DistroMatch: Distributed Disjoint Weighted Matchings in Demand-Aware Reconfigurable Optical Datacenters

Kathrin Hanauer
Faculty of Computer Science
University of Vienna
Vienna, Austria
Research Network Data Science
University of Vienna
Vienna, Austria
kathrin.hanauer@univie.ac.at

Sophia Heck
Faculty of Computer Science
University of Vienna
Vienna, Austria
Doctoral School Computer Science
University of Vienna
Vienna, Austria
sophia.heck@univie.ac.at

Stefan Schmid
TU Berlin and Fraunhofer SIT
Berlin, Germany
schmiste@gmail.com

Abstract

Reconfigurable optical circuit switches revolutionize datacenter networks by allowing to adjust the physical topology in a dynamic and demand-aware manner. These switches directly match currently frequently communicating racks, reducing bandwidth tax and hence improving throughput. The underlying optimization problem is essentially the NP-hard Weighted k-Disjoint Matchings problem. Existing efficient solutions to this problem require a centralized controller, which constitutes a scalability bottleneck. This paper introduces the first distributed approach. Our main contributions are four new algorithms and a new approach to trade solution quality for running time via a parameter $\varepsilon \in [0, 1]$. Our best algorithm guarantees a $\frac{1}{3}$-approximation. We provide an extensive empirical evaluation on 87 real-world and synthetic workloads with billions of edges showing scalability and a speedup over state-of-art algorithms up to 1-2 orders of magnitude on most instances while retaining high-quality solutions.

CCS Concepts

• **Networks** → **Network architectures**; **Data center networks**; • **Theory of computation** → **Distributed algorithms**; **Approximation algorithms analysis**; **Graph algorithms analysis**.

Keywords

Datacenter Networks, Performance, Optimization, Distributed Control, Optical Interconnects

ACM Reference Format:
Kathrin Hanauer, Sophia Heck, and Stefan Schmid. 2026. DistroMatch: Distributed Disjoint Weighted Matchings in Demand-Aware Reconfigurable Optical Datacenters. In *2026 International Conference on Supercomputing (ICS '26), July 06–09, 2026, Belfast, United Kingdom.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3797905.3800555

1 Introduction

Datacenters are experiencing a massive growth in communication traffic. This is also due to the data-intensive applications related to AI/ML [59], the trend toward resource disaggregation and heterogeneous computing, as well as the storage on high I/O speed media [43]. Accordingly, great efforts are being made to improve datacenter network performance [40].

Optical switching technologies provide a particularly interesting approach to improve datacenter network performance. Emerging *optical circuit switches* (OCSs) not only avoid overheads related to the optical-electrical signal conversion, which greatly increases bandwidth capacity, they also support fast reconfigurations of the physical network topology to optimize for the current communication demand [13, 26]. The latter allows to reduce route lengths and hence bandwidth tax, which in turn improves throughput [48]. A prominent example of such a demand-aware reconfigurable optical network is Google's evolving Jupiter datacenter [54].

While optical switching technologies, such as reconfigurable micromirrors and tunable lasers, allow to change the physical interconnect faster and faster, within microseconds [26] or even nanoseconds [5], the control plane required to perform demand-aware optimizations is becoming a bottleneck. Specifically, each OCS provides direct connections between endpoints in form of a *matching*, which pairs entities, e.g. top-of-rack (ToRs) switches, that communicate intensively. By offloading traffic from the traditional electrical network to the optical one, network bandwidth performance can be improved significantly [28]. Computing an optimal configuration which offloads a maximum amount of traffic to an optical network consisting of k OCSs requires to find k pairwisely edge-disjoint matchings of maximum total weight. This problem is known to be $\mathcal{NP}$-hard [23, 35]. While several heuristics and approximation algorithms have been developed to solve this problem more efficiently in the offline [35], streaming [25] and dynamic [33] setting,

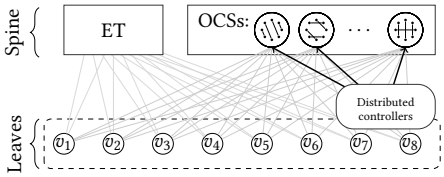

Figure 1: Distributed computation of the OCS configurations. Each configuration corresponds to a weighted matching that is edge-disjoint from all others. ET represents an electrical topology over which the remaining demand can be routed.

This work is licensed under a Creative Commons Attribution 4.0 International License.
ICS '26, Belfast, United Kingdom
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2522-7/26/07
https://doi.org/10.1145/3797905.3800555

these approaches require a centralized control architecture, which limits scalability.

Modeling the Problem. In this paper, we study the problem of computing k pairwisely edge-disjoint matchings in a distributed setting. See Figure 1 for a visualization of our abstract model. Whereas most prior work on reconfigurable network architectures assumes a centralized controller, there is broad consensus that scalability will require decentralized designs [4].

We consider the typical scale-out architecture where optical switches connect racks or pods (see for example Google's Jupiter network); the leafs in Figure 1 hence represent electrical switches.

Our Contributions. We initiate the study of distributed (and hence scalable) control algorithms for reconfigurable datacenter networks. Towards this goal, we present several efficient algorithms which come with formal performance guarantees. Some of these algorithms are also of independent interest, as they improve upon existing offline algorithms.

Specifically, we present a 0.393-approximation algorithm which is based on the repeated computation of a matching, where we employ two different algorithms to obtain the matching in the distributed setting: local maximality [55] and suitor [44]. By using a known theorem for a more general problem, we can plug in any distributed algorithm for matching, leading to a $1-(1-\frac{\beta}{k})^k$ approximation, where β is the approximation ratio of the matching algorithm and k is the number of matchings (i.e, optical circuit switches).

Moreover, we extend two paradigms for computing (distributed) matchings, local maximality [55] and suitor [44], to obtain k disjoint matchings simultaneously. We also introduce a novel approach to trade solution quality for running time via a parameter ε, which can be combined with the local max principle. Our best algorithm, kCS, guarantees a $\frac{1}{3}$-approximation.

We perform extensive empirical evaluations using 87 real-world traffic matrices on a server-to-server and synthetic workloads, with billions of communication pairs, and find that kCS scales well and outperforms the state-of-the-art non-distributed algorithms on the majority of instances. On large instances, it achieves 1 to 2 orders of magnitude speedup while the solution weight remains between +0.1 % and −2.8 % on average. On traces and synthetic graphs, kCS achieves weights within 99.7 % of the best quality-achieving algorithm while using 32 controllers being 5 to 87 × faster. Compared to the previously fastest algorithm, it outruns it with 32 controllers on most instances by multiple factors and improves solution quality.

As a contribution to the research community, to ensure reproducibility and to facilitate follow-up work, we share the source code of our simulations and datasets together with this paper[1].

2 Preliminaries

In Figure 2, we show our model: Multiple distributed controllers observe the demand network and coordinate to compute k pairwise edge-disjoint matchings, which are deployed as configurations on the optical switches. The remaining demand is routed over the electrical network.

We model the demand as an undirected graph $G = (V, E, D)$ with a set of vertices V representing servers, pods, or top-of-rack switches in the network and a set of edges E that represent non-zero demands

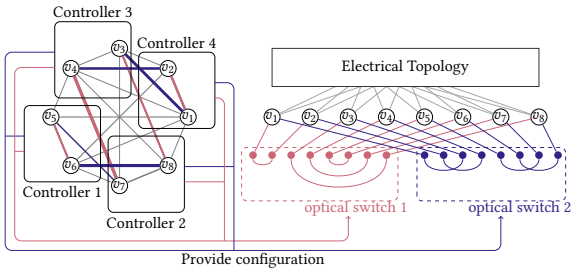

Figure 2: Distributed computation of the OCS configurations. The demand network (left) is observed by multiple controllers which configure the OCSs (right). Edge weights in the demand network are shown by thickness.

between these vertices. Each edge $e \in E$ is assigned a demand $D(e) \in \mathbb{R}^+$, which represents the traffic or weight associated with the edge. For a subset of edges $S \subseteq E$, we write $D(S) = \sum_{e \in S} D(e)$. Note that the edges in G represent communication demand and not the physical links in the data center.

We call two vertices $u, v \in V$ *adjacent* to each other if $\{u, v\} \in E$. Then, $\{u, v\}$ is also said to be *incident* to u and v. Similarly, two edges are *adjacent* if they share a common end vertex. The *neighborhood of an edge* $e = \{u, v\}$ is $N(e) = \{e' \in E : |e \cap e'| = 1\}$, i.e., the set of all edges adjacent to it. The degree $\deg(v)$ of a vertex v is its number of incident edges, and Δ is the maximum vertex degree. A *matching* $M \subseteq E$ is a set of edges such that no vertex is incident to two edges contained in M. A matching is *maximal* if no further edge can be added to it.

Weighted k-Disjoint Matchings Problem. The OCSs provide direct routing options in form of one *matching* per switch. Since routing over electrical networks can incur a bandwidth tax and require multiple hops, we aim to maximize throughput by routing as much as possible over the optical switches. Therefore, we aim to compute a set of matchings whose edges have high demand. Capacity constraints can be modeled by using multiple parallel edges for demands exceeding capacity in the demand graph [33].

The task of computing a throughput-optimized configuration for k optical switches can be formulated as the *Weighted k-Disjoint Matchings Problem* on the demand graph. We follow the formal definition provided by Hanauer et al. [33]:

Definition 1 (Weighted k-Disjoint Matchings Problem (k-DM))**.** Given an edge-weighted, undirected graph $G = (V, E, D)$ and an integer $k \in \mathbb{N}$, find k pairwisely edge-disjoint matchings $\mathcal{M} = (M_1, \ldots, M_k)$ such that the demand $D(\mathcal{M}) := \sum_{1 \le i \le k} D(M_i)$, with $D(M_i) = \sum_{e \in M_i} D(e)$, is maximized. We also refer to $D(\mathcal{M})$ as solution weight.

This problem can equivalently be formulated as *Weighted k-Edge Coloring Problem* (k-EC). Let $C : E \to [k] \cup \{\perp\}$ be a partial coloring function[2], where $C(e) = \perp$ indicates that edge e remains uncolored. It aims to find a partial edge coloring C such that no two adjacent edges share the same color and maximize the total demand of colored edges. Two of the introduced algorithms are

[1] https://github.com/DJ-Match/DistroMatch [32]

[2] $[k] := \{1, \ldots, k\}$

based on this view. The equivalence between k-DM and k-EC is as follows: In k-EC, each set of same colored edges forms a matching since no two adjacent edges share the same color. Conversely, in k-DM, assigning color i to all edges in matching i yields a valid edge coloring.

Distributed control. With the quickly increasing scale of datacenters, the control plane is becoming a bottleneck and more decentralized solutions are critical.
In such a distributed control setting, the graph $G = (V, E, D)$ is spread across multiple controllers. Each controller p handles a subset of vertices $\mathcal{V}_p \subseteq V$ (i.e., top-of-rack switches) and their incident edges $\mathcal{E}_p \subseteq E$, enabling parallel computation and efficient resource utilization. We denote the number of controllers in the system by n_p. Note that memory is not shared between the controllers. This results in a local view of the graph for every controller p.
An edge $\{u, v\} \in \mathcal{E}_p$ is said to be *local* if both $u, v \in \mathcal{V}_p$ and *cross* otherwise. A vertex $v \notin \mathcal{V}_p$ is a *ghost vertex* for controller p if there is a vertex $u \in \mathcal{V}_p$ with a cross edge $\{u, v\} \in \mathcal{E}_p$. We denote by $\mathcal{V}'_p \supseteq \mathcal{V}_p$ the set of the local and ghost vertices of the controller p.
All running times stated in the following refer to the total *work* done in sum by all controllers.

3 Distributed Control

In this section, we describe and analyze four new distributed algorithms for k-DM: `MRepLM`, `MRepS`, `kCS`, and `kMM`. Our algorithms are based on established concepts for computing traditional weighted matchings (1-DM) in a distributed setting. For k-DM, we follow different approaches to adapt this concept: Our first two algorithms compute k matchings successively, while the other two adopt the k-EC approach to obtain all k disjoint matchings simultaneously.

3.1 Review: Distributed/Parallel Matching

We briefly review two widely used paradigms for parallel/distributed matching, i.e., 1-DM, which serve as a base for the algorithms described later. Both use asynchronous communication overlapping sending and receiving with latency.

Local maximality. The distributed `LocalMax` algorithm [11, 37] is an edge-greedy method based on the principle of *local maximality*. An edge $e = \{u, v\}$ is *locally maximal* if its weight is maximum among all incident edges at both endpoints. A vertex v is called *candidate* of u, denoted by $C(u)$, iff $D(\{u, v\}) \geq D(\{u, x\})\ \forall x \in N(u)$ and *locally maximal* iff $C(u) = C(v)$. The algorithm proceeds in rounds. Each controller performs the following steps: Local vertices u select their candidate $C(u)$, where $C(u)$ may point to a local or a ghost neighbor. Then, matches where both endpoints are local are resolved: We add $\{u, v\}$ to the matching iff $C(u) = v$ and $C(v) = u$; in this case, u and v are marked as matched. For remaining ghost candidates, the controller sends a proposal to the owner of that neighbor and receives proposals from other controllers. Incoming proposals are checked for mutual selection (i.e., whether $C(u) = C(v)$) and upon success, the corresponding edge is added to the matching and their endpoints are marked as matched. Matched (local and ghost) vertices and all their incident edges are then removed. Vertices not incident to a selected edge remain unmatched and participate in the next round. A consistent global tie-breaking rule is assumed to ensure determinism across controllers.
`LocalMax` terminates in at most m synchronous rounds, with each round completing a linear pass over all edges in $O(m)$ time [11]. The resulting matching provides a $\frac{1}{2}$-approximation of the maximum-weight matching [37].

Suitor-based algorithms. We adopt the vertex-centric `Suitor` paradigm into the distributed-memory setting, following the sorted-adjacency variant `SortSuitor` for the shared-memory setting introduced by Manne and Halappanavar [44], which often outperforms edge-centric methods in practice [44]. The distributed `Suitor` approach has also been used for the more general b-matching[3] problem [41]. The central idea is that vertices *propose* to their current best potential match, i.e., initially a neighbor to which they have a heaviest edge. The algorithm computes a *stable matching*, i.e., every pair of matched vertices cannot find a better matching partner from their local perspective.
As long as an unmatched local vertex has a neighbor it has not yet proposed to, the respective controller performs the following steps: Each local vertex u proposes to its most preferred neighbor v to which it has not yet proposed, making u a *suitor* of v. Neighbors are processed in non-increasing order of demand, breaking ties deterministically. Each vertex u issues at most one proposal per round. For each proposal from u to v, first check whether v is local. If v is local, the controller compares u with its own proposal and accepts u if it matches; upon acceptance, mark u and v as matched, otherwise reject u. If v is a ghost, prepare a proposal message to v's owner. The controller sends proposals and receives proposals of ghost vertices from other controllers to local vertices. For a local vertex u and a ghost vertex v, the controller applies the same comparison: if u is a suitor of v, accept and mark u and v as matched; otherwise reject the proposal (by symmetry no acknowledgment to the sender is required). Rejected vertices advance to their next preferred neighbor and continue proposing in subsequent rounds. When no proposals remain, the set of held pairs forms the matching.
`SortSuitor` computes a matching in $O(n + m \log \Delta)$ and obtains a $\frac{1}{2}$-approximate maximum-weight matching [44].

3.2 `MRep`: Repeated Distributed Algorithms

A rather straightforward approach that extends distributed matching to $k > 1$ is to compute a single matching, remove the matched edges from the graph, and repeat the process on the remaining graph $k-1$ times. We call this algorithm `MatchingRepeated` (`MRep`).
For the analysis of the approximation ratio of `MRep`, we use the observation that k-DM is a special case of the (weighted) Maximum Coverage problem, as demonstrated by Feige et al. [23]. Consequently, Theorem 1 below directly follows from a proof about Maximum Coverage by Hochbaum [36, Theorem 3.9].

Theorem 1. A solution to k-Disjoint Matchings computed by k times computing a maximal matching with approximation ratio β is a $1 - (1 - \frac{\beta}{k})^k > 1 - \frac{1}{e^\beta}$ approximation of the optimal solution.

[3] A set of edges is a b-matching if every vertex is incident to at most b edges. This problem is polynomial-time solvable and fundamentally different from k-DM, see, e.g., [19].

Algorithm 1: `kColoringSuitor` (kCS)

Input: Local vertices $\mathcal{V}$ and their edges $\mathcal{E}$
$\forall e \in E : C(e) \leftarrow \perp$; $Q \leftarrow \mathcal{V}$; $\forall v \in \mathcal{V}$: Sort $\mathcal{E}[v]$;
while $Q.size > 0$ **do**
  `// Proposal phase`
  Determine proposals P to make for all $v \in Q$;
  Remove done vertices from Q;
  Prepare messages for done vertices;
  `// Proposal processing phase`
  forall $v \in Q$ **do**
    $u \leftarrow P[v]$;
    if *u is local and* $P[u] = v$ **then**
      Color u and v with a common free color if possible;
      Mark u and v to move to new partners;
    else
      Prepare proposal w. $\mathcal{F}(v)$ for q s.t. $u \in \mathcal{V}_q$
  `// Communication phase`
  Send out messages to active controllers;
  Receive messages from active controllers;
  forall *Received messages msg* **do**
    if *msg is a proposal of u to v and* $P[v] = u$ **then**
      Color u and v with a common free color if possible;
      Mark v to move to new partner;
    else if *msg is vertex or controller done message* **then**
      Store this information ;
Prepare and send done messages

Note that with an *optimal* matching subroutine in `MRep`, we would obtain a ≈ 0.63 approximation [36] for k-DM.
We consider two variants of this concept with `SortSuitor` and `LocalMax` as distributed matching algorithms. In the following, we refer to these algorithms as `MatchingRepeatedSuitor` (`MRepS`) and `MatchingRepeatedLocalMax` (`MRepLM`), respectively. As both guarantee an approximation ratio of $\frac{1}{2}$ for weighted matching [37, 44], we have:

Corollary 1. `MRepS` and `MRepLM` are approximation algorithms for k-DM with ratio $1 - (1 - \frac{0.5}{k})^k > 1 - \frac{1}{e^{0.5}} \approx 0.393$.

The running time of `MRepS` and `MRepLM` is determined by the number of matchings k and the time to obtain one matching. Recall that the time to compute a matching is $O(n + m \log \Delta)$ for `SortSuitor` [44] and $O(m^2)$ for `LocalMax` [11].

Lemma 1. `MRepLM` *and* `MRepS` *run in worst-case time* $O(km^2)$ *and* $O(k(n + m \log \Delta))$*, respectively.*

3.3 kCS: Distributed Coloring Suitor

In this section, we present an extension of the `SortSuitor` [44] algorithm for k-DM. We refer to this algorithm as `kColoredSuitors` (`kCS`) and provide the pseudocode in Algorithm 1.
Recall that `SortSuitor` is vertex-based, where unmatched vertices propose to their heaviest neighbors. A receiving vertex accepts the best proposal and rejects others, forcing rejected vertices to propose again. `kCS` aims to find k matchings simultaneously, where we represent each matching with a color. The main idea is that each vertex makes one proposal at a time while it still has an available color and there is a neighbor left to propose to. The neighbors are asked in order of their edge's weight. Note that a proposal as such does not contain a color, which is only chosen once a matching proposal is found.
The algorithm maintains a queue Q of all *active* local vertices. An *active* vertex has still unassigned colors and has not yet proposed to all its neighbors. For each local vertex v, the algorithm maintains a set of available colors $\mathcal{F}(v)$ as well as the *active degree*, i.e.the number of active neighbors. Additionally, it maintains an `active` flag for each ghost vertex which indicates whether it still has free colors left. The algorithm operates in rounds, where each round refers to one iteration over all active vertices Q. First, every active vertex submits a proposal. Next, all proposals are processed. Finally, in the communication phase, all outgoing messages are sent and all incoming messages are received and handled. We now describe each phase of the round in detail.

Proposal phase. Each active vertex proposes to its heaviest active neighbor u that it has not yet proposed to. Ties are broken by choosing the neighbor with the smaller ID. We refer to the proposal of vertex v as $P[v]$ and call v a *suitor* of u. If a vertex cannot make a proposal, it is no longer active and removed from Q. In this case, an inactive vertex message is prepared for all affected controllers, which will be sent in the communication phase. With this further proposals to vertices without free colors are avoided.

Proposal processing phase. `kCS` tries to find matching proposals by checking all vertices. We say the proposals match if two vertices u, v made proposals to each other, i.e., $P[v] = u$ and $P[u] = v$. Such proposals are processed by u and v by choosing a common free color $\mathcal{F}(v) \cap \mathcal{F}(u)$ if possible. Otherwise, the edge remains uncolored. In case there are multiple common colors, we use a fixed selection scheme to choose one. The processing concludes with marking u and v to make proposals to their next heaviest neighbor. Matching proposals between local vertices can be directly identified and processed. In case of a proposal to a ghost vertex, the current controller cannot know whether the proposals match. Therefore, it prepares a proposal message to the controller of the proposed vertex that contains the suitor, the proposed vertex, and the available colors of the suitor. If this proposal matches, the controller of the ghost also sends a proposal message in the same round, so both controllers can identify and process the match in the communication step.

Communication phase. The controller sends all prepared messages to the appropriate other controllers which still have active vertices and receives their answers. This is done asynchronously such that sending and receiving can overlap and received messages can directly be processed. There are three message types: a proposal, a flag that a vertex is inactive, and a flag that a controller has finished. For proposals from a ghost, a controller can now check whether it matches the proposal from the local vertex. If so, it processes them as described above while using the received colors for ghost vertex and locally maintained ones for the local vertex. For the other two message types, it simply stores the received information.
Once a controller has no active vertices left, it sends a finish message to all other controllers to avoid idle messages to a finished process.

Each vertex can only color one incident edge per round. Handling multiple proposals in a single round would raise a challenge in the distributed setting, as the two controllers would need to agree on the same colors without introducing conflicts. The number of messages per round for kCS is in $O(n_p^2)$, as each controller can communicate proposals or statuses to every other controller.

Hanauer et al. [34, Lemma 6] proved that a $\frac{1}{3}$-approximation algorithm for k-DM can be achieved if, for every uncolored edge e and every color $c \in [k]$, there exist one or two adjacent edges already colored with c whose combined weight is at least as large as the weight of e. This holds for kCS, as the edges of a vertex are considered in weight-decreasing order and hence, an edge will only remain uncolored if all colors were used by at least one heavier adjacent edge before.

Corollary 2. kCS is a $\frac{1}{3}$-approximation algorithm for k-DM.

To analyze the running time, we observe that sorting the neighbor lists once in the beginning takes $O(n\Delta \log \Delta) = O(m \log \Delta)$ time. In each round, kCS iterates over all active vertices, which takes $O(n)$ time. The number of rounds is again at most m (and at least k). This dominates the first step.

Lemma 2. kCS *runs in worst-case time* $O(nm)$.

3.4 kMM: Distributed Local (ε-)Maximality

For our next algorithm, kMM, we introduce an extension of the LocalMax principle, so that k disjoint matchings are handled simultaneously. We take the k-EC view here, where the edges of the i-th matching M_i receive color i.

The primary concept underlying this approach is the intention to color multiple edges within a single iteration. To avoid explicitly keeping track of multiple candidates, we define a *candidate weight* W_c for each vertex to be the weight of its heaviest incident uncolored edge. We slightly revise the definition of a *candidate* and say that $e = \{u, v\}$ is a candidate iff $C(e) = \perp$, $D(e) = W_c(u)$, and $D(e) = W_c(v)$. This deviates from the definition in the plain matching algorithm, where a candidate is one-sided. Note that there may be more than one candidate edge for a vertex if they have the same weight; in this case, more than one of its candidate edges may be colored in the same iteration.

To speed up the algorithm and have even more candidates in the same round, we consider a relaxation of the candidate selection process. Instead of using only the heaviest edges of a vertex as candidates, we also consider further "heavy" edges. Formally, we introduce a parameter ε and extend the candidate definition such that an edge $e = \{u, v\}$ is a candidate iff $D(e) \geq (1 - \varepsilon)W_c(u)$ and $D(e) \geq (1 - \varepsilon)W_c(v)$. We call this relaxation *local ε-maximality*.

Invariant 1. *Let e, e' be adjacent edges. With local ε-maximality, e can only be colored before e' if $D(e) \geq (1 - \varepsilon)D(e')$.*

The invariant naturally holds as e would not be a candidate in the same round when an edge e' with demand more than $\frac{1}{(1-\varepsilon)}D(e)$ is still in the graph. Note that local maximality is the special case with $\varepsilon = 0$. In the following, we first outline the main structure sequentially and then discuss the communication.

Similar to LocalMax for 1-DM, our algorithm operates in rounds. For every vertex $v \in V$, it maintains a set of *free colors* $\mathcal{F}(v)$, which are colors not assigned to any of its incident edges. Every round starts by initializing the candidate weight $W_c(v)$ for each vertex $v \in V$ based on the weights of its uncolored incident edges. Next, for every candidate edge $e = \{u, v\}$, we check whether $\mathcal{F}(u) \cap \mathcal{F}(v) \neq \emptyset$. If positive, we color e with a random color $i \in \mathcal{F}(u) \cap \mathcal{F}(v)$ and leave it uncolored otherwise. To enhance the efficiency of the algorithm, we remove an edge from the graph if it is colored or if its end vertices have no common free color. Here, we examine candidate edges before non-candidate edges to maximize the number of edges removed in each round, as each newly colored edge reduces the number of free colors at its adjacent edges.

The central idea to distribute this approach is to make local decisions and handle conflicts if necessary. We call this algorithm kMatchMediator (kMM) and outline it in Algorithm 2.

For edges with endpoints in different controllers, we arbitrarily choose a responsible controller to avoid duplicated edges. Cross edges and local edges are then treated alike.

At the start of each round, *setDistributedCandidateWeights()* computes the local candidate weights followed by a synchronous communication step to ensure all controllers have the correct candidate weights. This can be implemented efficiently with MPI Allgather. Otherwise, the candidate weight W_c would be a local view, such that a controller p might have a ghost vertex v with $W_c^p(v)$, but the controller q of v has a $W_c^q(v) > W_c^p(v)$.

Each controller p maintains a local set of free colors $\mathcal{F}_p(v)$ for every vertex $v \in \mathcal{V}'_p$, i.e., for every local and ghost vertex. Note that $\mathcal{F}_p(v) \supseteq \mathcal{F}(v)$ and $\bigcap_p \mathcal{F}_p(v) = \mathcal{F}(v)$. During the candidate iteration phase, the controllers need to communicate to maintain consistency and resolve conflicts. As communication is synchronized, a main challenge for efficient scaling is to perform similar local work between communication steps.

Each controller p performs the following steps for each candidate edge $\{u, v\}$: It assigns a random preliminary color c based on the local information $\mathcal{F}_p(u) \cap \mathcal{F}_p(v)$, then collects the decisions from

Algorithm 2: kMatchMediator (kMM)

```
Input: Sets V and E of local vertices and their edges
∀e ∈ E : C(e) ← ⊥;
while ∃e ∈ E : c(e) = ⊥ do
    setDistributedCandidateWeights();
    forall candidates e = {u, v} ∈ E do
        if u and v have a common free color then
            Select color at random;
            Set candidate;
            Receive p − 1 other candidates;
            Handle candidates and conflicts;
            if e has no conflict then
                color e with free color;
                E = E \ {e};
        if u and v have no common free color then
            E = E \ {e};
    forall non-candidates e = {u, v} ∈ E do
        if u and v have no common free color then
            E = E \ {e};
```

the $n_p - 1$ other controllers in a synchronous communication step decision and resolves conflicts if necessary. If there was no conflict, $\{u, v\}$ is colored. The conflict detection and handling works as follows. Let $\mathcal{R}$ be a list of all n_p local coloring decisions, i.e., for each controller p, it contains an entry with p's current edge and preliminary color. The order of $\mathcal{R}$ is arbitrary, but consistent across all controllers. Conflicts are handled by processing the information in $\mathcal{R}$ one by one and updating the local information $\mathcal{F}_p(v)$ for every $v \in \mathcal{V}'_p$ if there is no conflict. This ensures that every controller has a global view of the candidate edges and can make informed decisions for subsequent edges. The algorithm maintains the following invariant during each step of the conflict resolution process.

Invariant 2. *Let $\mathcal{R}$ be the list of edge-color pairs containing the local decisions of all controllers in algorithm* kMM*. After each processed pair, every controller p has globally correct internal state of free colors $\mathcal{F}_p(v)$ for every $v \in \mathcal{V}'_p$.*

We omit the details due to space restrictions. The candidate communication step can be implemented efficiently with an MPI Allreduce routine. Furthermore, whereas total communication cost in MRepLM, MRepS and kCS scales with the number of cross edges, it instead scales with the number of considered candidates in kMM. This makes kMM efficient on graphs where relatively few edges are matched.
Similar to MRepLM, the running time of kMM depends on the number of rounds and the time for one round, but noticeably not on k. As each candidate is communicated to all controllers, the worst case running time of a round is based on the number of local edges plus the number of received candidates. Due to the conflict handling being done in every controller, this results in $O(m + m' n_p)$, where m' is the number of candidates during the execution, which is upper bounded by m. The maximum number of rounds is again m.

Lemma 3. kMM *runs in worst-case time $O(m^2 n_p)$.*

As kMM has global candidate weights, it meets Invariant 1. We prove that the approximation guarantee is $\frac{1-\varepsilon}{3-\varepsilon}$, following the proof in [34]. We will later see in our empirical evaluation that our worst-case analysis is conservative and that the algorithm performs even significantly better under real-world workloads.

Theorem 2. The weight of a k-disjoint matching M produced by local ε-maximality for k-DM is at least $\frac{1-\varepsilon}{3-\varepsilon}$ of the weight of an optimal k-disjoint matching.

Proof. Let $\mathcal{M}^\star = (M_1^\star, M_2^\star, \ldots, M_k^\star)$ be an optimal k disjoint matching, and let $\mathcal{M} = (M_1, M_2, \ldots, M_k)$ be a k-disjoint matching produced by local ε-maximality for k-DM. Let $C^\star$ and C be the colorings corresponding to $\mathcal{M}^\star$ and $\mathcal{M}$, respectively.
We categorize the edges based on their colorings in $C^\star$ and C, where $E_=$ contains edges with the same color in $C^\star$ and C, E_c those that are colored in $\mathcal{M}$, but not in $\mathcal{M}^\star$, E_u vice-versa those that are uncolored in $\mathcal{M}$, but colored in $\mathcal{M}^\star$, and $E_\neq$ those that received different colors.
Consider an edge $e = \{u, v\} \in E_u$, i.e., $C(e) = \perp$. This can only occur if $\mathcal{F}(u)$ and $\mathcal{F}(v)$ had no common free color. By Invariant 1, before e only adjacent edges with weight at least $(1-\varepsilon)D(e)$ could have been colored. Hence, there is an adjacent edge $e' \in N(e) \cap (E_c \cup E_\neq)$ with $C(e') = C^\star(e)$ and $D(e') \geq (1-\varepsilon)D(e)$. We charge e' the cost of $D(e) \leq \frac{1}{1-\varepsilon} D(e')$.
Vise versa, every edge $e' \in (E_c \cup E_\neq)$ is charged at most $\frac{2}{1-\varepsilon} D(e')$: There can be at most two edges in the neighborhood $e, \hat{e} \in N(e)$ with $C^\star(e) = C^\star(\hat{e}) = C(e')$. More precisely, $e, \hat{e} \in E_u \cup E_\neq$, and each of them can contribute at most $\frac{1}{1-\varepsilon} D(e')$. As every edge $e \in E_u$ is charged to an edge in $E_c \cup E_\neq$, we have $D(E_u) \leq 2\frac{1}{1-\varepsilon} D(E_c) + 2\frac{1}{1-\varepsilon} D(E_\neq)$. Hence,

$$\begin{aligned} D(\mathcal{M}^\star) &= D(E_=) + D(E_u) + D(E_\neq) \\ &\leq D(E_=) + \tfrac{2}{1-\varepsilon} D(E_c) + \tfrac{2}{1-\varepsilon} D(E_\neq) + D(E_\neq) \\ &\leq \tfrac{3-\varepsilon}{1-\varepsilon} D(\mathcal{M}). \end{aligned}$$

Note that if $E_u = \emptyset$, then $\mathcal{M}$ is optimal and $E_c = \emptyset$. Vice versa, if $E_c = \emptyset$ and $E_\neq \neq \emptyset$, it may be that $E_u \neq \emptyset$ because the different coloring hinders the coloring of additional edges. $E_=$ may or may not be empty. □

4 Empirical Evaluation

In this section, we complement our theoretical analysis with empirical evaluations to assess the performance of our algorithms under realistic workloads.

4.1 Methodology

We implemented[4] all algorithms in C++20 with MPI and compiled with GCC and the `-O3` flag. For the implementation of the `LocalMax` in MRep we used implementation provided by Birn et al. [11][5].
We used two test environments for our experiments. Machine A is an Ubuntu 24.04.1 LTS machine with Linux kernel version 6.8, a dual-socket AMD EPYC 9534 CPU with 128 physical cores running at max 3.72 GHz, and 1.48 TB of RAM. We let all MPI ranks communicate only via TCP. Machine B is part of the Austrian Scientific Cluster (ASC) and consists of AlmaLinux 8.8 nodes with kernel version 4.18, each of which is equipped with two AMD 7713 processors with 512 GB of RAM each. The nodes are connected by Mellanox HDR InfiniBand with 200 Gbit/s bandwidth. Here, each MPI rank used one node exclusively. Unless stated otherwise, all experiments were run on Machine A, which simulates a dedicated, fast communication structure between the controllers; this is a reasonable assumption, and existing literature often assumes such out-of-band control channels. We also report on scaling experiments on a cluster using Machine B.
We use non-blocking, asynchronous communication by MPI to hide network latency by overlapping it with computation for all algorithms except kMM. This algorithm requires synchronized phases to resolve inter-matching conflicts, limiting overlap.
In all cases, the input vertices were distributed randomly over the controllers using different random seeds. We also experimented with different vertex partitioning schemes, but with none we could more than compensate the additional computational overhead by a speedup in running time, see Section A. A similar observation was made by Khan et al. for b-matching [41].
We use the geometric mean to average results over different instances and/or values of k. To indicate that an algorithm `A` is run on n_p controllers, we refer to it as `A_`n_p, e.g., `kCS_32`.

[4] https://github.com/DJ-Match/DistroMatch [32]
[5] https://github.com/LocalMaxMatching/LocalMaxMatching

Table 1: Overview of all instances. For *rmat* instances, we aggregated averages over the six instances of same scale. min/max deg refer to the overall min/max degree.

graph	n	m	density	mean deg	max deg	min deg
fbA	457.0K	64.6M	1.0e-3	241.18	57 676	1
fbB	4.9M	164.3M	1.3e-5	66.20	376 689	1
fbC	990.0M	61.0M	1.24e-4	82.13	104 821	1
$rmat_{x14}$	16.4K	1.1M	8.1e-3	88.64	2827	0
$rmat_{x15}$	32.8K	2.2M	4.1e-3	90.34	4392	0
$rmat_{x16}$	65.5K	4.5M	2.1e-3	91.64	6676	0
$rmat_{x17}$	131.1K	9.1M	1.1e-3	92.65	9944	0
$rmat_{x18}$	262.1K	18.4M	5.3e-4	93.42	15 249	0
$rmat_{x19}$	524.3K	37.0M	2.7e-4	94.01	22 838	0
$rmat_{x20}$	1.0M	74.3M	1.4e-4	94.47	33 800	0
$rmat_{x21}$	2.1M	74.9M	3.4e-5	47.63	30 482	0
$rmat_{x22}$	4.2M	150.1M	1.7e-5	47.72	44 636	0
$rmat_{x23}$	8.4M	300.7M	8.7e-6	47.79	65 348	0
$rmat_{x24}$	16.8M	601.9M	4.0e-6	47.84	94 574	0
$rmat_{x26}$	67.1M	536.4M	2.4e-07	15.99	78 174	0
$rmat_{x27}$	134.2M	1.1B	1.2e-07	15.99	111 795	0
$rmat_{x28}$	268.4M	2.1B	6.0e-08	15.99	158 717	0
mycielskian20	786.4K	1.36B	4.40e-3	3446.42	393 217	19
com-Friendster	65.6M	1.81B	8.39e-7	55.06	5214	1
GAP-kron	134.2M	2.11B	2.34e-7	31.47	1 572 838	1
GAP-urand	134.2M	2.15B	2.38e-7	32.00	68	6
MOLIERE_2016	30.2M	3.34B	7.30e-6	220.81	2 106 904	1
AGATHA_2015	184.0M	5.79B	3.42e-7	62.99	12 642 631	1

4.2 Datasets

Following the methodology of prior work [25, 33, 35], we use a variety of datasets to ensure a robust evaluation of our algorithms. These datasets include real-world datacenter workloads and synthetic graphs. We provide an overview in Table 1. In addition to real-life traces, we also used synthetic data introduced in previous work [25, 33]. We are not aware of datacenter-specific traffic generators or simulators for general workloads.

Real-world Traffic Instances. We use real-world traffic data collected from Facebook data centers [56]. Attributes include timestamps, source/destination IPs, and packet lengths. We aggregated the packet lengths for each IP pair across all timestamps to reflect server-to-server traffic.

Synthetic Instances. We generated 75 synthetic instances using the R-MAT model [12]. To test scalability, we increased the number of vertices (and edges) to 2^x with $x \in \{14, \ldots, 24\}$, compared to $\{10, \ldots, 20\}$ in [25, 33, 35]. Following Hanauer et al. [33, 35], we use three initiator matrices for the R-MAT model: $rmat_a = (0.55, 0.15, 0.15, 0.15)$, $rmat_b = (0.45, 0.15, 0.15, 0.25)$, and $rmat_c = (0.25, 0.25, 0.25, 0.25)$. In addition, following Khan et al. [41], we also consider very big instances with $n = \{20^{26}, 20^{27}, 20^{28}\}$ vertices using the same initiators. We assigned edge weights in the range of one to half a million, drawn from uniform and exponential distributions. We refer to these synthetic instances as *rmat* graphs.

Large Real-world Instances. In addition to synthetic and real-world data traffic instances, we also incorporated large real-world instances to showcase performance. Following [25], we used the six largest available instances from the SuiteSparse Matrix Collection [16]. Since half of these instances are unweighted, we created two versions of each graph, assigning edge weights up to half a million drawn from uniform and exponential distributions.

Table 2: Comparison of the benchmark algorithms, approximation ratios, and time complexities.

Algorithm	Approximation	Time
`k-EC`	$\leq 1/2$	$O(\sqrt{m} + kn^2)$
`stk`	$\frac{1}{3+\varepsilon}$	$O(km)$
`stk-dp`	$\frac{1}{3+\varepsilon}$	$O(km)$
`kMM`	$\frac{1-\varepsilon}{3-\varepsilon}$	$O(n_p m^2)$
`MRepLM`	$1-(1-\frac{0.5}{k})^k > 0.393$	$O(k \cdot m^2)$
`MRepS`	$1-(1-\frac{0.5}{k})^k > 0.393$	$O(k(n + m \log \Delta))$
`kCS`	$1/3$	$O(nm)$

4.3 Benchmark Algorithms

We compare our algorithms against several benchmark algorithms. As this work introduces the first distributed algorithms for k-DM, no state of the art in this setting exists so far. To show the increased efficiency of our approach, we also compare with the state-of-the-art sequential algorithms. The current state-of-the-art offline algorithms are due to Hanauer et al. [35]. We consider their best-performing algorithm `k-EC`, which is based on the Misra-Gries edge coloring algorithm [50]. As recommended by the authors, we set the algorithm's CC and RL flags. `k-EC` processes edges in decreasing weight order. If both endpoints share a free color, the edge is colored immediately. Otherwise, the algorithm tries to free a color by recoloring other edges on a potentially very long path. Hence, `k-EC` takes a global view, which yields high solution quality.
Moreover, we compare with the streaming approach by Ferdous et al. [24]. Their `stk` algorithm is based on the primal-dual interpretation of the Paz and Schwartzman [53] algorithm. It maintains stacks for each color and uses approximate dual variables when considering edges as candidates for the solution, thus ensuring that only sufficiently heavy edges are added to the stacks. The k-disjoint matchings are computed in the post-processing phase. They also use a dynamic programming-based weight improvement method proposed by Manne and Halappanavar [44] to enhance the weight of a matching. This is achieved by computing $2k$ edge-disjoint matchings with `stk` and then merging these into k matchings using dynamic programming. They refer to this variant as `stk-dp`. All algorithms with approximation guarantee and running time are listed in Table 2.

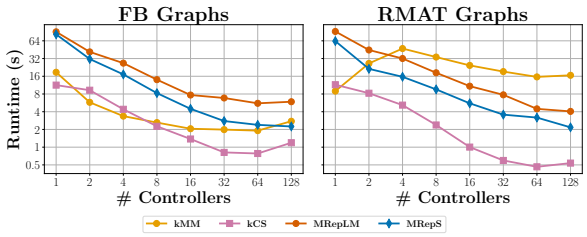

Figure 3: Strong scaling of all proposed algorithms for $k \in \{4, 8, 16, 32\}$ on fbA, fbC, and rmat instances with $n = 2^{20}$ vertices. Values were aggregated using the geometric mean.

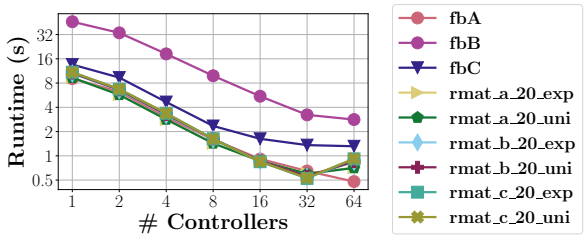

Figure 4: Strong scaling of kCS on machine B for $k \in \{4, 8, 16, 32\}$ on all facebook and rmat instances with $n = 2^{20}$ vertices. Values were aggregated using the geometric mean.

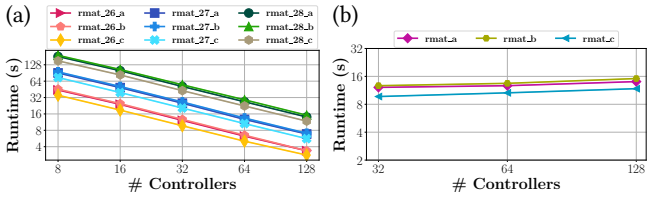

Figure 5: Strong (a) and weak (b) scaling of kCS on machine B for $k = 4$ and rmat instances with $n = \{2^{26}, 2^{27}, 2^{28}\}$ vertices. As different weight distributions performed similar, we group these for better readability.

4.4 Performance Analysis

4.4.1 ***Scaling of our Algorithms.*** Distributed algorithms are commonly evaluated in terms of scalability as the number of controllers increases, which is what we look at first.

For strong scaling, we run an algorithm repeatedly on the same instances, but double the number of controllers in each step. Here, we vary the number of controllers $n_p \in \{1, 2, 4, 8, 16, 32, 64, 128\}$ and consider different values of $k \in \{4, 8, 16, 32\}$. We focus on a subset of real-world and synthetic instances that contains all six rmat graphs with 2^{20} vertices and the facebook traces fbA and fbC. The experiment was run on machine A, and the results were aggregated over the different values of k using the geometric mean. The results for all our distributed algorithms are depicted in Figure 3.

For kMM, we observe very different scaling behavior for rmat and facebook graphs. The structure of the facebook graphs results in a small number of matched edges compared to the total number of edges. Hence, the number of communicated candidates is low, and most work can be done locally. On rmat graphs, we observe a runtime increase when introducing the distributed setting. Note that with more controllers the number of participants needed for synchronization and the message size increase, while the number of total communication steps goes down. Per communication step, each controller makes one local decision, hence the number of communication steps is roughly the number of candidates divided by n_p (plus the differences in number of candidates between the controllers), while each communication has more candidates. However, even with a larger number of controllers, it does not outrun sequential execution on these instances.

For the other three algorithms we observe a near-linear speedup on all instances as the amount of local work compared to communication depends on the locality of the graph, i.e., the ratio of local and cross edges. They also use asynchronous communication to overcome delays. When analyzing the running time as the number of controllers doubles, ideally the running time halves, corresponding to a scaling factor of 0.5. Doubling the number of controllers from initially 1 to $[2, 4, 8, 16, 32, 64, 128]$ in turn, we get the factors $[0.35, 0.68, 0.58, 0.57, 0.68, 0.88, 0.74]$ for MRepS, $[0.48, 0.70, 0.56, 0.58, 0.75, 0.63, 0.94]$ for MRepLM, and for kCS the factors are $[0.74, 0.59, 0.47, 0.46, 0.59, 0.82, 1.24]$. While kCS demonstrates better scaling for smaller numbers of controllers, it does not scale over 64 controllers in this experiment. As additional experiments on larger instances showed scaling also for more controllers, we assume that the results here are due to the (in comparison) limited size of the instances. Despite this, kCS is the overall fastest.

We therefore also evaluated the scaling of kCS on machine B and show the results of the same experiment including fbB with $k \in \{4, 8, 16, 32\}$ in Figure 4. The results are consistent with the results on machine A, with scaling factors $[0.64, 0.50, 0.50, 0.57, 0.67, 1.22]$. This is expected since the only difference is the larger message delay, which is largely avoided due to asynchronous communication and interleaved local computation. However, on some instances, we notice that scaling stops earlier, likely due to the impact of this delay combined with less local work.

To further investigate this, we also considered very large synthetic instances with $n = \{2^{26}, 2^{27}, 2^{28}\}$ vertices, similar to Khan et al. [41], and ran scaling experiments for $k = 4$ on machine B. The results for strong scaling are shown in Figure 5a. Note that we grouped the different weight distributions for better readability, as they performed similarly. On these large instance we observe near optimal scaling factors $[0.54, 0.52, 0.52, 0.53]$ even with a large number of controllers.

We additionally performed weak scaling experiments, where both the number of controllers and the input size doubles in each step. doubling the number of controllers also doubling the input size. Ideally, this would result in an unchanged running time. We used 32 controllers with the graphs of size 2^{26}, 64 controllers with the graphs of size 2^{27} and 128 controllers with the graphs of size 2^{28}. As Figure 5b shows, the results are nearly optimal.

In summary, all our proposed algorithms scale well on traces, whereas on differently structured graphs only the MRep *algorithms and* kCS *perform well.*

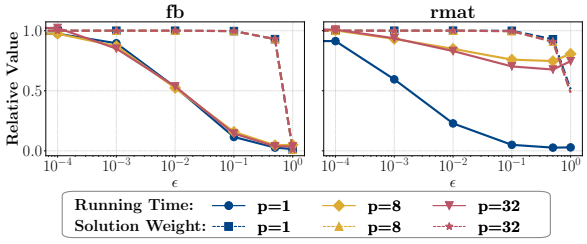

Figure 6: Tradeoff between running time and weight of kMM for $k = \{8, 16\}$ according to ε. Weights are normalized by the result for $\varepsilon = 0$ with the same graph, k, and n_p.

4.4.2 ***Tradeoff between solution quality and running time.*** We proposed a scaling parameter ε for kMM to trade solution quality for running time (see Section 3.4). In this experiment, we investigate the sensitivity to different values of ε.

We used a subset of our instances consisting of the three facebook graphs and the six rmat graphs with 2^{20} vertices. To get a good overview, we vary $\varepsilon \in \{0.0001, 0.001, 0.01, 0.1, 0.5, 1\}$, $k \in \{8, 16\}$ and use different numbers of controllers $n_p \in \{1, 8, 32\}$. Figure 6 shows that for values of ε up to 0.1 the running time decreases significantly, while the weight of the solution is only marginally affected. Only for $\varepsilon \gtrapprox 0.5$, the solution weight starts to decrease noticeably. However, with a larger number of controllers for rmat graphs, we see much less improvement in running time due to the communication overhead. A more thorough examination of the cutoff point $\varepsilon = 0.1$ reveals that for facebook traces, on average, there is a 84.95% reduction in running time, accompanied by only a 0.3% decrease in the weight of the solution. For rmat graphs, we observe a 49.15% running time reduction and 0.27% weight decrease.

In summary, we can drastically improve the running time without losing significant weight. We hence fix $\varepsilon = 0.1$ for all other experiments with kMM.

4.4.3 ***Comparison to Sequential Algorithms.*** As we are the first to propose distributed algorithms for k-DM, we compare to the best sequential algorithms on realistic workloads and synthetic graphs and evaluate the quality and efficiency.

In the first part, we investigate the behavior of the algorithms depending on k. For $k \in \{2, 8, 32, 64\}$, we run all distributed algorithms on one controller (i.e., sequentially) and, due to the observed scaling behavior, on 32 controllers. For the rmat instances, we report results for graphs with 2^{20} vertices. Figure 7 shows the results for traces and rmat instances separately.

Among the repetition-based algorithms MRepLM and MRepS, we observe that MRepS outperforms MRepLM in terms of running time in the sequential and distributed setting for all k, except $k = 2$ on one controller. On average over all instances, MRepS achieves speedups over MRepLM of 0.657×, 1.182×, 1.688×, and 1.951×, with one controller and 1.591×, 2.442×, 3.915×, and 4.593× with 32 controllers for the different values of k. As both algorithms compute the same solution, MRepS *is the clear winner in this comparison.*

For kMM, we first observe that in the sequential setting, it *outperforms one of the so far best offline algorithms* k-EC with average speedup factors of 7.426×, 4.388×, 2.829×, and 2.505×, respectively, as k increases, while incurring only a small weight loss of up to

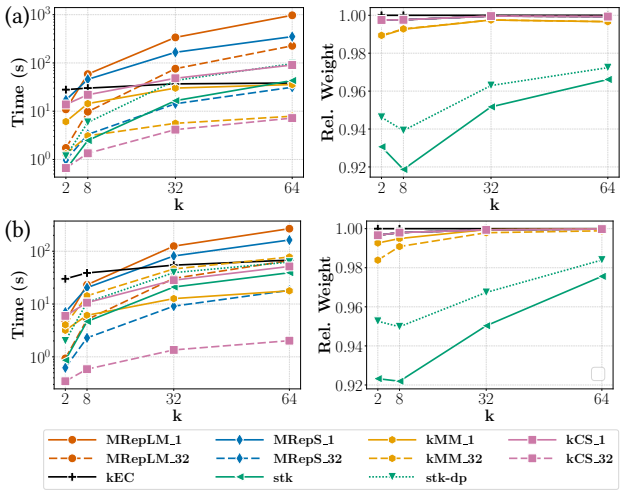

Figure 7: Comparison on facebook (a) and rmat (b) graphs for different k. We normalized the weight results by k-EC.

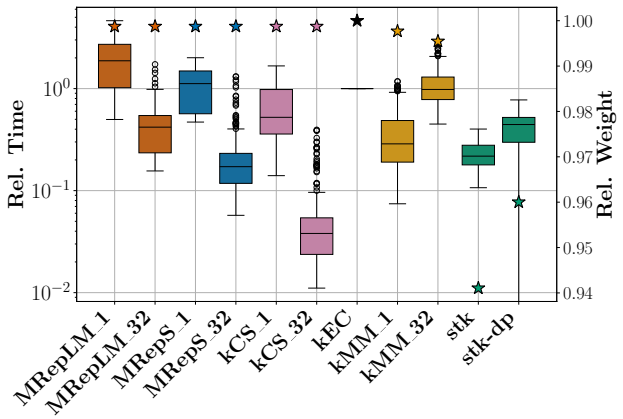

Figure 8: Comparison of the benchmark algorithms on all 66 rmat graphs for $k = 16$. Results are normalized for each instance by k-EC (geometric mean runtime: 14.93s across all instances). The star refers to the weight, the box plots to time.

0.9 %. As previously discussed, kMM scales well only for facebook traces, which is why we focus on these in the following. With 32 controllers, the so far fastest algorithm, stk, still remains faster for $k \in \{2, 4\}$. However, for larger values of $k \in \{32, 64\}$, kMM outperforms stk, achieving speedups of up to 2.946× and 5.488×, respectively. Additionally, kMM results in a weight increase of 3.2 % and 8.1 %, respectively, over stk, while incurring only a small loss of 1.1 % to 0.3 % compared to the best quality result from k-EC.

Finally, we analyze kCS. In terms of matching weight, it stays within 0.3 % percent of the best quality result (k-EC), and increases it by between 2.5 % and 8.6 % over the weight of the fastest algorithm (stk). While kCS does not outperform other offline algorithms sequentially for large k on traces, it scales well to 32 controllers across all instances. Using 32 controllers, kCS achieves speedups over k-EC of 5.3 to 87.2 ×. Against stk, it achieves 2.498×, 8.008×, 15.538×,

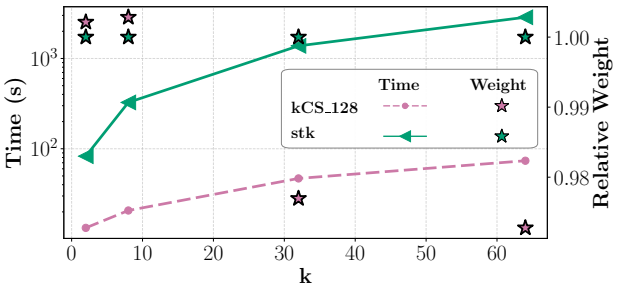

Figure 9: Comparison of the benchmark algorithms on the large graphs for different k, normalized by stk.

and 19.368×, respectively, on rmat graphs for the different values of k, and 0.937×, 1.842×, 3.977×, and 6.004×, respectively, on facebook traces. It is outperformed by stk on facebook instances only for $k = 2$. kCS on 32 controllers also always computes heavier matchings faster than the quality-optimized version stk-dp. Even though kMM on 32 controllers performs well on the facebook traces, kCS consistently outperforms it, achieving speedups over it of 1.961×, 2.302×, 1.35×, and 1.094×, respectively, for increasing k.

In the second part, we compare the benchmark algorithms on all 66 rmat graphs for $k = 16$ and show the results in Figure 8. Note the logarithmic scale for the running time axis. The results resemble the already found results on the synthetic graphs with 2^{20} instances. Unfortunately, stk and stk-dp encounter a segmentation fault on $rmat_b$ with 2^{24} vertices and exponential edge weight distribution. To obtain a fair comparison on the remaining algorithms, we therefore excluded this instance. The results show that kCS on 32 controllers still performs the overall best, being 23.847× faster than k-EC and 4.976× faster than stk. Regarding the matching weight, we see a 0.1% loss compared to k-EC, but a 9.1% increase over stk.

In summary, kCS *outperforms previous static and streaming algorithms by up to multiple factors and also dominates all distributed algorithms.* kMM *run sequentially is more than twice as fast and quality-wise almost as good as* k-EC.

4.4.4 ***Performance on Large Instances.*** In this experiment, we evaluate the capability of our algorithms to handle large-scale problems and compare their performance with the streaming algorithm. Ferdous et al. [24] showed that the offline algorithm k-EC could not be run with a 1 TB limit on all except the smallest graph, mycielskian20. Hence, we only compare stk with kCS on 128 controllers. Figure 9 shows relative running time and solution weight. The geometric mean results indicate that kCS_128 consistently outperforms stk, with runtimes increasing from 13.30s to 73.68s for kCS_128 and from 82.79s to 2859.82s for stk as k scales from 2 to 64. These constitute significant speedups of 6.226×, 15.733×, 29.381×, and 38.813× as k increases. For matching weight, we observe on average changes of +0.2 %, +0.3 %, −2.4 %, and −2.8 %, respectively. One outlier can be observed on the mycielskian20 graph with uniform edge weights and $k = 64$, where kCS achieves a speedup of 151.18×, however at the cost of a loss of 20% in weight.

Overall, kCS *on 128 controllers distinctly outperforms* stk *in terms of running time by several factors with only a little loss in weight on almost all instances.*

5 Related Work

Reconfigurable datacenter networks have recently received much attention [4, 30], both the demand-oblivious versions such as RotorNet [47] and others [1, 3, 46], as well as the demand-aware versions (the focus of our paper), such as Helios [21], ProjecToR [26], and Cerberus [28], and others [13, 14, 60, 61, 63, 64]. Due to the required optimizations, the control plane of demand-aware networks has also been studied intensively already. The problem is related to switch scheduling [45], and researchers have extended classic results to the datacenter context, both in the offline [63] and online [8, 18, 42, 58] setting. Recently, Han et al. [31] proposed a demand-aware distributed scheduling framework for reconfigurable datacenter networks. The problem is further related to graph augmentation problems [27, 49], where the optical links can be compared to so-called ghost edges [51] that can be added into the graph (modeling a hybrid electrical-optical datacenter network); in classic literature, however, the goal of these extra links is not to improve throughput but to provide small world properties [52] or to minimize the network diameter [10, 17]. From an online perspective, the problem is also related to caching where reconfigurable links can dynamically extend an otherwise fixed datacenter topology [9]. For throughput-optimized configurations of optical circuit switches, we need to find entire matchings in the demand network. Hanauer et al. have proposed the state-of-the-art algorithms for the weighted k-disjoint matchings problem in the static offline [35] and dynamic [33] settings, while a state-of-the-art algorithm for the static semi-streaming setting was proposed by Ferdous et al. [25].

Much less is known about distributed control algorithms. One exception is ProjecTor [26, 42] which however focuses on a slightly different architecture, based on free-space optics.

The problem of computing maximum-weight k-disjoint matchings is equivalent to the maximum k edge-colorable subgraph problem, which has been extensively studied. Several fundamental theoretical results are known for this problem. Feige et al. [23] demonstrated that approximating the problem within a ratio better than $1 - \varepsilon$ is NP-hard. The optimal solution can be formulated as LP [36], which has been tested experimentally [35] but is slow even for small instances. Hochbaum [36] proves an algorithm that iteratively computes k maximum coverage, which includes matching, to have an approximation ratio of $1 - (1 - \frac{1}{k})^k$. For small k, better bounds are provided: Chen et al. [15] give a ratio of 0.842 for $k = 2$, and Kamiński and Kowalik [38] provide ratios for $3 \leq k \leq 7$ as 13/15, 9/11, 19/22, 23/27, and 22/25, respectively. When using greedy maximal instead of maximum matchings, the approximation ratio can not be better than $\frac{1}{2}$ [35]. El-Hayek et al. [19] give a dynamic $(2 + \varepsilon)(k + 1)/k$-approximation algorithm for unweighted k-DM against an oblivious and a $(7 + \varepsilon)(3k + 3)/(3k - 1)$-approximation against an adaptive adversary, both with polylogarithmic update time. In addition, the problem has been researched in the online [22] and parameterized setting [2, 29].

The broader field of graph matching has been the subject of considerable research, including the development of parallel and distributed approaches to classical matching problems. Efficient distributed matching algorithms have been proposed for problems such as simple matchings [7, 11].

6 Conclusion and Future Work

In this work, we initiated the study of distributed control algorithms for reconfigurable datacenters based on optical circuit switches, and, in particular, for k-DM. We showed that our new algorithm kCS outperforms known algorithms with up to 1-2 orders of magnitude and little loss in solution weight.

For dynamic scheduling, our fast algorithms provide a distributed baseline, and the optical circuit switch configurations can be recomputed periodically. The switches' reconfiguration latency during updates can be overlapped with computation of the next configuration. An interesting direction for future work is to support dynamic updates without full recomputation in the dynamic or the adjustable batch-dynamic setting. Further, it will be interesting to study how to further improve the scalability of our approaches. It would also be interesting to study additional performance metrics, such as flow completion times, and to explore extensions to switch types based on b-matchings. Further, full-system evaluations in simulated or real environments would offer insights in the overall system performance. However, this would require a significant additional methodological contribution. In particular, a simulator would have to be highly scalable yet accurate, possibly using both packet-level and flow-level approaches.

A Edge-Balanced Vertex Partitioning

We study graphs distributed across multiple controllers, each owning a subset of vertices and incident edges, and ask whether partitioning can improve runtime. Classical partitioners (METIS [39], KaHIP [57]) optimize vertex balance and minimize weighted edge cut, that is, the sum of weights of inter-partition edges. However, because our algorithms color edges, balance must reflect the balance over edges per controller, not vertices. We refer to this optimization problem as *edge-balanced vertex partitioning*.

Offline Edge-Balanced Vertex Partitioning. We introduce two ways to achieve edge-balanced vertex partitioning using classical offline partitioners. In the first approach, we focus on unweighted graphs to reduce the number of cut edges rather than the cut weight. As this method minimizes the number of cross edges, it improves the efficiency. When using a partitioner P, we refer to this setting as U-P. In the second approach, we use the capabilities of classical partitioners to process graphs with vertex weights. To gain edge-balance, we pre-process the input instance by assigning each vertex a weight equal to its degree. We use this in combination with the first approach and, when using a partitioner P, refer to this as UW-P.

Distributed Streaming Edge-Balanced Vertex Partitioning. In recent literature, streaming partitioning has become more popular [6, 20, 62]. In particular, using the *fennel* cost function [6], which assigns a vertex v to a block V_i which maximizes $|v_i \cap N(v)| - \alpha * \gamma * |V_i|^{\gamma-1}$ according to the two fennel parameters α and γ.

We introduce Distributed Weighted Fennel (DW-FENNEL) by combining the approaches of Faraj et al. [20] and Battaglino et al. [6]. Faraj et at. [20] introduce a generalized fennel for the sequential setting with an edge weight function w and a vertex weight function c as $\sum_{u \in V_i \cap N(v)} w(u,v) - c(v) * \alpha * \gamma * c(V_i)^{\gamma-1}$. To achieve edge balance, we assign the vertex weight function $c(v) = deg(v)$ and use the edge demand for the edge weight function $w(e) = D(e)$.

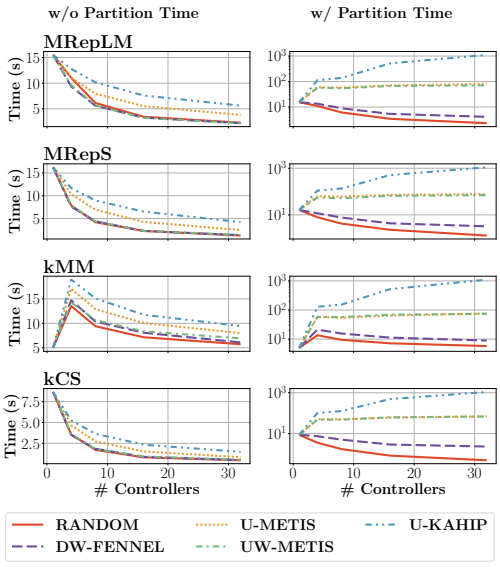

Figure 10: Effects of partitioning on the computation time.

We adapt the distributed approach of Battaglino et al. [6], which reassigns vertices to the partition with maximal fennel score over multiple passes, replacing the score with the weighted edge-balance cost. After each reassignment, we perform global communication of assignments. We use communication after every reassignment, compared to a single communication per round as in [6], since the latter caused stampedes toward the perceived smallest partition and high imbalance.

Impact of Partitioning on Running Time. We next explore how partitioning affects our distributed algorithms. Surprisingly, we find that the tested partitioning techniques do not improve the overall running time compared to random partitioning.

We evaluate offline partitioners METIS and KaHIP using unweighted variants (U-METIS, U-KaHIP) and a degree-weighted METIS (UW-METIS). Since U-KaHIP is slow and both tools failed on original edge weights, we preprocess weights and apply the aforementioned second approach only to METIS. We also test the distributed streaming method DW-FENNEL and a uniform RANDOM baseline. Experiments use six RMAT graphs with 2^{20} vertices and two Facebook traces (excluding fbB). We vary $k \in \{4, 8\}$ and controllers $n_p \in \{1, 4, 8, 16, 32\}$, reporting execution and partitioning times separately in Figure 10.

Overall, U-METIS and U-KaHIP perform worst. This can be explained since they achieve very good vertex balancing. However, the edge imbalance increases with the use of more controllers, i.e., only a few controllers handle the vertices with the majority of the edges and therefore perform the most work. Hence, they are the slowest for a larger number of controllers with all algorithms.

Using UW-METIS or DW-FENNEL can lead to a reduction in the execution time. However, in particular when including the partitioning times, RANDOM is the overall best choice.

In conclusion, graph partitioning fails to improve performance due to its high computational cost compared to k-DM and the limited locality of the tested instances, resulting in high edge cuts under edge-balance constraints.

Acknowledgments

The computational results on Machine B have been achieved using the Austrian Scientific Computing (ASC) infrastructure. The research was further supported by the European Research Council (ERC), grant agreement No. 864228 (AdjustNet), Horizon 2020, 2020-2025

References

[1] Vamsi Addanki, Chen Avin, and Stefan Schmid. 2023. Mars: Near-Optimal Throughput with Shallow Buffers in Reconfigurable Datacenter Networks. *Proc. ACM Meas. Anal. Comput. Syst.* 7, 1, Article 2 (mar 2023), 43 pages. doi:10.1145/3579312

[2] Akanksha Agrawal, Madhumita Kundu, Abhishek Sahu, Saket Saurabh, and Prafullkumar Tale. 2022. Parameterized Complexity of Maximum Edge Colorable Subgraph. *Algorithmica* 84, 10 (2022), 3075–3100. doi:10.1007/S00453-022-01003-0

[3] Daniel Amir, Nitika Saran, Tegan Wilson, Robert Kleinberg, Vishal Shrivastav, and Hakim Weatherspoon. 2024. Shale: A Practical, Scalable Oblivious Reconfigurable Network. In *Proceedings of the ACM SIGCOMM 2024 Conference* (Sydney, NSW, Australia) *(ACM SIGCOMM '24)*. 449–464.

[4] Chen Avin and Stefan Schmid. 2025. Revolutionizing Datacenter Networks via Reconfigurable Topologies. In *Communications of the ACM (CACM)*.

[5] Hitesh Ballani, Paolo Costa, Raphael Behrendt, Daniel Cletheroe, Istvan Haller, Krzysztof Jozwik, Fotini Karinou, Sophie Lange, Kai Shi, Benn Thomsen, et al. 2020. Sirius: A flat datacenter network with nanosecond optical switching. In *Proc. ACM SIGCOMM 2020 Conference*. 782–797.

[6] Casey Battaglino, Pienta Pienta, and Richard Vuduc. 2015. GraSP: distributed streaming graph partitioning. Barcelona Supercomputing Center. doi:10.5821/hpgm15.3 Accepted: 2015-07-29T11:40:29Z.

[7] Soheil Behnezhad, Laxman Dhulipala, Hossein Esfandiari, Jakub Lacki, Vahab S. Mirrokni, and Warren Schudy. 2021. Massively Parallel Computation via Remote Memory Access. *ACM Trans. Parallel Comput.* 8, 3 (2021), 13:1–13:25. doi:10.1145/3470631

[8] Marcin Bienkowski, David Fuchssteiner, Jan Marcinkowski, and Stefan Schmid. 2020. Online Dynamic B-Matching: With Applications to Reconfigurable Datacenter Networks. *SIGMETRICS Perform. Evaluation Rev.* 48, 3 (2020), 99–108. doi:10.1145/3453953.3453976

[9] Marcin Bienkowski, David Fuchssteiner, and Stefan Schmid. 2023. Optimizing Reconfigurable Optical Datacenters: The Power of Randomization. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2023, Denver, CO, USA, November 12-17, 2023*, Dorian Arnold, Rosa M. Badia, and Kathryn M. Mohror (Eds.). ACM, 83:1–83:11. doi:10.1145/3581784.3607057

[10] Davide Bilò, Luciano Gualà, and Guido Proietti. 2012. Improved approximability and non-approximability results for graph diameter decreasing problems. *Theor. Comput. Sci.* 417 (2012), 12–22. doi:10.1016/J.TCS.2011.05.014

[11] Marcel Birn, Vitaly Osipov, Peter Sanders, Christian Schulz, and Nodari Sitchinava. 2013. Efficient Parallel and External Matching. In *Euro-Par 2013 Parallel Processing - 19th International Conference, Aachen, Germany, August 26-30, 2013. Proceedings (Lecture Notes in Computer Science, Vol. 8097)*, Felix Wolf, Bernd Mohr, and Dieter an Mey (Eds.). Springer, 659–670. doi:10.1007/978-3-642-40047-6_66

[12] Deepayan Chakrabarti, Yiping Zhan, and Christos Faloutsos. 2004. R-MAT: A Recursive Model for Graph Mining. In *Proceedings of the 2004 SIAM International Conference on Data Mining (SDM)*. Society for Industrial and Applied Mathematics, 442–446. doi:10.1137/1.9781611972740.43

[13] Kai Chen, Ankit Singla, Atul Singh, Kishore Ramachandran, Lei Xu, Yueping Zhang, Xitao Wen, and Yan Chen. 2014. OSA: An optical switching architecture for data center networks with unprecedented flexibility. *IEEE/ACM Transactions on Networking (TON)* 22, 2 (2014), 498–511.

[14] Li Chen, Kai Chen, Zhonghua Zhu, Minlan Yu, George Porter, Chunming Qiao, and Shan Zhong. 2017. Enabling Wide-Spread Communications on Optical Fabric with Megaswitch. In *Proc. 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)* (Boston, MA, USA). USA, 577–593.

[15] Zhi-Zhong Chen, Sayuri Konno, and Yuki Matsushita. 2010. Approximating maximum edge 2-coloring in simple graphs. 158, 17 (2010), 1894–1901. doi:10.1016/J.DAM.2010.08.010

[16] Timothy A. Davis and Yifan Hu. 2011. The university of Florida sparse matrix collection. *ACM Trans. Math. Softw.* 38, 1 (Dec. 2011), 1:1–1:25. doi:10.1145/2049662.2049663

[17] Erik D. Demaine and Morteza Zadimoghaddam. 2010. Minimizing the Diameter of a Network Using Shortcut Edges. In *Algorithm Theory - SWAT 2010, 12th Scandinavian Symposium and Workshops on Algorithm Theory, Bergen, Norway, June 21-23, 2010. Proceedings (Lecture Notes in Computer Science, Vol. 6139)*, Haim Kaplan (Ed.). Springer, 420–431. doi:10.1007/978-3-642-13731-0_39

[18] Michael Dinitz and Benjamin Moseley. 2020. Scheduling for Weighted Flow and Completion Times in Reconfigurable Networks. In *39th IEEE Conference on Computer Communications, INFOCOM 2020, Toronto, ON, Canada, July 6-9, 2020*. IEEE, 1043–1052. doi:10.1109/INFOCOM41043.2020.9155537

[19] Antoine El-Hayek, Kathrin Hanauer, and Monika Henzinger. 2025. On b-Matching and Fully-Dynamic Maximum k-Edge Coloring. In *SAND 2025 (LIPIcs, Vol. 330)*, Kitty Meeks and Christian Scheideler (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 4:1–4:23. doi:10.4230/LIPICS.SAND.2025.4

[20] Marcelo Fonseca Faraj and Christian Schulz. 2022. Buffered Streaming Graph Partitioning. *ACM Journal of Experimental Algorithmics* 27 (Dec. 2022), 1–26. doi:10.1145/3546911

[21] Nathan Farrington, George Porter, Sivasankar Radhakrishnan, Hamid Hajabdolali Bazzaz, Vikram Subramanya, Yeshaiahu Fainman, George Papen, and Amin Vahdat. 2011. Helios: a hybrid electrical/optical switch architecture for modular data centers. *ACM SIGCOMM Comput. Commun. Rev. (CCR)* 41, 4 (2011), 339–350.

[22] Lene M. Favrholdt and Morten N. Nielsen. 2003. On-Line Edge-Coloring with a Fixed Number of Colors. *Algorithmica* 35, 2 (2003), 176–191. doi:10.1007/S00453-002-0992-3

[23] Uriel Feige, Eran Ofek, and Udi Wieder. 2002. Approximating Maximum Edge Coloring in Multigraphs. In *Approximation Algorithms for Combinatorial Optimization, 5th International Workshop, APPROX 2002, Rome, Italy, September 17-21, 2002, Proceedings (Lecture Notes in Computer Science, Vol. 2462)*, Klaus Jansen, Stefano Leonardi, and Vijay V. Vazirani (Eds.). Springer, 108–121. doi:10.1007/3-540-45753-4_11

[24] S. M. Ferdous, Alex Pothen, Arif Khan, Ajay Panyala, and Mahantesh Halappanavar. 2021. A Parallel Approximation Algorithm for Maximizing Submodular *b*-Matching. In *Proceedings of the 2021 SIAM Conference on Applied and Computational Discrete Algorithms, ACDA 2021, Virtual Conference, July 19-21, 2021*, Michael Bender, John Gilbert, Bruce Hendrickson, and Blair D. Sullivan (Eds.). SIAM, 45–56. doi:10.1137/1.9781611976830.5

[25] S. M. Ferdous, Bhargav Samineni, Alex Pothen, Mahantesh Halappanavar, and Bala Krishnamoorthy. 2024. Semi-Streaming Algorithms for Weighted k-Disjoint Matchings. In *32nd Annual European Symposium on Algorithms, ESA 2024, September 2-4, 2024, Royal Holloway, London, United Kingdom (LIPIcs, Vol. 308)*, Timothy Chan, Johannes Fischer, John Iacono, and Grzegorz Herman (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 53:1–53:19. doi:10.4230/LIPICS.ESA.2024.53

[26] Monia Ghobadi, Ratul Mahajan, Amar Phanishayee, Nikhil Devanur, Janardhan Kulkarni, Gireeja Ranade, Pierre-Alexandre Blanche, Houman Rastegarfar, Madeleine Glick, and Daniel Kilper. 2016. Projector: Agile reconfigurable data center interconnect. In *Proc. ACM SIGCOMM 2016 Conference*. ACM, 216–229.

[27] Andrew Gozzard, Max Ward, and Amitava Datta. 2018. Converting a network into a small-world network: Fast algorithms for minimizing average path length through link addition. *Inf. Sci.* 422 (2018), 282–289. doi:10.1016/J.INS.2017.09.020

[28] Chen Griner, Johannes Zerwas, Andreas Blenk, Manya Ghobadi, Stefan Schmid, and Chen Avin. 2021. Cerberus: The Power of Choices in Datacenter Topology Design (A Throughput Perspective). *Proc. ACM SIGMETRICS* (2021), 1–33.

[29] Niels Grüttemeier, Christian Komusiewicz, and Nils Morawietz. 2020. Maximum Edge-Colorable Subgraph and Strong Triadic Closure Parameterized by Distance to Low-Degree Graphs. In *17th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2020) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 162)*, Susanne Albers (Ed.). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 26:1–26:17. doi:10.4230/LIPIcs.SWAT.2020.26 ISSN: 1868-8969.

[30] Matthew Nance Hall, Klaus-Tycho Foerster, Stefan Schmid, and Ramakrishnan Durairajan. 2021. A survey of reconfigurable optical networks. *Optical Switching and Networking* 41 (2021), 100621.

[31] Subin Han, Eunsok Lee, Hyunkyung Yoo, Namseok Ko, and Sangheon Pack. 2025. Demand-Aware Distributed Scheduling With Adaptive Buffer Control in Reconfigurable Data Center Networks. *IEEE Trans. Cloud Comput.* 13, 3 (2025), 783–793. doi:10.1109/TCC.2025.3568369

[32] Kathrin Hanauer, Sophia Heck, and Stefan Schmid. 2026. *DistroMatch*. doi:10.5281/zenodo.19205433

[33] Kathrin Hanauer, Monika Henzinger, Lara Ost, and Stefan Schmid. 2023. Dynamic Demand-Aware Link Scheduling for Reconfigurable Datacenters. In *IEEE INFOCOM 2023 - IEEE Conference on Computer Communications*. 1–10. doi:10.1109/INFOCOM53939.2023.10229050 ISSN: 2641-9874.

[34] Kathrin Hanauer, Monika Henzinger, Lara Ost, and Stefan Schmid. 2025. Dynamic Demand-Aware Link Scheduling for Reconfigurable Datacenters. https://arxiv.org/abs/2301.05751 _eprint: 2301.05751.

[35] Kathrin Hanauer, Monika Henzinger, Stefan Schmid, and Jonathan Trummer. 2022. Fast and Heavy Disjoint Weighted Matchings for Demand-Aware Datacenter Topologies. In *IEEE INFOCOM 2022 - IEEE Conference on Computer Communications*. 1649–1658. doi:10.1109/INFOCOM48880.2022.9796921 ISSN: 2641-9874.

[36] Dorit S. Hochbaum. 1996. Approximating covering and packing problems: set cover, vertex cover, independent set, and related problems. In *Approximation algorithms for NP-hard problems*. PWS Publishing Co., USA, 94–143.

[37] Jaap-Henk Hoepman. 2004. Simple Distributed Weighted Matchings. *CoRR* cs.DC/0410047 (2004). http://arxiv.org/abs/cs.DC/0410047

[38] Marcin Jakub Kaminski and Lukasz Kowalik. 2014. Beyond the Vizing's Bound for at Most Seven Colors. *SIAM J. Discret. Math.* 28, 3 (2014), 1334–1362. doi:10.1137/120899765

[39] George Karypis and Vipin Kumar. 1998. A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs. *SIAM J. Sci. Comput.* 20, 1 (1998), 359–392. doi:10.1137/S1064827595287997

[40] Wolfgang Kellerer, Patrick Kalmbach, Andreas Blenk, Arsany Basta, Martin Reisslein, and Stefan Schmid. 2019. Adaptable and Data-Driven Softwarized Networks: Review, Opportunities, and Challenges. *Proc. IEEE* 107, 4 (2019), 711–731. doi:10.1109/JPROC.2019.2895553

[41] Arif M. Khan, Alex Pothen, Md Mostofa Ali Patwary, Mahantesh Halappanavar, Nadathur Rajagopalan Satish, Narayanan Sundaram, and Pradeep Dubey. 2016. Designing scalable b-Matching algorithms on distributed memory multiprocessors by approximation. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2016, Salt Lake City, UT, USA, November 13-18, 2016*, John West and Cherri M. Pancake (Eds.). IEEE Computer Society, 773–783. doi:10.1109/SC.2016.65

[42] Janardhan Kulkarni, Stefan Schmid, and Pawel Schmidt. 2021. Scheduling Opportunistic Links in Two-Tiered Reconfigurable Datacenters. In *SPAA '21: 33rd ACM Symposium on Parallelism in Algorithms and Architectures, Virtual Event, USA, 6-8 July, 2021*, Kunal Agrawal and Yossi Azar (Eds.). ACM, 318–327. doi:10.1145/3409964.3461786

[43] Yuliang Li, Rui Miao, Hongqiang Harry Liu, Yan Zhuang, Fei Feng, Lingbo Tang, Zheng Cao, Ming Zhang, Frank Kelly, Mohammad Alizadeh, and Minlan Yu. 2019. HPCC: high precision congestion control. In *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM '19)*. Association for Computing Machinery, New York, NY, USA, 44–58. doi:10.1145/3341302.3342085

[44] Fredrik Manne and Mahantesh Halappanavar. 2014. New Effective Multithreaded Matching Algorithms. In *2014 IEEE 28th International Parallel and Distributed Processing Symposium, Phoenix, AZ, USA, May 19-23, 2014*. IEEE Computer Society, 519–528. doi:10.1109/IPDPS.2014.61

[45] Nick McKeown. 1999. The iSLIP scheduling algorithm for input-queued switches. *IEEE/ACM Trans. Netw.* 7, 2 (1999), 188–201. doi:10.1109/90.769767

[46] William M Mellette, Rajdeep Das, Yibo Guo, Rob McGuinness, Alex C Snoeren, and George Porter. 2020. Expanding across time to deliver bandwidth efficiency and low latency. In *Proc. 17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. 1–18.

[47] William M Mellette, Alex Forencich, Rukshani Athapathu, Alex C Snoeren, George Papen, and George Porter. 2024. Realizing RotorNet: Toward Practical Microsecond Scale Optical Networking. In *Proceedings of the ACM SIGCOMM 2024 Conference (ACM SIGCOMM '24)*. 392–414.

[48] William M Mellette, Rob McGuinness, Arjun Roy, Alex Forencich, George Papen, Alex C Snoeren, and George Porter. 2017. Rotornet: A scalable, low-complexity, optical datacenter network. In *Proc. ACM SICOMM 2017 conference*. ACM, 267–280.

[49] Adam Meyerson and Brian Tagiku. 2009. Minimizing Average Shortest Path Distances via Shortcut Edge Addition. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, 12th International Workshop, APPROX 2009, and 13th International Workshop, RANDOM 2009, Berkeley, CA, USA, August 21-23, 2009. Proceedings (Lecture Notes in Computer Science, Vol. 5687)*, Irit Dinur, Klaus Jansen, Joseph Naor, and José D. P. Rolim (Eds.). Springer, 272–285. doi:10.1007/978-3-642-03685-9_21

[50] Jayadev Misra and David Gries. 1992. A Constructive Proof of Vizing's Theorem. *Inf. Process. Lett.* 41, 3 (1992), 131–133. doi:10.1016/0020-0190(92)90041-S

[51] Manos Papagelis, Francesco Bonchi, and Aristides Gionis. 2011. Suggesting ghost edges for a smaller world. In *Proceedings of the 20th ACM Conference on Information and Knowledge Management, CIKM 2011, Glasgow, United Kingdom, October 24-28, 2011*, Craig Macdonald, Iadh Ounis, and Ian Ruthven (Eds.). ACM, 2305–2308. doi:10.1145/2063576.2063952

[52] Nikos Parotsidis, Evaggelia Pitoura, and Panayiotis Tsaparas. 2015. Selecting Shortcuts for a Smaller World. In *Proceedings of the 2015 SIAM International Conference on Data Mining, Vancouver, BC, Canada, April 30 - May 2, 2015*, Suresh Venkatasubramanian and Jieping Ye (Eds.). SIAM, 28–36. doi:10.1137/1.9781611974010.4

[53] Ami Paz and Gregory Schwartzman. 2019. A $(2+\epsilon)$-Approximation for Maximum Weight Matching in the Semi-streaming Model. *ACM Trans. Algorithms* 15, 2 (2019), 18:1–18:15. doi:10.1145/3274668

[54] Leon Poutievski, Omid Mashayekhi, Joon Ong, Arjun Singh, Mukarram Tariq, Rui Wang, Jianan Zhang, Virginia Beauregard, Patrick Conner, Steve Gribble, et al. 2022. Jupiter evolving: transforming google's datacenter network via optical circuit switches and software-defined networking. In *Proc. ACM SIGCOMM 2022 Conference*. 66–85.

[55] Robert Preis. 1999. Linear Time 1/2-Approximation Algorithm for Maximum Weighted Matching in General Graphs. In *STACS 99, 16th Annual Symposium on Theoretical Aspects of Computer Science, Trier, Germany, March 4-6, 1999, Proceedings (Lecture Notes in Computer Science, Vol. 1563)*, Christoph Meinel and Sophie Tison (Eds.). Springer, 259–269. doi:10.1007/3-540-49116-3_24

[56] Arjun Roy, Hongyi Zeng, Jasmeet Bagga, George Porter, and Alex C. Snoeren. 2015. Inside the Social Network's (Datacenter) Network. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication (SIGCOMM '15)*. Association for Computing Machinery, New York, NY, USA, 123–137. doi:10.1145/2785956.2787472

[57] Peter Sanders and Christian Schulz. 2013. Think Locally, Act Globally: Highly Balanced Graph Partitioning. In *Proceedings of the 12th International Symposium on Experimental Algorithms (SEA'13) (LNCS, Vol. 7933)*. Springer, 164–175.

[58] Roy Schwartz, Mohit Singh, and Sina Yazdanbod. 2019. Online and Offline Greedy Algorithms for Routing with Switching Costs. *CoRR* abs/1905.02800 (2019). http://arxiv.org/abs/1905.02800 arXiv: 1905.02800.

[59] 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. 2015. Jupiter Rising: A Decade of Clos Topologies and Centralized Control in Google's Datacenter Network. *SIGCOMM Comput. Commun. Rev.* 45, 4 (2015), 183–197. doi:10.1145/2829988.2787508

[60] Ankit Singla, Atul Singh, Kishore Ramachandran, Lei Xu, and Yueping Zhang. 2010. Proteus: a topology malleable data center network. In *Proceedings of the 9th ACM Workshop on Hot Topics in Networks. HotNets 2010, Monterey, CA, USA - October 20 - 21, 2010*, Geoffrey G. Xie, Robert Beverly, Robert Morris, and Bruce Davie (Eds.). ACM, 8. doi:10.1145/1868447.1868455

[61] Min Yee Teh, Zhenguo Wu, and Keren Bergman. 2020. Flexspander: augmenting expander networks in high-performance systems with optical bandwidth steering. *JOCN* 12, 4 (2020), B44–B54. doi:10.1364/JOCN.379487

[62] Charalampos E. Tsourakakis, Christos Gkantsidis, Bozidar Radunovic, and Milan Vojnovic. 2014. FENNEL: streaming graph partitioning for massive scale graphs. In *Seventh ACM International Conference on Web Search and Data Mining, WSDM 2014, New York, NY, USA, February 24-28, 2014*, Ben Carterette, Fernando Diaz, Carlos Castillo, and Donald Metzler (Eds.). ACM, 333–342. doi:10.1145/2556195.2556213

[63] Shaileshh Bojja Venkatakrishnan, Mohammad Alizadeh, and Pramod Viswanath. 2018. Costly circuits, submodular schedules and approximate Carathéodory Theorems. *Queueing Syst. Theory Appl.* 88, 3-4 (2018), 311–347. doi:10.1007/S11134-017-9546-X

[64] Guohui Wang, David G. Andersen, Michael Kaminsky, Konstantina Papagiannaki, T. S. Eugene Ng, Michael Kozuch, and Michael P. Ryan. 2010. c-Through: part-time optics in data centers. In *Proceedings of the ACM SIGCOMM 2010 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, New Delhi, India, August 30 -September 3, 2010*, Shivkumar Kalyanaraman, Venkata N. Padmanabhan, K. K. Ramakrishnan, Rajeev Shorey, and Geoffrey M. Voelker (Eds.). ACM, 327–338. doi:10.1145/1851182.1851222