomized approaches; in fact, our first investigations in this direction indicate that the randomized setting is more challenging, also due to the introduced dependencies, and exploiting the connection to online paging is difficult. On the practical side, it would be interesting to investigate specific reconfigurable optical technologies as well as specific datacenter topologies (such as Clos topologies) in more details, and tailor our algorithms and develop distributed implementations for an optimal performance in this case study.

7. REFERENCES

[1] D. Achlioptas, M. Chrobak, and J. Noga. Competitive analysis of randomized paging algorithms. *Theoretical Computer Science*, 234(1–2):203–218, 2000.

[2] M. Al-Fares, A. Loukissas, and A. Vahdat. A scalable, commodity data center network architecture. In *Proc. ACM SIGCOMM*, pages 63–74, 2008.

[3] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. McKeown, B. Prabhakar, and S. Shenker. pfabric: Minimal near-optimal datacenter transport. *ACM SIGCOMM Computer Communication Review*, 43(4):435–446, 2013.

[4] R. P. Anstee. A polynomial algorithm for b-matchings: an alternative approach. *Information Processing Letters*, 24(3):153–157, 1987.

[5] C. Avin, M. Ghobadi, C. Griner, and S. Schmid. On the complexity of traffic traces and implications. In *Proc. ACM SIGMETRICS*, 2020.

[6] C. Avin, C. Griner, I. Salem, and S. Schmid. An online matching model for self-adjusting tor-to-tor networks. *arXiv preprint arXiv:2006.11148*, 2020.

[7] C. Avin, A. Hercules, A. Loukas, and S. Schmid. *rDAN*: Toward robust demand-aware network designs. *Information Processing Letters*, 133:5–9, 2018.

[8] C. Avin, K. Mondal, and S. Schmid. Demand-aware network designs of bounded degree. In *Proc. Int. Symp. on Distributed Computing (DISC)*, volume 91 of *LIPIcs*, pages 5:1–5:16, 2017.

[9] C. Avin, K. Mondal, and S. Schmid. Demand-aware network design with minimal congestion and route lengths. In *Proc. IEEE Int. Conference on Computer Communications (INFOCOM)*, pages 1351–1359, 2019.

[10] C. Avin, K. Mondal, and S. Schmid. Dynamically optimal self-adjusting single-source tree networks. In *Proc. Latin American Theoretical Informatics Symposium (LATIN)*, 2020.

[11] C. Avin and S. Schmid. Toward demand-aware networking: a theory for self-adjusting networks. *ACM SIGCOMM Computer Communication Review*, 48(5):31–40, 2018.

[12] C. Avin and S. Schmid. Renets: Toward statically optimal self-adjusting networks. *arXiv preprint arXiv:1904.03263*, 2019.

[13] N. H. Azimi, Z. A. Qazi, H. Gupta, V. Sekar, S. R. Das, J. P. Longtin, H. Shah, and A. Tanwer. Firefly: a reconfigurable wireless data center fabric using free-space optics. In *Proc. ACM SIGCOMM*, pages 319–330, 2014.

[14] H. Ballani, P. Costa, R. Behrendt, D. Cletheroe, I. Haller, K. Jozwik, F. Karinou, S. Lange, K. Shi, B. Thomsen, et al. Sirius: A flat datacenter network with nanosecond optical switching. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, pages 782–797, 2020.

[15] T. Benson, A. Anand, A. Akella, and M. Zhang. Understanding data center traffic characteristics. *ACM SIGCOMM Computer Communication Review*, 40(1):92–99, 2010.

[16] D. Bilò, L. Gualà, and G. Proietti. Improved approximability and non-approximability results for graph diameter decreasing problems. *Theoretical Computer Science*, 417:12–22, 2012.

[17] B. Birnbaum and C. Mathieu. On-line bipartite matching made simple. *SIGACT News*, 39(1):80–87, 2008.

[18] A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.

[19] N. Buchbinder, K. Jain, and J. Naor. Online primal-dual algorithms for maximizing ad-auctions revenue. In *Proc. 15th European Symp. on Algorithms (ESA)*, pages 253–264, 2007.

[20] S.-T. Chuang, A. Goel, N. McKeown, and B. Prabhakar. Matching output queueing with a combined input/output-queued switch. *IEEE Journal on Selected Areas in Communications*, 17(6):1030–1039, 1999.

[21] E. D. Demaine and M. Zadimoghaddam. Minimizing the diameter of a network using shortcut edges. In *Proc. Scandinavian Symposium and Workshops on Algorithm Theory (SWAT)*, pages 420–431, 2010.

[22] N. R. Devanur and K. Jain. Online matching with concave returns. In *Proc. 44th ACM Symp. on Theory of Computing (STOC)*, pages 137–144, 2012.

[23] N. R. Devanur, K. Jain, and R. D. Kleinberg. Randomized primal-dual analysis of RANKING for online bipartite matching. In *Proc. 24th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 101–107, 2013.

[24] M. Dinitz and B. Moseley. Scheduling for weighted flow and completion times in reconfigurable networks. *arXiv preprint arXiv:2001.07784*, 2020.

[25] L. Epstein, C. Imreh, A. Levin, and J. Nagy-György. Online file caching with rejection penalties. *Algorithmica*, 71(2):279–306, 2015.

[26] N. Farrington, G. Porter, S. Radhakrishnan, H. H. Bazzaz, V. Subramanya, Y. Fainman, G. Papen, and A. Vahdat. Helios: a hybrid electrical/optical switch architecture for modular data centers. In *Proc. ACM SIGCOMM*, pages 339–350, 2010.

[27] T. Fenz, K.-T. Foerster, S. Schmid, and A. Villedieu. Efficient non-segregated routing for reconfigurable demand-aware networks. In *2019 IFIP Networking Conference (IFIP Networking)*, pages 1–9. IEEE, 2019.

[28] A. Fiat, R. M. Karp, M. Luby, L. A. McGeoch, D. D. Sleator, and N. E. Young. Competitive paging algorithms. *Journal of Algorithms*, 12(4):685–699, 1991.

[29] K. Foerster, M. Ghobadi, and S. Schmid. Characterizing the algorithmic complexity of reconfigurable data center architectures. In *IEEE/ACM ANCS*, 2018.

[30] K.-T. Foerster, M. Pacut, and S. Schmid. On the complexity of non-segregated routing in reconfigurable data center architectures. In *ACM SIGCOMM Computer Communication Review (CCR)*, 2019.

[31] K.-T. Foerster and S. Schmid. Survey of reconfigurable data center networks: Enablers, algorithms, complexity. In *SIGACT News*, 2019.

[32] M. Ghobadi, R. Mahajan, A. Phanishayee, N. R. Devanur, J. Kulkarni, G. Ranade, P. Blanche, H. Rastegarfar, M. Glick, and D. C. Kilper. Projector: Agile reconfigurable data center interconnect. In *Proc. ACM SIGCOMM*, pages 216–229, 2016.

[33] A. Gozzard, M. Ward, and A. Datta. Converting a network into a small-world network: Fast algorithms for minimizing average path length through link addition. *Information Sciences*, 422:282–289, 2018.

[34] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu. Bcube: a high performance, server-centric network architecture for modular data centers. In *Proc. ACM SIGCOMM*, pages 63–74, 2009.

[35] S. Huq and S. Ghosh. Locally self-adjusting skip graphs. *Proc. IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pages 805–815, 2017.

[36] S. Jia, X. Jin, G. Ghasemiesfeh, J. Ding, and J. Gao. Competitive analysis for online scheduling in software-defined optical WAN. In *INFOCOM*, pages 1–9, 2017.

[37] X. Jin, Y. Li, D. Wei, S. Li, J. Gao, L. Xu, G. Li,

W. Xu, and J. Rexford. Optimizing bulk transfers with software-defined optical WAN. In *Proc. ACM SIGCOMM*, pages 87–100, 2016.

[38] C. Kachris and I. Tomkos. A survey on optical interconnects for data centers. *IEEE Communications Surveys and Tutorials*, 14(4):1021–1036, 2012.

[39] B. Kalyanasundaram and K. R. Pruhs. An optimal deterministic algorithm for online b-matching. *Theoretical Computer Science*, 233(1-2):319–325, 2000.

[40] S. Kandula, J. Padhye, and P. Bahl. Flyways to de-congest data center networks. In *HotNets*, 2009.

[41] R. M. Karp, U. V. Vazirani, and V. V. Vazirani. An optimal algorithm for on-line bipartite matching. In *Proc. 22nd ACM Symp. on Theory of Computing (STOC)*, pages 352–358, 1990.

[42] S. Kassing, A. Valadarsky, G. Shahaf, M. Schapira, and A. Singla. Beyond fat-trees without antennae, mirrors, and disco-balls. In *Proc. ACM SIGCOMM*, pages 281–294, 2017.

[43] H. Liu, F. Lu, A. Forencich, R. Kapoor, M. Tewari, G. M. Voelker, G. Papen, A. C. Snoeren, and G. Porter. Circuit switching under the radar with reactor. In *NSDI*, pages 1–15, 2014.

[44] V. Liu, D. Halperin, A. Krishnamurthy, and T. E. Anderson. F10: A fault-tolerant engineered network. In *NSDI*, pages 399–412, 2013.

[45] M. Mahdian and Q. Yan. Online bipartite matching with random arrivals: an approach based on strongly factor-revealing LPs. In *Proc. 43rd ACM Symp. on Theory of Computing (STOC)*, pages 597–606, 2011.

[46] L. A. McGeoch and D. D. Sleator. A strongly competitive randomized paging algorithm. *Algorithmica*, 6(6):816–825, 1991.

[47] N. McKeown. The islip scheduling algorithm for input-queued switches. *IEEE/ACM transactions on networking*, 7(2):188–201, 1999.

[48] A. Mehta. Online matching and ad allocation. *Foundations and Trends in Theoretical Computer Science*, 8(4):265–368, 2013.

[49] A. Mehta, A. Saberi, U. V. Vazirani, and V. V. Vazirani. Adwords and generalized online matching. *Journal of the ACM*, 54(5), 2007.

[50] W. M. Mellette, R. Das, Y. Guo, R. McGuinness, A. C. Snoeren, and G. Porter. Expanding across time to deliver bandwidth efficiency and low latency. *arXiv preprint arXiv:1903.12307*, 2019.

[51] W. M. Mellette, R. McGuinness, A. Roy, A. Forencich, G. Papen, A. C. Snoeren, and G. Porter. Rotornet: A scalable, low-complexity, optical datacenter network. In *Proc. ACM SIGCOMM*, pages 267–280, 2017.

[52] A. Meyerson and B. Tagiku. Minimizing average shortest path distances via shortcut edge addition. In *Proc. Int. Workshop on Approximation Algorithms for Combinatorial Optimization (APPROX)*, pages 272–285. 2009.

[53] J. Naor and D. Wajc. Near-optimum online ad allocation for targeted advertising. In *Proc. 16th ACM Conf. on Economics and Computation (EC)*, pages 131–148, 2015.

[54] M. Papagelis, F. Bonchi, and A. Gionis. Suggesting ghost edges for a smaller world. In *Proc. 20th ACM International Conference on Information and Knowledge Management (CIKM)*, pages 2305–2308, 2011.

[55] N. Parotsidis, E. Pitoura, and P. Tsaparas. Selecting shortcuts for a smaller world. In *Proc. SIAM International Conference on Data Mining*, pages 28–36, 2015.

[56] S. Ratnasamy, M. Handley, R. M. Karp, and S. Shenker. Topologically-aware overlay construction and server selection. In *INFOCOM*, pages 1190–1199, 2002.

[57] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren. Inside the social network's (datacenter) network. In *ACM SIGCOMM Computer Communication Review*, volume 45, pages 123–137, 2015.

[58] C. Scheideler and S. Schmid. A distributed and oblivious heap. In *ICALP (2)*, volume 5556 of *Lecture Notes in Computer Science*, pages 571–582, 2009.

[59] S. Schmid, C. Avin, C. Scheideler, M. Borokhovich, B. Haeupler, and Z. Lotker. Splaynet: Towards locally self-adjusting networks. *IEEE/ACM Trans. Netw.*, 24(3):1421–1433, 2016.

[60] A. Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*. Algorithms and combinatorics. Springer, 2003.

[61] A. Singla, C. Hong, L. Popa, and P. B. Godfrey. Jellyfish: Networking data centers randomly. In *NSDI*, pages 225–238, 2012.

[62] D. D. Sleator and R. E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.

[63] M. Y. Teh, Z. Wu, and K. Bergman. Flexspander: augmenting expander networks in high-performance systems with optical bandwidth steering. *IEEE/OSA Journal of Optical Communications and Networking*, 12(4):B44–B54, 2020.

[64] S. B. Venkatakrishnan, M. Alizadeh, and P. Viswanath. Costly circuits, submodular schedules and approximate carathéodory theorems. In *SIGMETRICS*, pages 75–88, 2016.

[65] M. Wang, Y. Cui, S. Xiao, X. Wang, D. Yang, K. Chen, and J. Zhu. Neural network meets dcn: Traffic-driven topology adaptation with deep learning. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 2(2):1–25, 2018.

[66] H. Wu, G. Lu, D. Li, C. Guo, and Y. Zhang. Mdcube: a high performance network structure for modular data center interconnection. In *CoNEXT*, pages 25–36, 2009.

[67] W. Xia, P. Zhao, Y. Wen, and H. Xie. A survey on data center networking (DCN): infrastructure and operations. *IEEE Communications Surveys and Tutorials*, 19(1):640–656, 2017.

[68] X. Zhou, Z. Zhang, Y. Zhu, Y. Li, S. Kumar, A. Vahdat, B. Y. Zhao, and H. Zheng. Mirror mirror on the ceiling: Flexible wireless links for data centers. *ACM SIGCOMM Computer Communication Review*, 42(4):443–454, 2012.