Fast and Heavy Disjoint Weighted Matchings for Demand-Aware Datacenter Topologies

Kathrin Hanauer
Faculty of Computer Science, *University of Vienna*, Austria

Monika Henzinger
Faculty of Computer Science, *University of Vienna*, Austria

Stefan Schmid
Faculty of Computer Science, *University of Vienna*, Austria
and *TU Berlin*, Berlin, Germany

Jonathan Trummer
Faculty of Computer Science, *University of Vienna*, Austria

***Abstract*—Reconfigurable optical topologies promise to improve the performance in datacenters by dynamically optimizing the physical network in a demand-aware manner. State-of-the-art optical technologies allow to establish and update direct connectivity (in the form of *edge-disjoint matchings*) between top-of-rack switches within microseconds or less. However, to fully exploit temporal structure in the demand, such fine-grained reconfigurations also require fast algorithms for optimizing the interconnecting matchings.**

Motivated by the desire to offload a maximum amount of demand to the reconfigurable network, this paper initiates the study of fast algorithms to find k disjoint heavy matchings in graphs. We present and analyze six algorithms, based on iterative matchings, b-matching, edge coloring, and node-rankings. We show that the problem is generally $\mathcal{NP}$-hard and study the achievable approximation ratios.

An extensive empirical evaluation of our algorithms on both real-world and synthetic traces (88 in total), including traces collected in Facebook datacenters and in HPC clusters reveals that all our algorithms provide high-quality matchings, and also very fast ones come within 95 % or more of the best solution. However, the running times differ significantly and what is the best algorithm depends on k and the acceptable runtime-quality tradeoff.

***Index Terms*—reconfigurable datacenters, optical circuit switches, graph algorithms, matching, edge coloring**

I. Introduction

With the popularity of data-centric applications, network traffic in datacenters is growing explosively [1], [2]. Accordingly, over the last years, several novel datacenter topologies have been proposed to improve network efficiency and performance, e.g., [3]–[5]. These network topologies typically have in common that they are oblivious to the traffic they serve.

Emerging reconfigurable optical technologies enable an intriguing alternative to existing datacenter network designs [6]–[8]: these technologies allow to enhance existing datacenter networks with reconfigurable optical matchings, e.g., one disjoint matching per optical circuit switch [9]–[15]. These matchings can be adapted towards the traffic demand, exploiting temporal and spatial structure [16]. State-of-the-art technologies allow in principle to change such matchings within microseconds or even less [14]. Given these reconfiguration times, the bottleneck becomes now how to *compute* such matchings fast in the control plane.

Accordingly, this paper initiates the study of fast algorithms to find k disjoint weighted matchings in graphs. Here, k is the number of optical circuit switches, each of which provides one reconfigurable matching. The matchings should be heavy, i.e., carry a maximal amount of traffic. Existing algorithmic work on the matching problem typically focuses on computing a single matching.

More formally, this paper considers the following model and terminology. We are given a weighted graph $G = (V, E, \mathcal{D})$ where $\mathcal{D} : V \times V \to \mathbb{N}_0$ describes a *demand* for each pair of vertices. A *matching* is a subset of edges that have no common vertices. The goal is to find k pairwise edge-disjoint matchings $\mathcal{M}_1, \ldots, \mathcal{M}_k$, such that $\sum_{1\le i\le k} \mathcal{D}(\mathcal{M}_i)$ is maximized. We show that this problem is $\mathcal{NP}$-hard.

Note that this problem is different from the well-known b-matching problem: Given a triangle, a 2-matching could choose all edges in the graph, while 2 disjoint matchings consist of 2 edges of the graph.

Perhaps a natural approach to find such maximum weight k disjoint matchings would be to repeatedly remove the edges of individual maximum weight matchings from the graph, leading to a polynomial-time algorithm. However, this approach is not optimal: Consider a triangle where each vertex is additionally adjacent to a further vertex of degree one. Thus, the graph has six vertices and six edges. Now assume unit edge weights and consider $k = 3$. The unique maximum weight matching consists of exactly the three edges incident to the degree-one

This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (Grant agreement No. 101019564 "The Design of Modern Fully Dynamic Data Structures (MoDynStruct)" and Grant agreement No. 864228 "Self-Adjusting Networks (AdjustNet)"), and from the Austrian Science Fund (FWF) project "Fast Algorithms for a Reactive Network Layer (ReactNet)", P 33775-N, with additional funding from the *netidee SCIENCE Stiftung*, 2020–2024.

{kathrin.hanauer, monika.henzinger, jonathan.trummer}@univie.ac.at, stefan.schmid@tu-berlin.de

vertices. By removing them we are left with a triangle, and hence the second maximum weight matching has size one, as does the third. An optimal solution, however, has size six: Each of the three matchings consists of one triangle edge plus the unique edge incident to the remaining triangle vertex.

Contributions: Motivated by novel optical technologies which allow to enhance fixed datacenter topologies with reconfigurable matchings, this paper studies algorithms for the fast computation of k heavy disjoint matchings. We show that the problem is $\mathcal{NP}$-hard and propose six efficient algorithms: three algorithms are based on the iterative computation of simple matchings, one algorithm leverages a connection to the related b-matching problem, one algorithm is an adaptation of an edge-coloring algorithm, and one node-centered algorithm uses a rating function that depends on the weights of a node's incident edges. Additionally, we study three postprocessing strategies to improve the weight of an existing matching, and discuss the achievable approximation ratios.

We perform an extensive evaluation of the quality and running time of our algorithms on a diverse set of instances, which include both real-world traces as well as synthetic traces, 88 instances in total. In particular, we consider six traces from Facebook datacenters, four traces from a High-Performance Computing (HPC) cluster, three widely-used synthetic pFabric traces, nine instances from the Florida sparse matrix collection, and 66 Kronecker graphs.

Our empirical results show that our algorithms constantly compute high-quality matchings. The running times however vary and which is the best algorithm depends on the value of k and the affordable tradeoff between running time and solution quality. For small values of k, an iterative approach is most attractive, especially when combined with a local swapping strategy: the algorithm `GPA-It`, which is based on the `Global Paths` matching algorithm, combined with the `LocalSwaps` postprocessing routine provides low running times and high-quality matchings whose weight is within 95 % or more of the best algorithm (executed with a 4 h time limit). For larger values of k, our edge coloring-based algorithm `k-EC` provides the best performance as its running time barely increases with k; for $k \geq 4$, its quality score is always at least within 93 % of the best algorithm, and at least 96 % on average. If running time is not of concern, the iterative algorithm `Blossom-It` can be an attractive choice.

As a contribution to the research community, to ensure reproducibility and to facilitate follow-up work, we will make all our experimental artefacts including our implementation (as open source) publicly available together with this paper.

Organization: The remainder of this paper is organized as follows. Sect. II introduces preliminaries and discusses related problems and prior work. We present our algorithms in Sect. III and report on our experimental results in Sect. IV. We conclude by discussing future research directions in Sect. V.

II. Preliminaries

A. Basic Definitions

We model our problem as a simple, undirected, edge-weighted graph $G = (V, E, \mathcal{D})$ with vertex set V, edge set E, and a non-negative integer *weight* or *demand* $\mathcal{D} : V \times V \to \mathbb{N}_0$ for each pair of vertices, which corresponds to the amount of communication (data flow) between them. We assume the demand to be symmetric, i.e., $\mathcal{D}(u, v) = \mathcal{D}(v, u)$ for all $u \neq v \in V$ and $\mathcal{D}(u, v) > 0$ iff $\{u, v\} \in E$. As shorthand notation, we also write $\mathcal{D}(e) = \mathcal{D}(u, v)$ for $e = \{u, v\}$. Note that because G is simple, $\mathcal{D}(v, v) = 0$ always. As usual, $n = |V|$ and $m = |E|$. Furthermore, we denote by $D = \max_{u,v \in V} \mathcal{D}(u, v)$ the maximum demand. An edge $e = \{u, v\}$ is *incident* to its *end vertices* u and v, and u and v are said to be *adjacent*. The *neighborhood* of a vertex v is $N(v) = \{u \mid \{u, v\} \in E\}$ and its *degree* is $\deg(v) = |N(v)|$. We denote the *maximum degree* by $\Delta = \max_{v \in V} \deg(v)$. A *path* $\mathcal{P}$ is a sequence of edges $\mathcal{P} = (e_1, e_2, \ldots, e_k)$ such that e_i and e_{i+1}, $1 \leq i < k$, share a common end vertex and no vertex appears more than once.

A *matching* $\mathcal{M} \subseteq E$ is a set of edges such that no vertex is incident to two edges contained in $\mathcal{M}$. In our context, the *weight* of a matching $\mathcal{M}$ is $\mathcal{D}(\mathcal{M}) = \sum_{e \in \mathcal{M}} \mathcal{D}(e)$. An edge e is said to be *matching* if $e \in \mathcal{M}$ and *non-matching* otherwise. A vertex is said to be *free* (w.r.t. $\mathcal{M}$) if it is not incident to a matching edge. An *alternating path* is a path that alternatingly consists of matching and non-matching edges. Given a matching $\mathcal{M}$ and a path $\mathcal{P}$, we denote by $\mathcal{M} \oplus \mathcal{P}$ the set of edges obtained as the symmetric difference of $\mathcal{M}$ and the edges in $\mathcal{P}$. An *augmenting path* is an alternating path $\mathcal{P}$ such that $\mathcal{M} \oplus \mathcal{P}$ is a matching and $\mathcal{D}(\mathcal{M}) < \mathcal{D}(\mathcal{M} \oplus \mathcal{P})$. A *$k$-disjoint matching* $\mathcal{M}$ is a collection of k matchings $(\mathcal{M}_1, \ldots, \mathcal{M}_k)$ that are pairwise disjoint. We slightly abuse notation and use $\mathcal{M}$ both for the collection of disjoint matchings as well as their union $\mathcal{M}_1 \cup \cdots \cup \mathcal{M}_k$.

In this paper, we study the k-Disjoint Matching *(k-DjM)* problem: Given a graph $G = (V, E, \mathcal{D})$ and an integer $k \in \mathbb{N}$, find a k-disjoint matching $\mathcal{M} = (\mathcal{M}_1, \ldots, \mathcal{M}_k)$ such that $\mathcal{D}(\mathcal{M}) = \sum_{1 \leq i \leq k} \mathcal{D}(\mathcal{M}_i)$ is maximized. If the maximum demand $D = 1$, we speak of the *unweighted* k-disjoint matching problem.

B. Related Problems and Prior Work

Our problem is especially motivated by reconfigurable datacenter networks in which optical switches can be used to augment a given fixed (electrical) topology, typically a Clos topology [1], with additional matchings between the top-of-rack switches [8], [9], [11]–[14], [17]. In prior work, these matchings are typically optimized individually and not for runtime [18]. In this regard, our paper is also related to graph augmentation problems [19], [20] where a given (fixed) graph needs to be enhanced with an optimal number of "extra edges", sometimes also referred to as "ghost edges" [21]: the objective in this literature is typically to provide small world properties [22] or minimize the network diameter [23], [24].

However, these algorithms are not directly applicable to our problem where we need to add entire matchings rather than individual edges.

The WEIGHTED MATCHING problem essentially corresponds to the special case of 1-DJM. Edmonds [25], [26] was the first to give a polynomial-time algorithm, which is known as the *blossom algorithm* and has a running time of $\mathcal{O}(mn^2)$. In a series of improvements [27]–[32], the running time has been reduced further to $\mathcal{O}(n(m+n\log n))$ [33] for the general case and to $\mathcal{O}(m\sqrt{n\log n}\log(nD))$ [34] for integer weights. For dense graphs, the fastest algorithm on integer-weighted graphs is randomized and runs in time $\tilde{\mathcal{O}}(Dn^{\omega})$ [35], where ω is the exponent in the running time of fast matrix multiplication algorithms ($\omega < 2.38$ [36]). A widely known simple greedy algorithm (cf. Sect. III) yields a $\frac{1}{2}$-approximation and runs in time $\mathcal{O}(m\log n)$. Both approximation ratio and running time have been subject to improvement over the years, leading to a $(1-\epsilon)$-approximation algorithm with $\mathcal{O}(m/\epsilon\log(1/\epsilon))$ running time for arbitrary edge weights and $\mathcal{O}(m/\epsilon\log D)$ running time for integer weights [37].

Different algorithms have been proposed and evaluated to tackle the problem in practice. Drake and Hougardy [38] experimentally compared the already mentioned greedy algorithm to the LD algorithm by Preis [39] and to the Path Growing Algorithm (PGA) [40] and showed that a variant of PGA, PGA', performs very well in practice. Maue and Sanders [41] later suggested the Global Paths Algorithm (GPA) and showed in an extensive study that in combination with a postprocessing routine called ROMA, it yields the best experimental results in comparison to the simple greedy algorithm and PGA'.

(WEIGHTED) PERFECT MATCHING is a restricted version that disallows free vertices. It can be solved in polynomial time by a variant of the blossom algorithm [42].

(WEIGHTED) b-MATCHING is a generalization of WEIGHTED MATCHING, where each vertex may be incident to up to b edges contained in the matching. In contrast to k-DISJOINT MATCHING, a b-matching need not be composed of b pairwise disjoint 1-matchings. Thus, every k-disjoint matching is a k-matching, but not necessarily vice-versa (the edges of a triangle, e.g., form a 2-matching but not a 2-disjoint matching). This problem can be solved exactly in $\mathcal{O}(\min\{bn, n\log n\}(m+n\log n))$ time [43] and approximated by a greedy algorithm analogous to the greedy weighted matching algorithm with a performance guarantee of $\frac{1}{2}$ [44] (cf. Sect. III). A $(1+\epsilon)$-approximation can be achieved in $\mathcal{O}_\epsilon(m\alpha(m,n))$ time [45].

Khan et al. [46] compared the performance of the b-matching variants of the simple greedy algorithm, PGA, PGA', and LD to their new algorithm b-SUITOR, which computes the same solution as the greedy algorithm, but is parallelizable and faster than PGA'. Recently, algorithms for the b-matching problem have been evaluated in the online setting in a similar context of data center reconfiguration [47].

The EDGE COLORING problem consists in determining the *chromatic index* of a graph, i.e., the minimum number of colors required to assign each edge a color such that edges incident to a common vertex receive different colors. The k-DISJOINT MATCHING problem is hence equivalent to finding a maximum-weight subgraph with chromatic index k. Whereas Δ naturally gives a lower bound on the chromatic index, an upper bound is given by $\Delta+1$ [48]. In general, it is $\mathcal{NP}$-hard to decide whether a graph has chromatic index Δ or $\Delta+1$ [49] already if the graph is cubic, i.e., $\deg(v) = 3$ for all $v \in V$:

Theorem 1 (Holyer [49])**.** *It is $\mathcal{NP}$-complete to determine whether the chromatic index of a cubic graph is* 3 *or* 4*.*

Proposition 1. *The k-*DISJOINT MATCHING *problem is $\mathcal{NP}$-hard already in the unweighted case and for $k = 3$.*

Proof. Consider a cubic graph G. There is a one-to-one correspondence between deciding the chromatic index of G and deciding whether $k = 3$ disjoint matchings of weight at least m exist in a graph where $D = 1$: The set of edges of the same color in the 3-coloring give three matchings and three disjoint matchings in G give a 3-coloring of G. □

Recall that for $k = 1$, k-DISJOINT MATCHING is equal to WEIGHTED MATCHING and thus solvable in polynomial time, whereas for $k = 2$ the complexity is still unknown. Computing a maximum weight matching, removing it, and computing a second does not necessarily yield an optimal solution for $k = 2$, as we show in Sect. III. Note that it is easy to tell whether a graph with maximum degree two can be colored with two or three colors: Two colors always suffice unless the graph has an odd-length cycle. We give below various polynomial-time algorithms whose running time is polynomial in n, m, and k (observe that $k \le \Delta + 1$).

In consequence of Theorem 1, EDGE COLORING is inapproximable within a factor less than $4/3$. This implies the following inapproximability result for DJM:

Proposition 2. *It is $\mathcal{NP}$-hard to approximate the k-*DISJOINT MATCHING *problem within a factor of $(1-\epsilon)$ for any $1/m \ge \epsilon > 0$.*

Proof. Consider a cubic graph G with $D = 1$. If G's chromatic index is 3, then every algorithm for k-DJM with an approximation ratio strictly greater than $(1-\frac{1}{m})$ must compute a solution for $k = 3$ that contains more than $m(1-\frac{1}{m}) = m-1$ edges, which implies it must contain m edges. But if it is 4, it can contain at most $m-1$ edges. Thus any such algorithm can be used to decide whether the chromatic index is 3 or 4. □

EDGE COLORING can be solved to optimality in $\mathcal{O}(2.2461^m)$ time by first obtaining the line graph and then applying an algorithm for VERTEX COLORING [50]. Misra and Gries [51] gave an algorithm that constructs a $\Delta+1$ coloring in time $\mathcal{O}(nm)$, whereas an algorithm that greedily colors the edges with the first available color can use up to $2\Delta-1$ colors, which is optimal in the online setting [52].

The r-FACTORIZATION problem asks for a partition of a graph's edge set into a disjoint collection of r-regular spanning subgraphs, called r-factors. Thus, a 1-factor is a perfect matching, and only regular graphs with an even number

TABLE I: Running Time complexities of various algorithms.

Algorithm	Running Time Complexity	Approximation
`Blossom-It`	$\mathcal{O}(kn(m + n \log n))$	$\leq 7/9$
`Greedy-It`	$\mathcal{O}(\text{sort}(m) + km)$	$1/2$
`GPA-It`	$\mathcal{O}(\text{sort}(m) + km)$	$\leq 1/2$
`bGreedy&Extend`	$\mathcal{O}(\text{sort}(m) + kn^2)$	$\leq 1/2$
`NodeCentered`	$\mathcal{O}(\text{sort}(n) + n \cdot \text{sort}(\Delta) + km)$	$\leq 1/2$
`k-EC`	$\mathcal{O}(\text{sort}(m) + kn^2)$	$\leq 1/2$

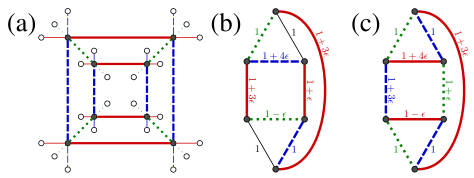

Fig. 1: (a): Hypercube Q_3 with 3 "greedy" matchings (thick) vs. optimum (thin). $\frac{5}{6}$- and $\frac{7}{9}$-approximations for $k = 2$ and $k = 3$, respectively, by `Blossom-It` (b) vs. optimum (c). Matchings have the same color/style, the first is red and thick.

of vertices can have a 1-factorization, which is then equivalent to the EDGE COLORING problem.

III. ALGORITHMS

In the following, we propose and engineer different approaches to obtain disjoint matchings. As the problem is $\mathcal{NP}$-hard and the field of application requires solutions computable within fractions of a second, we concentrate on algorithms from which we expect good, though not necessarily optimal quality. Our approaches are inspired by and partially also built on methods for the related problems of WEIGHTED MATCHING, WEIGHTED b-MATCHING, and EDGE COLORING, and are evaluated on a diverse set of instances in Sect. IV.

Table I provides an overview of all algorithms and lists their time complexity as well as approximation guarantees.

A. Algorithms based on Weighted Matching (*⋆*`-It`)

Given an algorithm $\mathcal{A}_{\text{M}}$ for the weighted matching problem, a straightforward approach to obtain k disjoint matchings of large total weight consists in running $\mathcal{A}_{\text{M}}$ k times and making the set of matching edges "unavailable" to subsequent runs of $\mathcal{A}_{\text{M}}$. As the disjoint matchings are obtained iteratively, we use the suffix -IT for algorithms following this scheme. We study different options for $\mathcal{A}_{\text{M}}$:

`Greedy-It`. A matching that has at least half the weight of a maximum weight matching can be obtained by a greedy algorithm in $\mathcal{O}(\text{sort}(m)) \subseteq \mathcal{O}(m \log n)$ time: Starting from an empty matching, it repeatedly adds the heaviest non-matching edge e and removes all edges incident to one of e's end vertices until the graph is empty. To obtain k disjoint matchings based on this greedy strategy, it suffices to sort the edges according to their weight once and construct the disjoint matchings by iterating k times over the list of sorted edges and removing an edge from the list as soon as it becomes part of a matching. The resulting algorithm `Greedy-It` has a running time of $\mathcal{O}(\text{sort}(m)+km)$. The greedy matching algorithm achieves an approximation ratio of $\frac{1}{2}$, which also transfers to `Greedy-It`:

Lemma 1. *`Greedy-It` computes a $\frac{1}{2}$-approximation to the k-DISJOINT MATCHING problem. This bound is tight.*

Proof. Let $\mathcal{M}$ be the solution computed by `Greedy-It`, let $\mathcal{M}^*$ be an optimal solution, and consider an edge $\{u, v\}$ in $\mathcal{M}^* \setminus \mathcal{M}$. By construction, $\mathcal{M}$ then contains at least k edges that are incident to either u or v and all have weight at least $\mathcal{D}(u, v)$. However, each of the edges in $\mathcal{M} \setminus \mathcal{M}^*$ can have prevented at most two edges in $\mathcal{M}^* \setminus \mathcal{M}$ from being picked themselves, one incident to each of its end nodes. Hence, $\mathcal{D}(\mathcal{M}^*) \leq 2 \cdot \mathcal{D}(\mathcal{M})$. The tightness follows by Lemma 2. ☐

`GPA-It`. The Global Paths Algorithm (`GPA`) is a $\frac{1}{2}$-approximation algorithm for the weighted matching problem introduced by Maue and Sanders [41]. It is especially of interest here as the authors have shown that it produces results that are very close to optimal in experiments, especially if combined with the postprocessing routine `ROMA` (see also the end of this section). `GPA` grows a set of paths and even-length cycles as follows: Initially, every vertex forms a path of zero length. An edge is called *applicable* if it connects two different paths or the two end nodes of the same odd-length path. The algorithm iterates over all edges in weight-descending order and joins or closes paths by applicable edges. Afterwards, it computes an *optimal* weighted matching for each path and even-length cycle via dynamic programming, which takes time linear in the length of the path or cycle. The total running time for `GPA` hence is $\mathcal{O}(\text{sort}(m) + m)$. To use `GPA` as $\mathcal{A}_{\text{M}}$, it suffices again to sort the edges just once and only run the path growing and dynamic programming steps k times, which results in a total running time of $\mathcal{O}(\text{sort}(m) + km)$ for this algorithm, which we refer to as `GPA-It`.

`Blossom-It`. We also evaluate the use of an optimal weighted matching algorithm as subroutine. `Blossom` is the famous algorithm developed by Edmonds originally for the unweighted matching problem [25], which he later extended also to the weighted case [26]. The key idea is to grow alternating-path trees and shrink odd-length cycles (called blossoms) to find augmenting paths, which is guided by a (dual) vertex labelling in the weighted case. Our algorithm `Blossom-It` follows the scheme described above and simply repeats this algorithm k times, which results in a running time of $\mathcal{O}(k \cdot n(m + n \log n))$. `Blossom-It` does not compute an optimal solution to k-DISJOINT MATCHING: As shown in Figs. 1b and 1c, its approximation ratio can be at most $\frac{5}{6}$ for $k = 2$ and $\frac{7}{9}$ for $k = 3$, *also if all edge weights are set to* 1. The solution can be forced by simply setting the weights of the edges in the first matching to $1+\epsilon$ for some small $\epsilon > 0$ or to the weights shown in the figure. Observe that with the shown weights, `Greedy-It` computes an optimal solution for both $k = 2$ and $k = 3$, i.e., *`Blossom-It` is not guaranteed to perform better than `Greedy-It`*. However, `Blossom-It` computes an optimal solution for $k = 1$ and hence trivially computes a $\frac{1}{2}$-approximation for $k = 2$, as the optimum for $k = 2$ can be at most twice as large as the optimum for $k = 1$.

B. Algorithms based on Weighted b-Matching and Coloring

`bGreedy&Extend`. We make use of the fact that every k-disjoint matching is also a k-matching, see also Sect. II-B. Furthermore, the edges of every graph with maximum vertex degree Δ can be partitioned into a set of at most $\Delta + 1$ matchings by coloring its edges, which implies that every $(k-1)$-matching can be translated into a k-disjoint matching without loss of weight.

Analogously to the greedy weighted matching algorithm used in `Greedy-It`, there is a naïve greedy b-matching algorithm that yields a $\frac{1}{2}$-approximation [44]. It iterates over all edges in weight-decreasing order and adds each edge $\{u, v\}$ to the b-matching unless u and v are already incident to b matching edges. The running time of this algorithm is $\mathcal{O}(\text{sort}(m))$. `bGreedy&Extend` first obtains a $(k-1)$-matching and then colors the subgraph induced by the edges of the $(k-1)$-matching with the edge coloring algorithm by Misra and Gries [51], which needs at most k colors and runs in $\mathcal{O}(kn^2)$ time. Note that the induced subgraph has a maximum vertex degree of $k-1$, so the number of edges is in $\mathcal{O}(kn)$ and the algorithm uses at most k colors. The coloring assigns each edge of the subgraph to one of the k disjoint matchings. `bGreedy&Extend` then runs `Greedy-It` to enlarge the k disjoint matchings if possible. The running time of this algorithm hence is $\mathcal{O}(\text{sort}(m) + kn^2)$.

All previously described procedures are based on matching or b-matching algorithms, which do not tackle the problem directly and on the whole, but have a more or less limited view. We therefore complement our set of algorithms by two further approaches that try to find a heavy-weight subgraph with chromatic index k.

`NodeCentered`. The algorithm follows a greedy, node-centered strategy: In a preprocessing step, it calculates a *rating* for each vertex and sorts the vertices according to their rating. We consider different options to obtain a vertex's rating from the weights of its incident edges: the arithmetic mean, the median, the sum, the maximum, as well as the sum of the k largest weights (called `kSUM`). Next, it processes the vertices in rating-decreasing order and tries to color its incident edges in weight-decreasing order. Each color represents one of the k disjoint matchings and the algorithm has to ensure that no vertex is incident to two edges of the same color. Hence, if for an edge $\{u, v\}$ the vertices u and v do not share any common free color, the edge is not picked. The algorithm stores for each vertex and color a Boolean flag whether this color has already been used for an incident edge, such that finding a common free color takes $\mathcal{O}(k)$ time if both end vertices have been matched at most $k-1$ times, and $\mathcal{O}(1)$ otherwise.

To avoid an overly greedy coloring, we introduce a *threshold* $\theta \in [0, 1]$ and ignore all edges with weight less than $\theta \cdot D$, where D is the maximum weight of any edge. In this case, the first phase, in which all vertices are processed as described above, is followed by a second phase, where we merge the sorted lists of the non-matching edges at each vertex into one sorted list and match and color greedily. The running time of the algorithm is $\mathcal{O}(\text{sort}(n) + n \cdot \text{sort}(\Delta) + km)$ and independent of θ. Note that $\theta = 0$ is equivalent to setting no threshold. Irrespective of how the rating function and threshold is chosen, the algorithm computes the same greedy weighted matching as `Greedy-It` for $k = 1$.

`k-EC`. We also designed a k-edge coloring algorithm that uses the algorithm by Misra and Gries (`MG`) [51] directly as basis, but only colors up to k incident edges of each vertex and takes edge weights into account. A key property of the `MG` algorithm with respect to our modification is that once an edge has been colored, it may only be recolored later, but never uncolored. To color an edge $\{u, v\}$, `MG` builds a maximal *fan* around u, which is a sequence $(w_0 = v, w_1, \ldots, w_\ell)$ of distinct neighbors of u such that, for all $1 \leq i \leq \ell$, if the edge $\{u, w_i\}$ has color c, then w_{i-1} is not incident to an edge with color c, i.e., c is *free* on w_{i-1}. `MG` then determines a color c that is free on u and a color d that is free on w_ℓ. If d is not free on u, it looks for a path that starts at u and whose edges are alternatingly colored d and c, and swaps these colors on the path. Afterwards, d is guaranteed to be free on u. The prefix of the fan up to the first neighbor w_j where d is free is then *rotated*, which means that each edge $\{u, w_i\}$ is recolored with the color of $\{u, w_{i+1}\}$, for all $0 \leq i < j$, and $\{u, w_j\}$ is colored with d.

Our adaptation `k-EC` proceeds as follows: It processes the edges in weight-descending order and, similar to `MG`, tries to color each edge $\{u, v\}$, however only with one of up to k colors. The edge is skipped if u or v are already incident to k colored edges. If the last neighbor in the maximal fan around u does not have a free color, `k-EC` tries to color the edge with swapped rules for u and v instead, and skips the edge if also this fails. We consider four flags that modify this basic routine: If `CC` (*common color*) is enabled, `k-EC` tries to find a common free color of u and v first when trying to color $\{u, v\}$. With `LC` (*lightest color*), it tries to balance the total weight of the edges of each color by always picking a free color with minimum total weight so far. If `RL` (*rotate long*) is set and the color d is free on u, it rotates the entire fan instead of determining the first neighbor w_j where d is free. We also consider an option `LF` (*large fan*), where we try to avoid neighbors without a free color as long as possible while constructing the fan. Note that by definition of the fan, a neighbor without free color cannot have a successor.

As there are at most kn edges that can be colored and coloring an edge can be done in $\mathcal{O}(n)$, the running time of the algorithm is $\mathcal{O}(\text{sort}(m) + kn^2)$.

Lemma 2. *The approximation ratio achieved by `Greedy-It`, `GPA-It`, `bGreedy&Extend`, `NodeCentered`, and `k-EC` is at most $\frac{1}{2}$.*

Proof. Let $k \geq 1$ and consider the k-dimensional hypercube graph Q_k, where each vertex additionally has k vertices of degree one attached to it, and set all edge weights equal. A solution returned by one of the mentioned algorithms may consist of all $2^{k-1}k$ hypercube edges, whereas the optimal solution consists of all $2^k \cdot k$ edges incident to the vertices of degree one, see also Fig. 1a. This solution can also be forced by assigning a (very large) weight w to all non-hypercube edges and an only slightly larger one $w+\epsilon$ to the hypercube edges. The approximation ratio is then $\frac{w+\epsilon}{2w} = \frac{1}{2} + \frac{\epsilon}{2w}$. $\square$

C. Postprocessing

We consider different postprocessing techniques to improve our algorithms: ROMA (*Random Order Matching Augmentation*) was originally proposed for GPA by Maue and Sanders [41] to improve the weight of a matching $\mathcal{M}$. For a configurable number l_{ROMA} of times, it randomly iterates through all vertices and for each matched vertex u attempts to improve $\mathcal{M}$ by a so-called maximum-gain 2-augmentation: A matching edge $\{u, v\} \in \mathcal{M}$ is replaced by two non-matching edges $\{u, r\}$ and $\{v, s\}$ if r and s are currently unmatched and the gain $\mathcal{D}(u, r) + \mathcal{D}(v, s) - \mathcal{D}(u, v)$ is positive and as large as possible. The procedure can be terminated early as soon as one iteration leads to no change in the matching. We adapt this approach straightforwardly for the k-DJM problem by calling the procedure for each of the k disjoint matchings separately right after they have been obtained. We also consider a variation Swaps, where we instead iterate once over all matching edges in weight-decreasing order and perform the same maximum-gain 2-augmentation as in ROMA.

For Swaps as well as each iteration of ROMA, we iterate over all matching edges and for each of its endpoints explore all neighbors to find the heaviest incident free edge. Each matching edge is hence considered once and each non-matching edge at most twice, which yields a running time complexity of $\mathcal{O}(m)$ in case of Swaps under the assumption that the edges are already sorted.

IV. Experiments

We performed extensive experiments to evaluate the performance of the algorithms described in Sect. III both with respect to solution quality and running time. To keep large numbers readable, we use K and M as abbreviations for thousands ($\times 10^3$) and millions ($\times 10^6$), respectively.

A. Instances, Setup, and Methodology

The first three collections we use originate from the application side and contain real-world as well as synthetic instances, the other two have been used in previous experimental evaluations for b-matchings [46]. **Facebook Data Traces** [2] are sets of production-level traces from three different clusters in Facebook's Altoona Data Center. For each cluster, there are over 300M traces, collected over a 24 h period. The resulting six instances have 14K to 5M vertices and 497K to 164M edges with demands between 28 and 262M. **HPC** represents four different applications run in parallel using MPI [16]. The instances have 1K vertices and up to 38K edges; the demands are between 2 and 1K. We also use three synthetic **pFabric** traces [16], [53], which have 144 vertices and are generated based on flows arriving according to a Poisson process, with flow rates in $\{0.1, 0.5, 0.8\}$. This results in about 10K edges and demands between 1 and 34K. Following the methodology of Khan et al. [46], we include nine instances from the **Florida Sparse Matrix Collection** [54]. These instances stem from collaboration networks, medical science, news networks, as well as sensor data, electro magnetics, and structural mechanics. They have 13K to 1M vertices and 121K to 24M edges, with demands between 1 and 2 147M. Following [46], we also generated 66 Kronecker instances using the Graph500 **RMAT** generator with 2^x vertices, $10 \leq x \leq 20$, and initiator matrices *rmat_b* with $(0.55, 0.15, 0.15, 0.15)$, *rmat_g* with $(0.45, 0.15, 0.15, 0.25)$, and *rmat_er* with $(0.25, 0.25, 0.25, 0.25)$. Demands are chosen according to a uniform (_uni) or exponential distribution (_exp) and range between 1 and half a million.

We implemented our algorithms in C++17 and compiled using GCC 7.5 and full optimization(-O3 -march=native -mtune=native). For Blossom-It, we adapted an implementation of the Blossom algorithm from the Lemon[1] library and made it iterative by calling the algorithm k times, after each round setting weights of matched edges to 0. We only report results for the variant where Blossom starts with an approximate matching obtained from a fractional solution instead of an empty matching, as both compute optimal weighted matchings, but the fractional option was considerably faster, requiring only half the running time in the geometric mean across different data sets.

To determine the optimal weight and use it in quality comparisons, we also implemented an exact algorithm by casting the problem as an ILP, using an adaptation of the assignment formulation for the edge coloring problem [55], and solved it with Gurobi[2]. Unfortunately, it only terminated within the timeout of 4 h for small instances and values of k.

All experiments were performed on a machine with NUMA architecture running Ubuntu 18.04 with Intel(R) Xeon(R) CPUs clocked at 3.40 GHz and 256 kByte and 20 MByte of L2 and L3 cache, respectively. The execution of each experiment was pinned to a single CPU and its local memory to prevent the cost of non-local memory accesses or swapping. To counteract artifacts of measurement in running time, we ran each experiment three times and use the median of the *elapsed real time* (wall time). The only exception to this rule is the ILP, which was run just once as we were mainly interested in the solution size. We set a timeout of 4 h.

B. Experimental Results

We performed experiments for $k \in \{2, 4, 8, 16, 32, 64, 96\}$. Our set of algorithms contained (1) NodeCentered in 15 configurations: with threshold $\theta \in \{0, 0.2, 0.5\}$ and vertex-ratings MAX, AVG, MEDIAN, SUM, and kSUM; (2) Greedy-It with and without Swaps; (3) GPA-It with and without Swaps and additionally with the postprocessing step ROMA ($l_{\text{ROMA}} = 4$) after each iteration, (4) Blossom-It,

[1] https://lemon.cs.elte.hu/trac/lemon
[2] http://www.gurobi.com

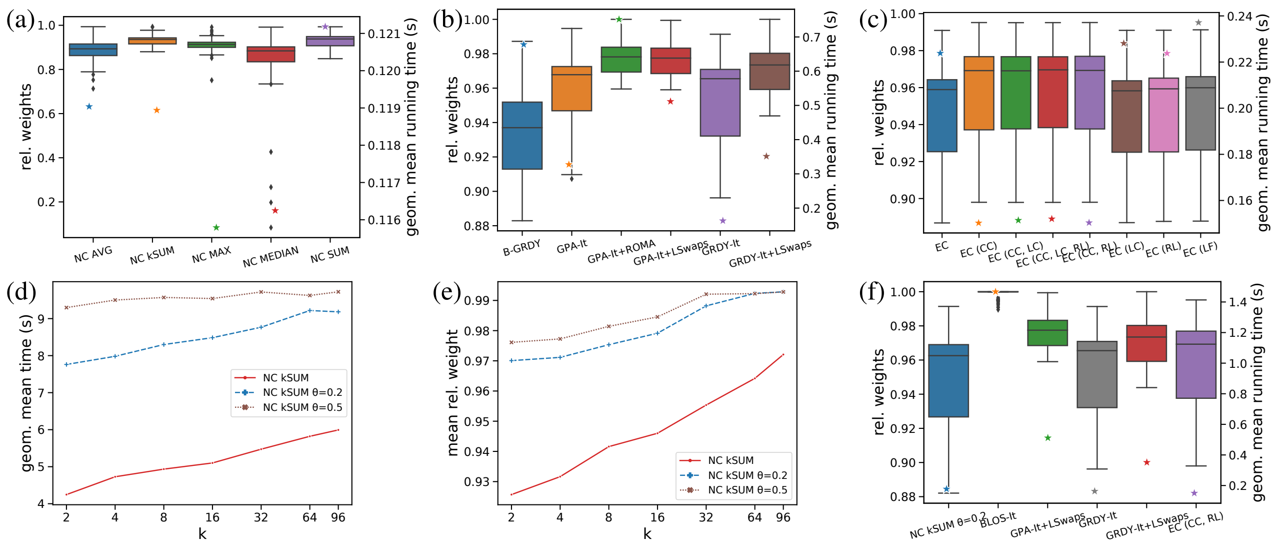

Fig. 2: Result quality (left axis) and running time (right axis, depicted as star) for `NodeCentered` with $\theta = 0$ and different aggregation functions (a), `bGreedy&Extend`, `GPA-It`, and `Greedy-It` with and without postprocessing (b), `k-EC` (c), and for the set of the best algorithms (f), in each case for $k = 4$ and all instance sets. Running time (d) and result quality (e) for `NodeCentered` with `kSUM` and different thresholds on `Facebook`.

as well as (5) `bGreedy&Extend` and (6) `k-EC`. When `Swaps` were used, they were either applied after each iteration (`LocalSwaps`) or once after all iterations finished (`GlobalSwaps`).

Intuitively, we would expect that it becomes “easier” for the algorithms to add high-demand edges to one of the matchings as k increases and, thus, that all algorithms should return an almost equally good solution when $k = 96$. This is also confirmed by our results. Still we can show interesting differences between the algorithms that we will describe in this section. We proceed as follows: We first look at the behavior of similar or the same algorithm with different configurations, and then compare it to other algorithms using only the best variant. Relative solution weights are expressed as a fraction of the optimum (OPT) or, if the optimum is unknown, the best that *any* algorithm has found (BEST). Note that all plots use a logarithmic axis for k.

`NodeCentered`: We first consider for each set of instances the relative weights and mean running times for `NodeCentered` with thresholds 0, 0.2, and 0.5, respectively, for the five different aggregation functions. As the threshold effectively limits the number of edges colored in the first phase and the aggregation function does not play a role in the second phase, we observe as expected that the differences in quality when using different aggregation functions become smaller the larger the threshold θ. In general, `MEDIAN` led to worse performance than the other aggregation methods, especially on the `Facebook` instance set with no threshold ($\theta = 0$), where it achieved, e.g., for $k = 4$ a solution quality of only $8\,\%$ on `clusterC-racks`. This behavior can be explained by the strongly biased demand distribution. `SUM` results in a higher rating of vertices with many (low-demand) edges, whereas `AVG` also takes a vertex's degree into account, which is however detrimental for small k and a skewed demand distribution. `MAX` can be led astray if vertices have a single edge with very high demand, but many others with low demand, which resulted in bad performance especially on the `pFabric` instances. Overall, `kSUM` showed the *best and most stable performance* in most cases, see also Fig. 2a.

We observe that large thresholds incur a larger time cost, as more edges are left unprocessed in the first phase and need to be reconsidered in the second phase. As an example, Fig. 2d shows the running times for the `Facebook` instances with `kSUM` as aggregation function for the different thresholds; the behavior on the other instances is similar. In the worst case, the variants where $\theta > 0$ ran more than twice as long as without threshold (125 s vs. 260 s (268 s) for $k = 4$ on `clusterB-ips` with $\theta = 0.2$ ($\theta = 0.5$)). In the geometric mean over all instances, $\theta = 0.2$ and $\theta = 0.5$ led to a slowdown by a factor between 1.5 and 2 in comparison to setting no threshold. Looking at the result quality (Fig. 2e), we see that thresholds are effective in avoiding overly greedy matching, as intended. The quality differences between $\theta = 0$ and $\theta = 0.2$ are “only” $4\,\%$ to $2\,\%$ for `Facebook` instances, e.g., which however corresponds to an average absolute gain or loss of 1 to 5 billion due to the large absolute values. Setting $\theta = 0.5$ increases the quality only marginally, but comes with an increased running time, which is why we consider *`kSUM` and $\theta = 0.2$ as the best configuration for `NodeCentered`.*

`GPA-It`, `Greedy-It`, `bGreedy&Extend`: As an example, Fig. 2b shows the result quality and running time for $k = 4$. We can observe a boost in quality for `GPA-It` when activating either `LocalSwaps` or `ROMA`, but no improvement with `GlobalSwaps` (omitted in the plot). Comparing

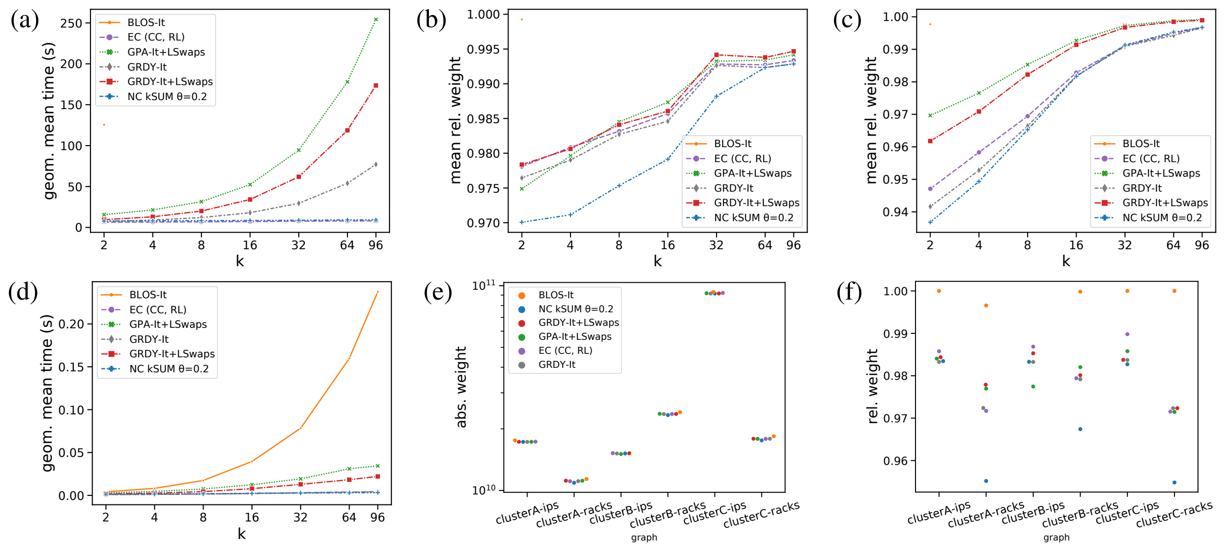

Fig. 3: Mean running time (a) and result quality (b) of the best algorithms on `Facebook`, mean result quality on all 88 instances (c), and mean running time on `HPC` and `pFabric` (d). (e, f): Per-instance absolute and relative weights on `Facebook`.

`LocalSwaps` and `ROMA`, we obtain almost equal result quality at distinctly faster speed with `LocalSwaps`. The running time with `GlobalSwaps` is similar to plain `GPA-It` and faster than with `LocalSwaps` by a factor of two with $k = 96$. As `GlobalSwaps` has almost equal quality as `GPA-It` this suggests that no or only very few swaps were performed. Given the trade-off between quality and running time, we consider *`GPA-It` with `LocalSwaps`* to be the better option, which we will use in our further analysis.

Similarly, we evaluated `Greedy-It` with local and global `Swaps`, compared to a base version without swaps. Again, we observe barely any improvement in quality by `GlobalSwaps`, yet `LocalSwaps` consistently yields results with better quality, at the expense of an increased running time. To consider both ends of the result quality vs. running time tradeoff, we include both `Greedy-It` alone as well as `Greedy-It` with `LocalSwaps` in our further analysis.

`bGreedy&Extend` was inferior to `Greedy-It` on all sets of instances both with respect to running time and solution quality and is therefore not considered further.

k-EC: Fig. 2c shows the result quality and running time for `k-EC` with different combinations of flags for $k = 4$ and all instances. As expected, `CC` (common color) decreases the running time, here by over 30 %, as fan construction and rotation are no longer required in many cases. It increases the result quality distinctly as it can also color an edge if the last neighbor in the fan does not have a free color. To the contrary, `LC` (lightest color) leads to a clearly visible decline without `CC` and in general to a slight increase in running time due to the additional maintenance of color weights. `RL` (rotate long) marginally improves result quality and has a negligible effect on the running time, whereas `LF` leads to a slowdown in general and slightly better quality only if `CC` is not set. We thus consider `CC` and `RL` as the best parameters for `k-EC`.

C. Overall Running Times and Result Quality

Given our choice of representatives for each algorithm, we analyze these representatives regarding their running time and result quality in detail on the instance set `Facebook` and only give a summary about the others. *We do not discuss the other instance sets in detail any further, as the algorithms perform very consistently on all of them.* Note that a given algorithm is only represented for a given k if that algorithm finished on *all* instances of a set within our 4 h time limit.

Looking purely at the running time complexities (cf. Table I), one might expect to see `GPA-It` and `Greedy-It` behaving similarly to `NodeCentered`. The former two have a slightly larger preprocessing time, yet afterwards all perform $\mathcal{O}(km)$ work (with and without `LocalSwaps`). `k-EC`, on the other hand, has both large preprocessing time and performs $\mathcal{O}(kn^2)$ work, so it could be expected to be the slowest. However, Figs. 3a and 3d paint a vastly different picture, as `NodeCentered` and `k-EC` compute the disjoint matchings significantly faster than the `⋆-It` algorithms. This can be observed consistently on all instances. For the `Facebook` instances and $k = 4$, `k-EC` achieves in the geometric mean a speedup of 2 and 3.2 over `Greedy-It` with `LocalSwaps` and `GPA-It`, respectively, and for $k = 96$ the speedups increase to 21.6 and 31.7. The running time of `Greedy-It` without `LocalSwaps` is less than the time for `Greedy-It` with `LocalSwaps`, but larger than for `k-EC`. `NodeCentered` is equally fast as `k-EC`. Over all instances and values of k, `k-EC` is faster than `Greedy-It` and `GPA-It` with `LocalSwaps` by a factor of 1.8 to 7.3 and 2.6 to 9.5, respectively. `Blossom-It` terminated on all instances in `HPC` and `pFabric`, but was 4.5 to 57.9 times slower than `k-EC`. The speedup by `k-EC` over the plain variant of `Greedy-It` is less pronounced, but still between 1.2 and 9.6 for `Facebook` and up to 2, e.g., on Florida. The reason that Greedy-It without LocalSwaps is faster than Greedy-It with LocalSwaps is that LocalSwaps prevents the algorithm from efficiently cutting down the list of edges to process in the next each iteration: As LocalSwaps changes the matching, the non-matching edges need either be sorted after each iteration or all edges are processed in each iteration, causing $\Theta(k)$ work per edge. NodeCentered and k-EC, on the other hand, operate more locally. NodeCentered scans each edge at most three times and *only if* both end vertices have been matched less than k times so far, it compares two lists of Boolean arrays of length k to determine a common free color. Thus, the work per edge is often just constant. The situation is similar for k-EC.

Regarding quality (Figs. 3b, c), for Facebook instances and $k \leq 4$, k-EC and Greedy-It with LocalSwaps perform best. For $k \geq 8$, k-EC stays slightly behind Greedy-It and GPA-It with LocalSwaps by less than $0.01\,\%$ (regarding the mean of weights relative to BEST). The mean performance of NodeCentered always remains within $1\,\%$ of BEST. Across *all instances*, GPA-It with LocalSwaps performed best, with a mean relative weight of at least $97\,\%$ of BEST, closely followed by Greedy-It with LocalSwaps and k-EC with at least $96\,\%$ and $94.7\,\%$ on average. NodeCentered performed worst, however still within $93.7\,\%$ of BEST on average for $k = 2$ and $99.7\,\%$ for $k = 96$. If we look at the *worst performance* per algorithm across all instances, we observe a quality ratio of at least $95\,\%$ for GPA-It, $93\,\%$ for Greedy-It with LocalSwaps, $87\,\%$ ($90\,\%$ for $k \geq 4$) for Greedy-It and k-EC, and $76\,\%$ ($88\,\%$ for $k \geq 4$) for NodeCentered, see also Fig. 2f.

Figs. 3e and 3f show absolute and relative per-instance weight comparisons for $k = 4$ on the Facebook instances. We can clearly observe that NodeCentered struggles with the rack-level instances clusterA-racks, clusterB-racks, and clusterC-racks. k-EC is second-best after Blossom-It on the IP-level instances.

Blossom-It finished within the $4\,$h time limit on $97\,\%$ of all experiments. Its asymptotic running time has an additive factor of $\mathcal{O}(knm)$ as compared to $\mathcal{O}(km)$ for the other algorithms and this is confirmed by our experiments: It is the slowest on all graphs. However, it always achieves the best quality results and for all graphs where the ILP terminated, Blossom-It was within $99\,\%$ of the result quality of the optimum. However, for $k \geq 64$ the faster algorithms achieved almost the same result quality. Thus, Blossom-It is a good choice only for small values of k and in settings where running time is not a limiting factor.

The ILP completed on all HPC instances for $k \in [2, 16]$, all pFabric instances for $k \in [2, 8]$, most RMAT instances with $n = 2^{10} \ldots 2^{12}$ for $k \in [2, 16]$ (rmat_er, rmat_g_12 only for $k \in \{2, 4\}$), as well as on three Facebook instances (clusterA-racks and clusterB-racks for $k \in \{2, 4\}$ and clusterC-racks only for $k = 2$).

The order of the algorithms with respect to running time and result quality is consistent on all instances except for pFabric, where k-EC on average finds larger solutions than Greedy-It with LocalSwaps and partially also GPA-It for all values of k.

Overall we conclude that for medium and large values of k, k-EC with CC and RL enabled is the best-performing algorithm. Unlike the running time of the $\star$-It algorithms, its running time barely increases with k and its quality score is *on average* within $95\,\%$ or more of the best algorithm within the $4\,$h time limit, and $98\,\%$ or better for $k \geq 32$. *It is also one of the best algorithms for small values of k on the Facebook and pFabric instances. On the other instances, GPA-It with LocalSwaps is a good choice for small values of k, as its quality is always within $95\,\%$ of the best algorithm while its running time is still moderate; if running time is not of concern, Blossom-It is a better choice.*

V. Future Work

There remain several interesting avenues for future work. In particular, it would be interesting to further explore the power of randomized algorithms. The only randomized algorithm we analyzed is GPA-It with ROMA, but it did not show the strongest performance. On the practical front, it will be interesting to deploy and experiment with our algorithms in a small datacenter network using optical circuit switches.

The authors have provided public access to their code at https://doi.org/10.5281/zenodo.5851268.

References

[1] A. Singh, J. Ong, A. Agarwal, G. Anderson, A. Armistead, R. Bannon, S. Boving, G. Desai, B. Felderman, P. Germano *et al.*, "Jupiter rising: A decade of clos topologies and centralized control in google's datacenter network," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, pp. 183–197, 2015.

[2] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren, "Inside the social network's (datacenter) network," *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, 2015.

[3] A. Valadarsky, G. Shahaf, M. Dinitz, and M. Schapira, "Xpander: Towards optimal-performance datacenters," in *Proceedings of the 12th International on Conference on emerging Networking EXperiments and Technologies*, 2016, pp. 205–219.

[4] 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 *Proceedings of the ACM SIGCOMM 2009 conference on Data communication*, 2009, pp. 63–74.

[5] A. Singla, C. Hong, L. Popa, and P. B. Godfrey, "Jellyfish: Networking data centers randomly," in *Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2012, San Jose, CA, USA, April 25-27, 2012*, S. D. Gribble and D. Katabi, Eds. USENIX Association, 2012, pp. 225–238. [Online]. Available: https://www.usenix.org/conference/nsdi12/technical-sessions/presentation/singla

[6] M. N. Hall, K.-T. Foerster, S. Schmid, and R. Durairajan, "A survey of reconfigurable optical networks," in *Optical Switching and Networking (OSN), Elsevier*, 2021.

[7] M. Zhang, J. Zhang, R. Wang, R. Govindan, J. C. Mogul, and A. Vahdat, "Gemini: Practical reconfigurable datacenter networks with topology and traffic engineering," *CoRR*, vol. abs/2110.08374, 2021. [Online]. Available: http://arxiv.org/abs/2110.08374

[8] C. Griner, J. Zerwas, A. Blenk, S. Schmid, M. Ghobadi, and C. Avin, "Cerberus: The power of choices in datacenter topology design (a throughput perspective)," in *Proc. ACM SIGMETRICS*, 2022.

[9] 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 *ACM SIGCOMM 2016, Florianopolis, Brazil, August 22-26, 2016*, M. P. Barcellos, J. Crowcroft, A. Vahdat, and S. Katti, Eds. ACM, 2016, pp. 216–229. [Online]. Available: https://doi.org/10.1145/2934872.2934911

[10] 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 *ACM SIGCOMM 2017, Los Angeles, CA, USA, August 21-25, 2017*. ACM, 2017, pp. 267–280. [Online]. Available: https://doi.org/10.1145/3098822.3098838

[11] 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," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 4, pp. 339–350, 2011.

[12] G. Wang, D. G. Andersen, M. Kaminsky, K. Papagiannaki, T. E. Ng, M. Kozuch, and M. Ryan, "c-through: Part-time optics in data centers," in *Proceedings of the ACM SIGCOMM 2010 Conference*, 2010.

[13] K. Chen, A. Singla, A. Singh, K. Ramachandran, L. Xu, Y. Zhang, X. Wen, and Y. Chen, "Osa: An optical switching architecture for data center networks with unprecedented flexibility," *IEEE/ACM Transactions on Networking*, vol. 22, no. 2, pp. 498–511, April 2014.

[14] H. Ballani, P. Costa, R. Behrendt, D. Cletheroe, I. Haller, K. Jozwik, F. Karinou, S. Lange, K. Shi, B. Thomsen, and H. Williams, "Sirius: A flat datacenter network with nanosecond optical switching," in *ACM SIGCOMM 2020, USA, August 10-14, 2020*, H. Schulzrinne and V. Misra, Eds. ACM, 2020, pp. 782–797. [Online]. Available: https://doi.org/10.1145/3387514.3406221

[15] K.-T. Foerster, M. Ghobadi, and S. Schmid, "Characterizing the algorithmic complexity of reconfigurable data center architectures," in *Proc. ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, 2018.

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

[17] D. Amir, T. Wilson, V. Shrivastav, H. Weatherspoon, R. Kleinberg, and R. Agarwal, "Optimal oblivious reconfigurable networks," *CoRR*, vol. abs/2111.08780, 2021. [Online]. Available: http://arxiv.org/abs/2111.08780

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

[19] 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," *Inf. Sci.*, vol. 422, pp. 282–289, 2018.

[20] A. Meyerson and B. Tagiku, "Minimizing average shortest path distances via shortcut edge addition," in *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*. Springer, 2009, pp. 272–285.

[21] M. Papagelis, F. Bonchi, and A. Gionis, "Suggesting ghost edges for a smaller world," in *Proceedings of the 20th ACM international conference on Information and knowledge management*, 2011, pp. 2305–2308.

[22] N. Parotsidis, E. Pitoura, and P. Tsaparas, "Selecting shortcuts for a smaller world," in *Proceedings of the 2015 SIAM International Conference on Data Mining*. SIAM, 2015, pp. 28–36.

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

[24] E. D. Demaine and M. Zadimoghaddam, "Minimizing the diameter of a network using shortcut edges," in *SWAT 2010*. Springer, 2010, pp. 420–431.

[25] J. Edmonds, "Paths, trees, and flowers," *Canadian Journal of Mathematics*, vol. 17, pp. 449–467, 1965.

[26] ——, "Maximum matching and a polyhedron with 0,1-vertices," *Journal of Research of the National Bureau of Standards Section B Mathematics and Mathematical Physics*, p. 125, 1965.

[27] H. N. Gabow, "Implementation of algorithms for maximum matching on nonbipartite graphs," Ph.D. dissertation, Stanford University, 1974.

[28] ——, "An efficient implementation of Edmonds' algorithm for maximum matching on graphs," *J. ACM*, vol. 23, no. 2, pp. 221–234, 1976. [Online]. Available: https://doi.org/10.1145/321941.321942

[29] E. L. Lawler, *Combinatorial Optimization: Networks and Matroids*. Holt, Rinehart and Winston, New York, 1976.

[30] Z. Galil, S. Micali, and H. N. Gabow, "An $O(EV \log V)$ algorithm for finding a maximal weighted matching in general graphs," *SIAM J. Comput.*, vol. 15, no. 1, pp. 120–130, 1986. [Online]. Available: https://doi.org/10.1137/0215009

[31] H. N. Gabow, "Scaling algorithms for network problems," *J. Comput. Syst. Sci.*, vol. 31, no. 2, pp. 148–168, 1985. [Online]. Available: https://doi.org/10.1016/0022-0000(85)90039-X

[32] H. N. Gabow, Z. Galil, and T. H. Spencer, "Efficient implementation of graph algorithms using contraction," *J. ACM*, vol. 36, no. 3, pp. 540–572, 1989. [Online]. Available: https://doi.org/10.1145/65950.65954

[33] H. N. Gabow, "Data structures for weighted matching and nearest common ancestors with linking," in *SODA 1990, 22-24 January, San Francisco, CA, USA*, D. S. Johnson, Ed. SIAM, 1990, pp. 434–443. [Online]. Available: http://dl.acm.org/citation.cfm?id=320176.320229

[34] H. N. Gabow and R. E. Tarjan, "Faster scaling algorithms for general graph-matching problems," *J. ACM*, vol. 38, no. 4, pp. 815–853, 1991. [Online]. Available: https://doi.org/10.1145/115234.115366

[35] M. Cygan, H. N. Gabow, and P. Sankowski, "Algorithmic applications of baur-strassen's theorem: Shortest cycles, diameter, and matchings," *J. ACM*, vol. 62, no. 4, pp. 28:1–28:30, 2015. [Online]. Available: https://doi.org/10.1145/2736283

[36] F. Le Gall, "Powers of tensors and fast matrix multiplication," in *ISSAC '14, Kobe, Japan, July 23-25, 2014*, K. Nabeshima, K. Nagasaka, F. Winkler, and Á. Szántó, Eds. ACM, 2014, pp. 296–303. [Online]. Available: https://doi.org/10.1145/2608628.2608664

[37] R. Duan and S. Pettie, "Linear-time approximation for maximum weight matching," *J. ACM*, vol. 61, no. 1, pp. 1:1–1:23, 2014. [Online]. Available: https://doi.org/10.1145/2529989

[38] D. E. Drake and S. Hougardy, "Linear time local improvements for weighted matchings in graphs," in *WEA 2003, Ascona, Switzerland, May 26-28, 2003*, ser. Lecture Notes in Computer Science, K. Jansen, M. Margraf, M. Mastrolilli, and J. D. P. Rolim, Eds., vol. 2647. Springer, 2003, pp. 107–119. [Online]. Available: https://doi.org/10.1007/3-540-44867-5_9

[39] R. Preis, "Linear time 1/2-approximation algorithm for maximum weighted matching in general graphs," in *STACS 1999, Trier, Germany, March 4-6, 1999*, ser. Lecture Notes in Computer Science, C. Meinel and S. Tison, Eds., vol. 1563. Springer, 1999, pp. 259–269. [Online]. Available: https://doi.org/10.1007/3-540-49116-3_24

[40] D. E. Drake and S. Hougardy, "A simple approximation algorithm for the weighted matching problem," *Inf. Process. Lett.*, vol. 85, no. 4, pp. 211–213, 2003. [Online]. Available: https://doi.org/10.1016/S0020-0190(02)00393-9

[41] J. Maue and P. Sanders, "Engineering algorithms for approximate weighted matching," in *WEA 2007, Rome, Italy, June 6-8, 2007*, ser. Lecture Notes in Computer Science, C. Demetrescu, Ed., vol. 4525. Springer, 2007, pp. 242–255. [Online]. Available: https://doi.org/10.1007/978-3-540-72845-0_19

[42] V. Kolmogorov, "Blossom V: a new implementation of a minimum cost perfect matching algorithm," *Mathematical Programming Computation*, vol. 1, no. 1, pp. 43–67, 2009.

[43] H. N. Gabow, "Data structures for weighted matching and extensions to b-matching and f-factors," *ACM Trans. Algorithms*, vol. 14, no. 3, pp. 39:1–39:80, 2018. [Online]. Available: https://doi.org/10.1145/3183369

[44] J. Mestre, "Greedy in approximation algorithms," in *ESA 2006, September 11-13, 2006, Zurich, Switzerland*, ser. Lecture Notes in Computer Science, Y. Azar and T. Erlebach, Eds., vol. 4168. Springer, 2006, pp. 528–539. [Online]. Available: https://doi.org/10.1007/11841036_48

[45] D. Huang and S. Pettie, "Approximate generalized matching: f-factors and f-edge covers," *CoRR*, vol. abs/1706.05761, 2017. [Online]. Available: http://arxiv.org/abs/1706.05761

[46] A. M. Khan, A. Pothen, M. M. A. Patwary, N. R. Satish, N. Sundaram, F. Manne, M. Halappanavar, and P. Dubey, "Efficient approximation algorithms for weighted b-matching," *SIAM J. Sci. Comput.*, vol. 38, no. 5, 2016. [Online]. Available: https://doi.org/10.1137/15M1026304

[47] M. Bienkowski, D. Fuchssteiner, J. Marcinkowski, and S. Schmid, "Online dynamic b-matching: With applications to reconfigurable datacenter networks," *SIGMETRICS Perform. Evaluation Rev.*, vol. 48, no. 3, pp. 99–108, 2020. [Online]. Available: https://doi.org/10.1145/3453953.3453976

[48] V. G. Vizing, "On an estimate of the chromatic class of a p-graph," *Discret. Analiz.*, vol. 3, pp. 25–30, 1964.

[49] I. Holyer, "The NP-completeness of edge-coloring," *SIAM J. Comput.*, vol. 10, no. 4, pp. 718–720, 1981. [Online]. Available: https://doi.org/10.1137/0210055

[50] A. Björklund, T. Husfeldt, and M. Koivisto, "Set partitioning via inclusion-exclusion," *SIAM J. Comput.*, vol. 39, no. 2, pp. 546–563, 2009. [Online]. Available: https://doi.org/10.1137/070683933

[51] J. Misra and D. Gries, "A constructive proof of Vizing's theorem," *Inf. Process. Lett.*, vol. 41, no. 3, pp. 131–133, 1992. [Online]. Available: https://doi.org/10.1016/0020-0190(92)90041-S

[52] A. Bar-Noy, R. Motwani, and J. Naor, "The greedy algorithm is optimal for on-line edge coloring," *Inf. Process. Lett.*, vol. 44, no. 5, pp. 251–253, 1992. [Online]. Available: https://doi.org/10.1016/0020-0190(92)90209-E

[53] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. McKeown, B. Prabhakar, and S. Shenker, "pfabric: minimal near-optimal datacenter transport," *ACM SIGCOMM 2013*, 2013.

[54] T. A. Davis and Y. Hu, "The university of florida sparse matrix collection," *ACM Trans. Math. Softw.*, vol. 38, no. 1, Dec. 2011. [Online]. Available: https://doi.org/10.1145/2049662.2049663

[55] A. Jabrayilov and P. Mutzel, "New integer linear programming models for the vertex coloring problem," in *Proc. Latin American Symposium on Theoretical Informatics (LATIN)*, M. A. Bender, M. Farach-Colton, and M. A. Mosteiro, Eds., 2018.