Data Reduction for Maximum Matching on Real-World Graphs: Theory and Experiments

Viatcheslav Korenwein
TU Berlin, Institut für Softwaretechnik und Theoretische Informatik, Berlin, Germany

André Nichterlein
TU Berlin, Institut für Softwaretechnik und Theoretische Informatik, Berlin, Germany
andre.nichterlein@tu-berlin.de

Rolf Niedermeier
TU Berlin, Institut für Softwaretechnik und Theoretische Informatik, Berlin, Germany
rolf.niedermeier@tu-berlin.de

Philipp Zschoche
TU Berlin, Institut für Softwaretechnik und Theoretische Informatik, Berlin, Germany
zschoche@tu-berlin.de

Abstract

Finding a maximum-cardinality or maximum-weight matching in (edge-weighted) undirected graphs is among the most prominent problems of algorithmic graph theory. For n-vertex and m-edge graphs, the best known algorithms run in $\widetilde{O}(m\sqrt{n})$ time. We build on recent theoretical work focusing on linear-time data reduction rules for finding maximum-cardinality matchings and complement the theoretical results by presenting and analyzing (thereby employing the kernelization methodology of parameterized complexity analysis) linear-time data reduction rules for the positive-integer-weighted case. Moreover, we experimentally demonstrate that these data reduction rules provide significant speedups of the state-of-the art implementation for computing matchings in real-world graphs: the average speedup is 3800% in the unweighted case and "just" 30% in the weighted case.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis

Keywords and phrases Maximum-cardinality matching, maximum-weight matching, linear-time algorithms, preprocessing, kernelization, parameterized complexity analysis

Digital Object Identifier 10.4230/LIPIcs.ESA.2018.53

Related Version `http://arxiv.org/abs/1806.09683`

Acknowledgements We are grateful to anonymous reviewers of *ESA '18* for constructive and detailed feedback.

1 Introduction

In their book chapter on weighted matching, Korte and Vygen [11] write that "weighted matching appears to be one of the 'hardest' combinatorial optimization problems that can be solved in polynomial time". Correspondingly, the design and analysis of matching algorithms plays a pivotal role in algorithm theory as well as in practical computing. Complementing the rich literature on matching algorithms (see Coudert, Ducoffe, and Popa [6] and Duan, Pettie, and Su [8] for recent accounts, the latter also providing a literature overview), in this work we focus on efficient linear-time data reduction rules that may help to speed up superlinear matching algorithms. Notably, while recent breakthrough results on matching (including linear-time approximation algorithms [7]) focus on the theory side, we study both theory and practice, thereby achieving gains on both sides.

To achieve our results, we follow and significantly extend recent purely theoretical work [13] presenting and analyzing linear-time data reductions for the unweighted case. More specifically, on the theoretical side we complement these results by performing an analysis for the weighted case (turning out to become more technical); on the practical side, we demonstrate that these data reduction rules may serve to speed up the state-of-the-art matching solver due to Kolmogorov [10]. Similar data reduction rules have been implemented for finding maximum independent sets and minimum vertex covers [2, 5], leading to significant speedups of algorithms for both problems.

Formally, we study the following two problems; note that we formulate them as decision problems since this better fits with presenting our theoretical part where we prove kernelization results (thereby employing the framework of parameterized complexity analysis); our data reduction rules also work and are implemented for the optimization versions.

Maximum-Cardinality Matching
Input: An undirected graph $G = (V, E)$ and $s \in \mathbb{N}$.
Question: Is there a size-s subset $M \subseteq E$ of nonoverlapping (that is, pairwise vertex-disjoint) edges?

Maximum-Weight Matching
Input: An undirected graph $G = (V, E)$, edge weights $\omega\colon E \to \mathbb{N}$, and $s \in \mathbb{N}$.
Question: Is there a subset $M \subseteq E$ of nonoverlapping edges of weight $\sum_{e \in M} \omega(e) \geq s$?

We remark that all our results extend to the case of rational weights; however, integers are easier to cope with and are used in the implementation of Kolmogorov [10].

Our contributions. We lift known kernelization results [13] for Maximum-Cardinality Matching to Maximum-Weight Matching. To this end, we provide algorithms to efficiently apply our newly developed data reduction rules. Herein, we have a particular eye on exhaustively applying the data reduction rules in linear time, which seems imperative in an effort to practically improve matching algorithms. Hence, our main theoretical contribution lies in developing efficient algorithms implementing the data reduction rules, thereby also showing a purely theoretical guarantee on the amount of data reduction that can be achieved in the worst case (this is also known as kernelization in parameterized algorithmics)[1]. We proceed by implementing and testing the data reduction algorithms for Maximum-Cardinality Matching and Maximum-Weight Matching, thereby demonstrating their significant practical effectiveness. More specifically, combining them in form of preprocessing with Kolmogorov's state-of-the-art solver [10, 16] yield partially tremendous speedups on sparse real-world graphs (taken from the SNP library [12]). Concretely, comparing Kolmogorov's algorithm with and without our data reduction algorithms, the average speedup is 3800% in the unweighted case and "only" 30% in the weighted case.

Notation. We use standard notation from graph theory. All graphs considered in this work are simple and undirected. For a graph G, we denote with $E(G) = E$ the edge set.

[1] We clearly remark, however, that our theoretical findings (that is, kernel size upper bounds based on the feedback edge set number) are too weak in order to fully explain the practical success of the data reduction rules.

We write uv to denote the edge $\{u, v\}$ and $G - v$ to denote the graph obtained from G by removing v and all its incident edges. A feedback edge set of a graph G is a set X of edges such that $G - X$ is a tree or forest. The feedback edge number denotes the size of a minimum feedback edge set. A *matching* in a graph is a set of pairwise disjoint edges. Let G be a graph and let $M \subseteq E(G)$ be a matching in G. We denote by $\text{match}(G)$ the maximum-cardinality matching respectively the maximum-weight matching in G, depending on whether we have edge weights or not. If there are edge weights $\omega \colon E \to \mathbb{N}$, then for a matching M we denote by $\omega(M) = \sum_{e \in M} \omega(e)$ the weight of M. Moreover, we denote with $\omega(G)$ the weight of a maximum-weight matching $\text{match}(G)$, i. e. $\omega(G) = \omega(\text{match}(G))$. A vertex $v \in V$ is called *matched* with respect to M if there is an edge in M containing v, otherwise v is called *free* with respect to M. If the matching M is clear from the context, then we omit "with respect to M".

Kernelization. A *parameterized problem* is a set of instances (I, k) where $I \in \Sigma^*$ for a finite alphabet Σ, and $k \in \mathbb{N}$ is the *parameter*. We say that two instances (I, k) and (I', k') of parameterized problems P and P' are *equivalent* if (I, k) is a yes-instance for P if and only if (I', k') is a yes-instance for P'. A *kernelization* is an algorithm that, given an instance (I, k) of a parameterized problem P, computes in polynomial time an equivalent instance (I', k') of P (the *kernel*) such that $|I'| + k' \leq f(k)$ for some computable function f. We say that f measures the *size* of the kernel, and if $f(k) \in k^{O(1)}$, then we say that P admits a polynomial kernel. Often, a kernel is achieved by applying polynomial-time executable data reduction rules. We call a data reduction rule $\mathcal{R}$ *correct* if the new instance (I', k') that results from applying $\mathcal{R}$ to (I, k) is equivalent to (I, k). An instance is called *reduced* with respect to some data reduction rule if further application of this rule has no effect on the instance.

2 Kernelization Algorithms

In this section, we recall the data reduction rules for Maximum-Cardinality Matching and show how to lift them to Maximum-Weight Matching. For Maximum-Cardinality Matching two simple data reduction rules are due to a classic result of Karp and Sipser [9]. They deal with vertices of degree at most two.

▶ **Reduction Rule 2.1** ([9]). *Let $v \in V$. If $\deg(v) = 0$, then delete v. If $\deg(v) = 1$, then delete v and its neighbor, and decrease the solution size s by one.*

▶ **Reduction Rule 2.2** ([9]). *Let v be a vertex of degree two and let u, w be its neighbors. Then remove v, merge u and w, and decrease the solution size s by one.*

In each application of the two data reduction rules the considered vertex v is matched (hence the decrease of the solution size). When applying Reduction Rule 2.1, then v is matched to its only neighbor u. For Reduction Rule 2.2 the situation is not so clear as v is matched to u or to w depending on how the maximum-cardinality matching in the rest of the graph looks like. Thus, one can only fix the matching edge with endpoint v (in the original graph) in a postprocessing step.

Both of the above data reduction rules can be exhaustively applied in linear time. While for Reduction Rule 2.1 this is easy to see, for Reduction Rule 2.2 the algorithm needs further ideas [3]. Using the above data reduction rules, one can show a kernel with respect to the parameter feedback edge number, that is, the size of a minimum feedback edge set.

▶ **Theorem 1** ([13]). Maximum-Cardinality Matching *admits a linear-time computable linear-size kernel with respect to the parameter feedback edge number k.*

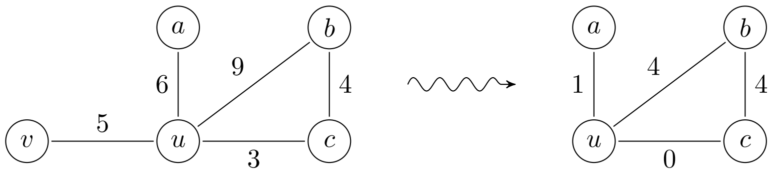

Figure 1 Left: Input graph. Right: The graph after applying Reduction Rule 2.4 to vertex v.

Applying the $O(m\sqrt{n})$-time algorithm for MAXIMUM-CARDINALITY MATCHING [14] altogether yields an $O(n + m + k^{1.5})$-time algorithm, where k is the feedback edge number.

Weighted Matching. In the remainder of this section, we show how to lift Theorem 1 to the weighted case. Reduction Rules 2.1 and 2.2 are based on the simple observation that for every vertex $v \in V$ of degree at least one, there exists a maximum-cardinality matching containing v: If v is not matched, then take an arbitrary neighbor u of v, remove the edge containing u from a maximum-cardinality matching, and add the edge uv. This observation does not hold in the weighted case – see e. g. Figure 1 (left side) where the only maximum-weight matching $\{au, bc\}$ leaves v free. Thus, we need new ideas to obtain data reduction rules for the weighted case.

Vertices of degree at most one. We start with the simple case of dealing with vertices of degree at most one. Here, the following data reduction rule is obvious.

▶ **Reduction Rule 2.3.** *If* $\deg(v) = 0$ *for a vertex* $v \in V$*, then delete* v*. If* $\omega(e) = 0$ *for an edge* $e \in E$*, then delete* e*.*

Next, we show how to deal with degree-one vertices, see Figure 1 for a visualization.

▶ **Reduction Rule 2.4.** *Let* $G = (V, E)$ *be a graph with non-negative edge weights* $\omega\colon E \to \mathbb{N}$*. Let* v *be a degree-one vertex and let* u *be its neighbor. Then delete* v*, set the weight of every edge* e *incident with* u *to* $\max\{0, \omega(e) - \omega(uv)\}$*, and decrease the solution value* s *by* $\omega(uv)$*.*

While proving the correctness of this rule (see next lemma) is relatively straightforward, the naive algorithm to exhaustively apply Reduction Rule 2.4 is too slow for our purpose: If the edge weights are adjusted immediately after deleting v, then exhaustively applying the rule to a star requires $\Theta(n^2)$ time. However, as we subsequently show, Reduction Rule 2.4 can be exhaustively applied in linear time.

▶ **Lemma 2.** *Reduction Rule 2.4 is correct.*

Due to space restrictions, we defer the poof of Lemma 2 to the arXiv version.

▶ **Lemma 3.** *Reduction Rule 2.4 can be exhaustively applied in* $O(n + m)$ *time.*

Proof. The basic idea of the algorithm exhaustively applying Reduction Rule 2.4 in linear time is as follows: We store in each vertex a number indicating the weight of the heaviest incident edge removed due to Reduction Rule 2.4. Then, whenever we want to access the "current" weight of an edge e, then we subtract from $\omega(e)$ the two numbers stored in the two incident vertices. Once Reduction Rule 2.4 is no more applicable, then we update the edge weights to get rid of the numbers in the vertices in order to create a MAXIMUM-WEIGHT MATCHING instance.

The details of the algorithm are as follows. First, in $O(n + m)$ time we collect all degree-one vertices in a list L and initialize for each vertex v a counter $c(v) := 0$. Then, we process L one by one. For a degree-one vertex $v \in L$, let u be its neighbor. We decrease s by $\max\{0, w(uv) - c(u) - c(v)\}$, then set $c(u) := c(u) + \max\{0, w(uv) - c(u) - c(v)\}$, and then delete v. If after the deletion of v its neighbor u has degree one, then u is added to L. Thus, after at most n steps, each one doable in constant time, we processed L. When L is empty, then in $O(m)$ time we update for each edge uv its weight $w(uv) := \max\{0, w(uv) - c(u) - c(v)\}$. This finishes the description of the algorithm.

Observe that we have the following invariant when processing the list L: the weight of an edge uv is $\max\{0, w(uv) - c(u) - c(v)\}$. With this invariant, it is easy to see that the algorithm indeed applies Reduction Rule 2.4 exhaustively. ◀

Note that after applying Reduction Rule 2.4 we can have weight-zero edges and thus Reduction Rule 2.3 might become applicable. We do not know whether Reduction Rules 2.3 and 2.4 *together* can be applied exhaustively in linear time. However, for the kernel we present in the end of this section it is sufficient to apply Reduction Rule 2.4 exhaustively.

Vertices of degree two. Lifting Reduction Rule 2.2 to the weighted case is more delicate than lifting Reduction Rule 2.1 to Reduction Rules 2.3 and 2.4. The reason is that the two incident edges might have different weights. As a consequence, we cannot decide locally what to do with a degree-two vertex. Instead, we process multiple degree-two vertices at once. To this end, we use the following notation.

▶ **Definition 4.** Let G be a graph. A path $P = v_0 v_1 \dots v_\ell$ is a *maximal path* in G if $\ell \geq 3$, the inner vertices $v_1, v_2, \dots, v_{\ell-1}$ all have degree two in G, but the endpoints v_0 and v_ℓ do not, that is, $\deg_G(v_1) = \dots = \deg_G(v_{\ell-1}) = 2$, $\deg_G(v_0) \neq 2$, and $\deg_G(v_\ell) \neq 2$.

▶ **Definition 5.** Let G be a graph. A cycle $C = v_0 v_1 \dots v_\ell v_0$ is a *pending cycle* in G if at most one vertex in C does *not* have degree two in G.

The reason to study maximal paths and pending cycles is that we can compute a maximum-weight matching in these graphs quickly, as stated next. This allows us to preprocess all vertices in a maximal path or a pending cycle at once.

▶ **Observation 6.** Maximum-Weight Matching *can be solved in $O(n)$ time on paths and cycles.*

Proof. If the input graph G is a path, then by exhaustively applying Reduction Rules 2.3 and 2.4, we can compute a maximum-weight matching. Otherwise, if G is a cycle, then we take an arbitrary edge e and distinguish two cases. First, we take e into a matching and remove both endpoints from the graph. In the resulting path, we compute in linear time a maximum-weight matching M. Second, we delete e and obtain a path for which we compute in linear time a maximum-weight matching M'. We then simply choose between $M \cup \{e\}$ and M' the heavier matching as the result. ◀

Now, using Observation 6, we introduce data reduction rules for maximal paths and pending cycles. Both rules are based on a similar idea which is easier to explain for a pending cycle. Let C be a pending cycle and $u \in C$ the degree-at-least-three vertex in C. Then there are two cases: u is matched with a vertex not in C or it is not. Now let M be a maximum-weight matching for G, and let M' be a maximum-weight matching with the constraint that u is matched to a vertex outside C. Clearly, $M \cap E(C)$ is at least as

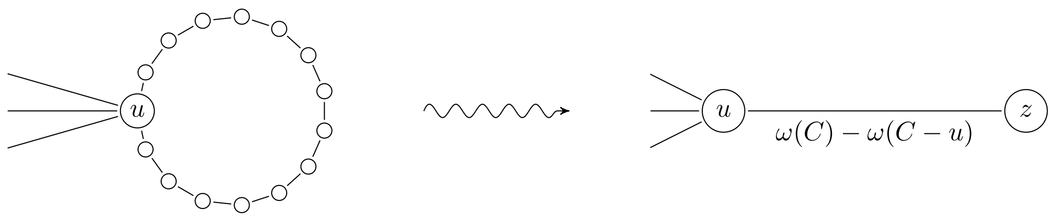

Figure 2 Left: A pending cycle C with u being the vertex of degree more than three. Right: The graph after applying Reduction Rule 2.5 where s is reduced by $\omega(C-u)$.

large as $M' \cap E(C)$. Looking only at C, all that we need to know is the difference of the weights of these two matchings. This can be encoded with one vertex z which replaces the whole cycle C (see Figure 2 for an illustration). Then, matching z corresponds to taking the matching in C and not matching z corresponds to taking the matching in $C - u$. Formalizing this idea, we arrive at the following data reduction rule.

▶ **Reduction Rule 2.5.** *Let G be a graph with non-negative edge weights. Let C be a pending cycle in G, where $u \in C$ has degree at least three in G. Then replace C by an edge uz with $\omega(uz) = \omega(C) - \omega(C-u)$ and decrease the solution value s by $\omega(C-u)$.*

▶ **Lemma 7.** *Reduction Rule 2.5 is correct.*

Proof. Let C be a pending cycle in G where $u \in C$ has degree at least three in G and let G' be the graph obtained applying Reduction Rule 2.5 to C. We show $\omega(G') = \omega(G) - \omega(C-u)$. Let M be a maximum-weight matching in G. Let $M_{\overline{C}} := M \setminus E(C)$. Observe that $\omega(M_{\overline{C}}) = \omega(M) - \omega(M \cap E(C)) \geq \omega(G) - \omega(C)$. If u is matched with respect to $M_{\overline{C}}$, then we have $M_{\overline{C}} = M \setminus E(C-u)$. Hence, $\omega(G') \geq \omega(M_{\overline{C}}) \geq \omega(G) - \omega(C-u)$. If u is free with respect to $M_{\overline{C}}$, then $M_{\overline{C}} \cup \{uz\}$ is a matching in G' with weight at least $(\omega(G) - \omega(C)) + (\omega(C) - \omega(C-u)) = \omega(G) - \omega(C-u)$. Hence, in both cases we have $\omega(G') \geq \omega(G) - \omega(C-u)$.

Conversely, let M' be a maximum-weight matching in G'. Recall that, for an edge-weighted graph H, $\text{match}(H)$ denotes a maximum-weight matching in H. If $uz \in M'$, then $(M' \setminus \{uz\}) \cup \text{match}(C)$ is a matching in G with $\omega(G') - (\omega(C) - \omega(C-u)) + \omega(C) = \omega(G') + \omega(C-u)$. Hence, $\omega(G) \geq \omega(G') + \omega(C-u)$. If $uz \notin M'$, then $M' \cup \text{match}(C-u)$ is a matching in G with weight at least $\omega(G') + \omega(C-u)$. Again, in both cases we have $\omega(G) \geq \omega(G') + \omega(C-u)$. Combined with $\omega(G') \geq \omega(G) - \omega(C-u)$, we arrive at $\omega(G') = \omega(G) - \omega(C-u)$. ◀

The basic idea for maximal paths is the same as for pending cycles. The difference is that we have to distinguish four cases depending on whether or not the two endpoints u and v of a maximal path P are matched within P. To simplify the notation, we set $\omega(uv) = 0$ if the edge uv does not exist in G. Furthermore, $P - u - v$ denotes the path obtained from removing in P the vertices u and v. This avoids some trivial case distinctions.

Figure 3 visualizes the next data reduction rule.

▶ **Reduction Rule 2.6.** *Let $G = (V, E)$ be a graph with non-negative edge weights $\omega\colon E \to \mathbb{N}$. Let P be a maximal path in G with endpoints u and v. Then remove all vertices in P except u and v, add a new vertex z and, if not already existing, add the edge uv. Furthermore, set $\omega(uz) := \omega(P - v) - \omega(P - u - v)$, $\omega(vz) := \omega(P - u) - \omega(P - u - v)$, and $\omega(uv) := \max\{\omega(uv), \omega(P) - \omega(P - u - v)\}$, and decrease the solution value s by $\omega(P - u - v)$.*

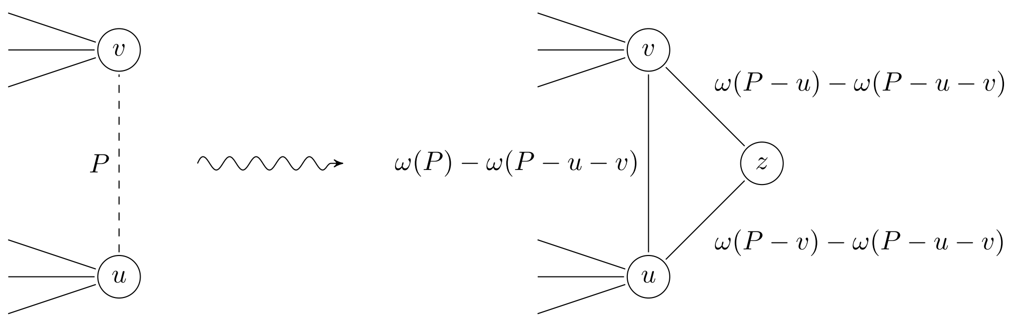
v
P
u
v
$\omega(P-u)-\omega(P-u-v)$
$\omega(P)-\omega(P-u-v)$
z
$\omega(P-v)-\omega(P-u-v)$
u

Figure 3 Applying Reduction Rule 2.6 on a path P with endpoints u and v (where u and v are not adjacent). The four choices for u and v on whether or not they are matched to a vertex within the path are reflected by the three (full) edges on the right where at most one can be taken into a matching. Since the edge uv is not contained in the input graph the weight of the edge uv in the reduced graph simplifies to the displayed value.

▶ **Lemma 8.** *Reduction Rule 2.6 is correct.*

Proof. Let G be the input graph with a maximal path P with endpoints u and v. Furthermore, let G' be the reduced instance with z defined as in the data reduction rule. We show that $\omega(G') = \omega(G) - \omega(P-u-v)$.

Let M be a maximum-weight matching for G. We define $M_{\overline{P}} := M \setminus E(P)$. Observe that $\omega(M_{\overline{P}}) = \omega(M) - \omega(M \cap E(P)) \geq \omega(G) - \omega(P)$ We distinguish four cases.

1. If both u and v are matched with respect to $M_{\overline{P}}$, then $M_{\overline{P}} = M \setminus E(P-u-v)$ and hence $\omega(M_{\overline{P}}) = \omega(M) - \omega(M \cap E(P-u-v)) \geq \omega(G) - \omega(P-u-v)$.
2. If u is matched and v is free with respect to $M_{\overline{P}}$, then $M_{\overline{P}} = M \setminus E(P-u)$ and hence $\omega(M_{\overline{P}}) \geq \omega(G) - \omega(P-u)$. Thus, $M_{\overline{P}} \cup \{vz\}$ is a matching of weight at least $(\omega(G) - \omega(P-u)) + (\omega(P-u) - \omega(P-u-v)) = \omega(G) - \omega(P-u-v)$.
3. If v is matched and u is free with respect to $M_{\overline{P}}$, then $M_{\overline{P}} = M \setminus E(P-v)$ and hence $\omega(M_{\overline{P}}) \geq \omega(G) - \omega(P-v)$. Thus, $M_{\overline{P}} \cup \{uz\}$ is a matching of weight at least $(\omega(G) - \omega(P-v)) + (\omega(P-v) - \omega(P-u-v)) = \omega(G) - \omega(P-u-v)$.
4. Finally, if both u and v are free with respect to $M_{\overline{P}}$, then $M_{\overline{P}} \cup \{uv\}$ is a matching of weight at least $(\omega(G) - \omega(P)) + (\omega(P) - \omega(P-u-v)) = \omega(G) - \omega(P-u-v)$.

Thus in each case we have $\omega(G') \geq \omega(G) - \omega(P-u-v)$.

Conversely, let M' be a maximum-weight matching for G'. We define $\overline{M'} := M' \setminus \{uz, vz, uv\}$. Again, we distinguish four cases.

1. If both u and v are matched with respect to $\overline{M'}$, then $\overline{M'} = M'$. Hence, $\overline{M'} \cup \text{match}(P-u-v)$ is a matching in G with weight at least $\omega(G') + \omega(P-u-v)$.
2. If u is matched and v is free with respect to $\overline{M'}$, then w.l.o.g. $vz \in M'$. Hence, $\overline{M'} \cup \text{match}(P-u)$ is a matching in G with weight at least $\omega(G') - (\omega(P-u) - \omega(P-u-v)) + \omega(P-u) = \omega(G') + \omega(P-u-v)$.
3. If u is matched and v is free with respect to $\overline{M'}$, then w.l.o.g. $uz \in M'$. Hence, $\overline{M'} \cup \text{match}(P-v)$ is a matching in G with weight at least $\omega(G') - (\omega(P-v) - \omega(P-u-v)) + \omega(P-v) = \omega(G') + \omega(P-u-v)$.
4. Finally, if both u and v are free with respect to $\overline{M'}$, then w.l.o.g $uv \in M'$ as $\omega(uv) \geq \omega(uz)$ and $\omega(uv) \geq \omega(vz)$. Now, we encounter two subcases.
   a. If $\omega(uv) > \omega(P) - \omega(P-u-v)$, then the edge uv is in G and in G', having the same weight in both graphs. Then, $M' \cup \text{match}(P-u-v)$ is a matching in G with weight at least $\omega(G') + \omega(P-u-v)$.
   b. Otherwise, $\overline{M'} \cup \text{match}(P)$ is a matching in G with weight at least $\omega(G') - (\omega(P) - \omega(P-u-v)) + \omega(P) = \omega(G') + \omega(P-u-v)$.

Hence, in all cases we have $\omega(G) \geq \omega(G') + \omega(P-u-v)$. Combined with $\omega(G') \geq \omega(G) + \omega(P-u-v)$, we can infer that $\omega(G') = \omega(G) - \omega(P-u-v)$. ◀

▶ **Lemma 9.** *Reduction Rules 2.5 and 2.6 can be exhaustively applied in $O(n+m)$ time.*

Proof. First, we collect in $O(n+m)$ time all maximal paths and all pending cycles [4, Lemma 2]. Given a maximal path or a pending cycle on ℓ vertices due to Observation 6 one can compute the necessary maximum-weight matchings (at most four) in $O(\ell)$ time. Moreover, replacing the maximal path or the pending cycle by the respective structure is doable in $O(\ell)$ time. Applying Reduction Rules 2.5 and 2.6 does not create new maximal paths (recall that a maximal path needs at least two vertices of degree two) or pending cycles. Thus, as all maximal paths and pending cycles combined contain at most n vertices, Reduction Rules 2.5 and 2.6 can be exhaustively applied in $O(n+m)$ time. ◀

Each of Reduction Rules 2.3, 2.4, and 2.6 can be exhaustively applied in linear time; however, we do not know whether all these data reduction rules together can be exhaustively applied in linear time. Note that after applying Reduction Rule 2.5 Reduction Rule 2.4 might become applicable. For our problem kernel below, however, Lemmas 3 and 9 are sufficient. In contrast to this subsequent theoretical result, in our experimental part it proved beneficial to apply the rules exhaustively in order to remove as many vertices and edges as possible.

▶ **Theorem 10.** MAXIMUM-WEIGHT MATCHING *admits a linear-time computable $20k$-vertex and $22k$-edge kernel with respect to the parameter feedback edge number k.*

Proof. The kernelization algorithm works as follows: First, exhaustively apply Reduction Rule 2.3 in $O(n+m)$ time. Second, exhaustively apply Reduction Rules 2.5 and 2.6 in $O(n+m)$ time (see Lemma 9). Third, exhaustively apply Reduction Rule 2.4 in $O(n+m)$ time (see Lemma 3). Without loss of generality, one can assume that the input graph does not contain a cycle where each vertex has degree two, because otherwise this cycle can be solved independently in linear time (see Observation 6). Note that when applying the rules in this order, the resulting graph $G = (V, E)$ does not contain any degree one-vertices, maximal paths, or pending cycles.

We claim that G has less than $20k$ vertices and $22k$ edges. First, note that the input graph contains at most k maximal paths [4, Lemma 1]. Thus, a feedback edge set $X \subseteq E$ for G contains at most $2k$ edges (each application of Reduction Rule 2.6 increases the feedback edge set by one). Denote with V^1_{G-X}, V^2_{G-X}, and $V^{\geq 3}_{G-X}$ the vertices that have degree one, two, and more than two in $G - X$. Observe that all vertices in the reduced graph G have degree at least two since it is reduced with respect to Reduction Rules 2.3 and 2.4. Thus, $|V^1_{G-X}| \leq 4k$ as each leaf in $G - X$ has to be incident to an edge in X. Next, since $G - X$ is a forest (or tree), we have $|V^{\geq 3}_{G-X}| < |V^1_{G-X}|$ and thus $|V^{\geq 3}_{G-X}| < 4k$. Finally, each degree-two vertex in G needs two neighbors of degree at least three since G is reduced with respect to Reduction Rules 2.5 and 2.6. Thus, the vertices in V^2_{G-X} are either incident to an edge in X or adjacent to one of the at most $|V^{\geq 3}_{G-X}| + 4k$ vertices in G that have degree at least three. The sum over all degrees of vertices in $V^{\geq 3}_{G-X}$ is

$$\sum_{v \in V^{\geq 3}_{G-X}} \deg_{G-X}(v) = 2m - \sum_{v \in V^{=2}_{G-X} \cup V^{=1}_{G-X}} \deg_{G-X}(v) \leq 2(n-1) - 2|V^{=2}_{G-X}| - |V^{=1}_{G-X}|$$

$$= 2|V^{\geq 3}_{G-X}| + |V^1_{G-X}| - 2 < 12k.$$

It follows that $|V^2_{G-X}| \leq 16k$. Thus, the number of vertices in G is $|V^1_{G-X}| + |V^2_{G-X}| + |V^{\geq 3}_{G-X}| \leq 20k$. Since $G - X$ is a forest, it follows that G has at most $|V| + 2k \leq 22k$ edges. ◀

Table 1 A selection of our test graphs from SNAP [12] with their respective size.

Graph	n	edges	Graph	n	edges
email-Eu-core	1,005	16,064	p2p-Gnutella08	6,301	20,777
p2p-Gnutella25	22,687	54,705	ca-CondMat	23,133	93,439
soc-...dot090221	82,141	$3.49 \cdot 10^5$	soc-Slashdot0811	77,360	$4.69 \cdot 10^5$
com-dblp	$3.2 \cdot 10^5$	$1.05 \cdot 10^6$	twitter-combined	81,306	$1.34 \cdot 10^6$
amazon0505	$4.1 \cdot 10^5$	$2.44 \cdot 10^6$	roadNet-CA	$2 \cdot 10^6$	$2.77 \cdot 10^6$
wiki-topcats	$1.8 \cdot 10^6$	$2.54 \cdot 10^7$	soc-LiveJournal1	$4.8 \cdot 10^6$	$4.29 \cdot 10^7$

3 Experimental Evaluation

In this section we provide an experimental evaluation of the data reduction rules on real-world graphs ranging from a few thousand vertices and edges to a hundred million vertices and edges. We analyze the effectiveness and efficiency of the kernelization as well as the effect on the subsequently used state-of-the-art solver “blossom5” of Kolmogorov [10].

3.1 Setup & Implementation Details

Setup. Our program is written in C++14 and source code in available from `http://fpt.akt.tu-berlin.de/software/`. One can replicate all experiments by following the manual next to the source code. We ran all our experiments on an Intel(R) Xeon(R) CPU E5-1620 3.60 GHz machine with 64 GB main memory under the Debian GNU/Linux 7.0 operating system, where we compiled the program with GCC 5.4.0. All tested graphs are from the established SNAP [12] data set. See Table 1 for a sample list of graphs with their respective number of vertices and edges. The weighted graphs are generated from the unweighted graphs by adding edge-weights between 1 and 1000 chosen independently and uniformly at random. The full list is given in the arXiv version.

Implementation Details. The first step of our program is to read the graph into our data structure. On average this took 9% of the overall running time. When running blossom5 we also measured the time to handover the graph from our data structure to the solver’s data structure, which took on average 4% of the overall running time.

We implemented kernelization algorithms for the unweighted and weighted case. The first kernelization is for Maximum-Cardinality Matching, which exhaustively applies Reduction Rules 2.1 and 2.2. Note that one can (theoretically) improve our implementation of Reduction Rule 2.2 by a linear-time algorithm of Bartha and Kresz [3]. However, our naive implementation proved to be sufficient. The second kernelization is for Maximum-Weight Matching. We use the algorithms described in Lemmas 3 and 9 to apply Reduction Rules 2.4 to 2.6. Deviating from the algorithm described in Theorem 10, based on empirical observations our program applies Reduction Rules 2.3 to 2.6 as long as possible. Hence, the kernelization does not run in linear time but further shrinks the input graph.

Note that all reported running times involving blossom5 are averages over 100 runs where we randomly permute vertex indices in the input. Although this permutation yields an isomorphic graph, we empirically observed that in the unweighted case the running time of blossom5 heavily depends on the permutation. For example, choosing a “good” or a “bad” permutation for the same graph yields speed ups of a factor 20 or more. (See the arXiv version for the results without this permutation.) When using blossom5 in the weighted case,

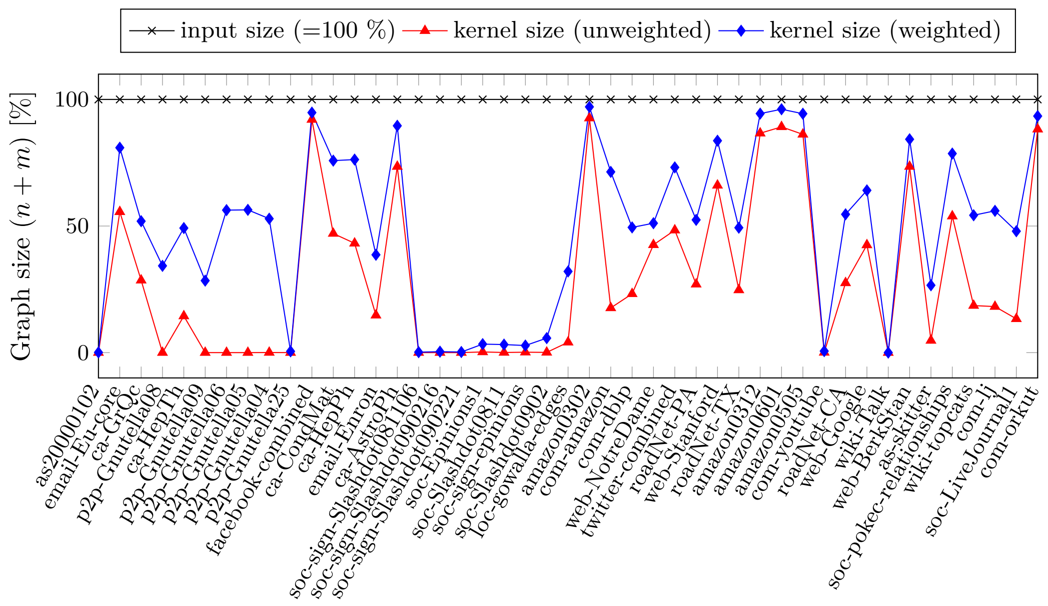

Figure 4 Percentage of the number of remaining edges and vertices after the respective kernelization algorithms (weighted, unweighted) relative to the numbers of vertices and edges in the input graph.

we did not observe this effect. For consistency, however, we take the average running time also in the weighted case. Note that our kernelization algorithm for the unweighted case was not at all affected by changing the permutation. In the weighted case, however, for different permutations the rules were applied in different order resulting in kernels slightly differing in size. The time for computing the random permutation is included in the times measured for reading and parsing of the graph.

3.2 Evaluation

We next present the results of our experimental evaluation starting with the size reduction and running time of the kernelization algorithm.

Kernel size. The effectiveness of our kernelization algorithms is displayed in Figure 4: Few graphs remain almost unchanged while other graphs are essentially solved by the kernelization algorithm. As expected, the kernelization algorithm for the unweighted case is much more effective than for the weighted case. On the 40 tested graphs, on average 70% of the vertices and edges are removed by the kernelization; the median is 81%. The least amenable graph was `amazon0302` with a size reduction of only 7%. In contrast, on 16 out of the 40 graphs the kernelization algorithm reduces more than 99% of the vertices and edges.

While the data reduction rules are less effective in the weighted case, they reduce the graphs on average still by 51% with the median value being a bit lower with 48%. The least amenable graph is again `amazon0302` with a size reduction of only 3%. Still, on seven out of the 40 graphs the kernelization algorithm reduces more than 99% of the vertices and edges.

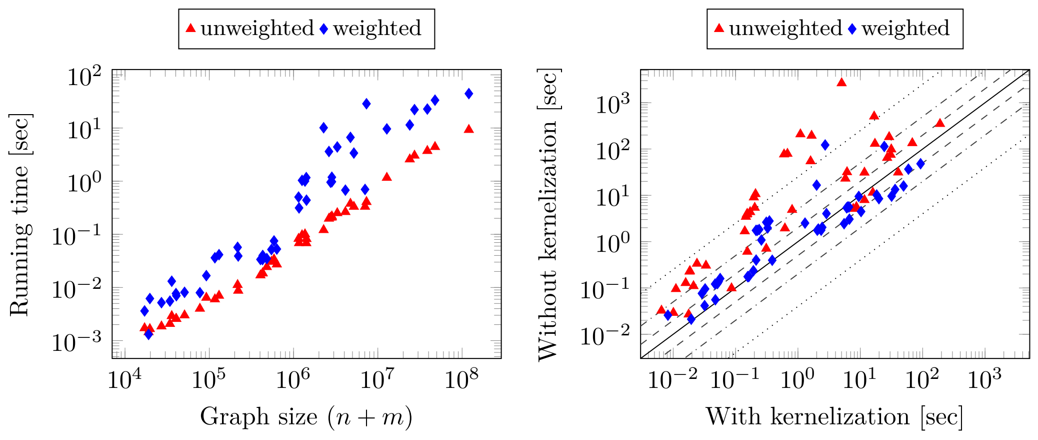

Figure 5 Left: Kernelization time depending on the graph sizes. Right: Running time comparison with and without using our kernelization algorithms before blossom5. The solid/dashed/dash dotted/dotted lines indicate a factor of 1/2/5/25 difference in the running time.

Kernelization time. We next discuss the running time of our kernelization algorithms on the test set. To this end, we consider the time spent on kernelization and the time spent on blossom5. We will, for now, omit the running times needed for reading and parsing the graph as these steps require on some instances more time than the kernelization algorithms.

As shown in Figure 5 (left), our kernelization algorithms are quite efficient. Even on the largest graphs with more than 120 million edges and vertices, the running time is less than 45 seconds in the weighted case and less than ten seconds in the unweighted case. In the unweighted case, our kernelization algorithm is always by at least a factor of 10 faster than blossom5 (on the first 40 graphs). Hence, applying the kernelization algorithm before the matching algorithm should – in the (unlikely) worst case of only few applications of the data reduction rules – only slightly increase the overall running time.

In the weighted case, however, our kernelization algorithm becomes slower than in the unweighted case. This is not surprising as the kernelization algorithm is more involved than the one for the unweighted case. Furthermore, blossom5 is significantly faster in the weighted case; on four graphs the matching algorithm is up to 2.5 times faster than our kernelization algorithm. However, on most graphs, our kernelization algorithm is still significantly faster than blossom5 (on average 17 times faster).

Running time comparison. We now compare the running time of only using blossom5 to first apply our kernelization algorithms and then use blossom5 on the kernel. Recall that all reported running times are averages over 100 runs. Since this 100 repetitions would have taken years for some graphs, we use only the 40 smallest graphs for the comparison. These graphs have between 12 thousand and 11 million edges.

Figure 5 (right) displays the results of the running time comparison. As one can clearly see, in the unweighted case the kernelization significantly accelerates the algorithm to find a maximum-cardinality matching. The speedup ranges from a factor 1.15 for the graph `facebook-combined` to a factor 525 for the graph `as-skitter` (the largest graph in this test set). The average speedup factor is 38, the median is 8.9.

For the weighted case, results are not as clear. With kernelization the algorithm is between 2.3 times slower for the graph `roadNet-CA` and 44.7 times faster for the graph `wiki-Talk`. However, the `wiki-Talk` is an exception as it is the only graph where the kernelization gave a speedup of a factor more than ten. The average speedup factor is 3.10 and the median is 1.3. On ten out of the 40 graphs the algorithm with kernelization was slower than without. As discussed above, there are even four graphs where blossom5 is even faster than our kernelization algorithm alone.

Summary. While the kernelization algorithms reduce the input graphs quite significantly in the weighted and unweighted case, the overall gain is very different in the two cases. When searching for a maximum-cardinality matching, we clearly recommend to always apply our kernelization algorithm. For the less clear weighted case, note that the kernelization is more frequently beneficial than it is not. However, the speedup is not as large as in the unweighted case and several instances are actually solved somewhat slower. There are several reasons for this behavior; some of which motivate future research and also lead to engineering challenges:

First, blossom5 is significantly faster on weighted graphs. We believe that the reason for this is in unweighted graphs there are a lot of symmetries, and unlucky tie-breaking seems to have a strong impact on blossom5. In the weighted case, the performance of blossom5 was much more consistent under permuting the vertices in the input graph. As a consequence, we believe that the following might speed up the algorithm: given an unweighted graph, introduce edge-weights such that a maximum-weight matching in the then weighted graph is also a maximum-cardinality matching in the unweighted graph. Using the famous Isolation Lemma [15] one might even enrich and support this with a theoretical analysis. As our focus was on data reduction, here we did not pursue this line of research yet.

Second, the kernelization algorithm in the weighted case is significantly slower than in the unweighted case (see Figure 5) as applying the rules is more involved. Note that Reduction Rules 2.1 and 2.2 only make changes in the local neighborhood of the affected vertices. This is not the case in the weighted case, where the application of Reduction Rules 2.4 to 2.6 involve iterations over all edges, see Lemmas 3 and 9. Hence, applying the data reduction rules exhaustively requires a larger overhead. Although some improvements in the implementation might be possible, an improved algorithmic approach to exhaustively apply the data reduction rules is needed. Is there a (quasi-)linear-time algorithm to exhaustively apply Reduction Rules 2.3 to 2.6?

4 Conclusion

Our work shows that it practically pays off to use (linear-time) data reduction rules for computing maximum matchings. The current state of the theoretical (kernel size upper bounds) analysis, however, is insufficient to fully explain this success. In future research, one might also study the combination of data reduction with linear-time approximation algorithms for matching [7]. Moreover, while our naive implementation for the unweighted case proved to be quite fast, we still work on improving the algorithm for the weighted case. In particular, parallelizing the kernelization algorithm seems promising for further speedups. We conclude with the following questions:

- Can exhaustive application of Reduction Rules 2.3 to 2.6 be realized in linear time?
- The "crown" data reduction rule known for the NP-hard Vertex Cover problem [1] can be transferred to Maximum-Cardinality Matching; however, our preliminary tests did not show a significant improvement of the kernelization algorithm. Is there any version of the "crown" data reduction rule with practical usefulness in matching computations?

References

1 Faisal N. Abu-Khzam, Michael R. Fellows, Michael A. Langston, and W. Henry Suters. Crown structures for vertex cover kernelization. *Theory of Computing Systems*, 41(3):411–430, 2007.

2 Takuya Akiba and Yoichi Iwata. Branch-and-reduce exponential/fpt algorithms in practice: A case study of vertex cover. *Theoretical Computer Science*, 609:211–225, 2016.

3 M. Bartha and M. Kresz. A depth-first algorithm to reduce graphs in linear time. In *Proceeding of the 11th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC' 09)*, pages 273–281. IEEE, 2009.

4 Matthias Bentert, Alexander Dittmann, Leon Kellerhals, André Nichterlein, and Rolf Niedermeier. Towards improving brandes' algorithm for betweenness centrality. *CoRR*, abs/1802.06701, 2018. URL: `http://arxiv.org/abs/1802.06701`.

5 Lijun Chang, Wei Li, and Wenjie Zhang. Computing A near-maximum independent set in linear time by reducing-peeling. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD '17)*, pages 1181–1196. ACM, 2017.

6 David Coudert, Guillaume Ducoffe, and Alexandru Popa. Fully polynomial FPT algorithms for some classes of bounded clique-width graphs. In *Proceedings of the 29th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '18)*, pages 2765–2784, 2018.

7 Ran Duan and Seth Pettie. Linear-time approximation for maximum weight matching. *Journal of the ACM*, 61(1):1:1–1:23, 2014.

8 Ran Duan, Seth Pettie, and Hsin-Hao Su. Scaling algorithms for weighted matching in general graphs. *ACM Transactions on Algorithms*, 14(1):8:1–8:35, 2018.

9 Richard M. Karp and Michael Sipser. Maximum matchings in sparse random graphs. In *Proceedings of the 22nd Annual IEEE Symposium on Foundations of Computer Science (FOCS '81)*, pages 364–375. IEEE, 1981.

10 Vladimir Kolmogorov. Blossom V: a new implementation of a minimum cost perfect matching algorithm. *Mathematical Programming Computation*, 1(1):43–67, 2009.

11 Bernd Korte and Jens Vygen. *Combinatorial Optimization – Theory and Algorithms*. Springer, 2018.

12 Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection. `http://snap.stanford.edu/data`, 2014.

13 George B. Mertzios, André Nichterlein, and Rolf Niedermeier. The power of linear-time data reduction for maximum matching. In *Proceedings of the 42nd International Symposium on Mathematical Foundations of Computer Science (MFCS 2017)*, LIPIcs, pages 46:1–46:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017.

14 Silvio Micali and Vijay V. Vazirani. An $O(\sqrt{|V|}|E|)$ algorithm for finding maximum matching in general graphs. In *Proceedings of the 21st Annual IEEE Symposium on Foundations of Computer Science (FOCS '80)*, pages 17–27. IEEE, 1980.

15 Ketan Mulmuley, Umesh V. Vazirani, and Vijay V. Vazirani. Matching is as easy as matrix inversion. *Combinatorica*, 7(1):105–113, 1987.

16 Sanne Wøhlk and Gilbert Laporte. Computational comparison of several greedy algorithms for the minimum cost perfect matching problem on large graphs. *Computers & OR*, 87:107–113, 2017.