The Power of Linear-Time Data Reduction for Maximum Matching*

George B. Mertzios[†1], André Nichterlein[‡2], and Rolf Niedermeier[3]

1 **School of Engineering and Computing Sciences, Durham University, UK**
george.mertzios@durham.ac.uk
2 **School of Engineering and Computing Sciences, Durham University, UK, and Institut für Softwaretechnik und Theoretische Informatik, TU Berlin, Germany**
andre.nichterlein@tu-berlin.de
3 **Institut für Softwaretechnik und Theoretische Informatik, TU Berlin, Germany**
rolf.niedermeier@tu-berlin.de

Abstract

Finding maximum-cardinality matchings in undirected graphs is arguably one of the most central graph primitives. For m-edge and n-vertex graphs, it is well-known to be solvable in $O(m\sqrt{n})$ time; however, for several applications this running time is still too slow. We investigate how linear-time (and almost linear-time) data reduction (used as preprocessing) can alleviate the situation. More specifically, we focus on *linear-time kernelization*. We start a deeper and systematic study both for general graphs and for bipartite graphs. Our data reduction algorithms easily comply (in form of preprocessing) with every solution strategy (exact, approximate, heuristic), thus making them attractive in various settings.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, G.2.2 Discrete Mathematics: Graph Theory

Keywords and phrases Maximum-cardinality matching, bipartite graphs, linear-time algorithms, kernelization, parameterized complexity analysis, FPT in P.

Digital Object Identifier 10.4230/LIPIcs.MFCS.2017.46

1 Introduction

"Matching is a powerful piece of algorithmic magic" [22]. In the maximum matching problem, given an undirected graph, one has to compute a maximum set of nonoverlapping edges. Maximum matching is arguably among the most fundamental graph-algorithmic primitives allowing for a polynomial-time algorithm. More specifically, on an n-vertex and m-edge graph a maximum matching can be found in $O(m\sqrt{n})$ time [20]. Improving this upper time bound resisted decades of research. Recently, however, Duan and Pettie [9] presented a linear-time algorithm that computes a $(1-\epsilon)$-approximate maximum-weight matching, where the running time dependency on ϵ is $\epsilon^{-1}\log(\epsilon^{-1})$. For the unweighted case, the $O(m\sqrt{n})$ algorithm of Micali and Vazirani [20] implies a linear-time $(1-\epsilon)$-approximation, where in this case the running time dependency on ϵ is ϵ^{-1} [9]. We take a different route: First, we do not give up the quest for optimal solutions. Second, we focus on efficient – more specifically, linear-time executable – data reduction rules, that is, not solving an instance but significantly

* A full version with all proof details is available at https://arxiv.org/abs/1609.08879
† Partially supported by the EPSRC grant EP/P020372/1.
‡ Supported by a postdoc fellowship of DAAD while at Durham University.

shrinking its size before actually solving the problem. Doing so, however, we focus here on the unweighted case. In the context of decision problems and parameterized complexity analysis this approach is known as kernelization.

The spirit behind our approach is thus closer to the identification of efficiently solvable special cases of maximum matching. There is quite some body of work in this direction. For instance, since an augmenting path can be found in linear time [11], the standard augmenting path-based algorithm runs in $O(s(n+m))$ time, where s is the number of edges in the maximum matching. Yuster [25] developed an $O(rn^2 \log n)$-time algorithm, where r is the difference between maximum and minimum degree of the input graph. Moreover, there are linear-time algorithms for computing maximum matchings in special graph classes, including convex bipartite [23], strongly chordal [8], chordal bipartite [7], and cocomparability graphs [19].

All this and the more general spirit of "parameterization for polynomial-time solvable problems" [13] (also referred to as "FPT in P" or "FPTP" for short) forms the starting point of our research. Remarkably, Fomin et al. [10] recently developed an algorithm to compute a maximum matching in graphs of treewidth k in $O(k^4 n \log n)$ randomized time.

Following the paradigm of *kernelization*, that is, provably effective and efficient data reduction, we provide a systematic exploration of the power of not only polynomial-time but actually linear-time data reduction for maximum matching. Thus, our aim (fitting within FPTP) is to devise problem kernels that are computable in linear time. In other words, the fundamental question we pose is whether there is a very efficient preprocessing that provably shrinks the input instance, where the effectiveness is measured by employing some parameters. The philosophy behind this is that if we can design linear-time data reduction algorithms, then we may employ them for free before afterwards employing any super-linear-time solving algorithm. We believe that this sort of question deserves deeper investigation and we initiate it based on the matching problem.

As kernelization is defined for decision problems, we use in the remainder of the paper the *decision version* of maximum matching. In a nutshell, a kernelization of a decision problem instance is an algorithm that produces an equivalent instance whose size can solely be upper-bounded by a function in the parameter (preferably a polynomial). The focus on decision problems is justified by the fact that all our results, although formulated for the decision version, in a straightforward way extend to the corresponding optimization version.

(Maximum-Cardinality) Matching
Input: An undirected graph $G = (V, E)$ and a nonnegative integer s.
Question: Is there a size s subset $M_G \subseteq E$ of nonoverlapping (i.e. disjoint) edges?

Note that for any polynomial-time solvable problem solving the given instance and returning a trivial yes- or no-instance always produces a constant-size kernel in polynomial time. Hence, we are looking for kernelization algorithms that are *faster* than the algorithms solving the problem. The best we usually can hope for is linear time. For NP-hard problems, each polynomial-time kernelization algorithm is faster than any solution algorithm, unless P = NP. While the focus of classical kernelization for NP-hard problems is mostly on improving the *size* of the kernel, we particularly emphasize that for polynomially solvable problems it now becomes mandatory to also focus on the *running time* of the kernelization algorithm. Indeed, we consider linear-time kernelization as the holy grail and this drives our research when studying kernelization for Matching.

Our contributions. We present three kernels for Matching (see Table 1 for an overview). All our parameterizations can be categorized as "distance to triviality" [6, 14, 18, 24]. They are motivated as follows. First, note that it is important that the parameters we exploit can

Table 1 Our kernelization results.

Parameter k	running time	kernel size	
Results for Matching			
Feedback edge number	$O(n+m)$	$O(k)$ vertices and edges	(Theorem 3)
Feedback vertex number	$O(kn)$	$2^{O(k)}$ vertices and edges	(Theorem 11)
Results for Bipartite Matching			
Distance to chain graphs	$O(n+m)$	$O(k^3)$ vertices	(Theorem 15)

be computed, or well approximated (within constant factors), in linear time regardless of the parameter value. For instance, it is not known whether this is possible for treewidth. Next, note that maximum-cardinality matchings can be trivially found in linear time on trees (or forests). So we consider the edge deletion distance (feedback edge number) and vertex deletion distance (feedback vertex number) to forests. Notably, there is a trivial linear-time algorithm for computing the feedback edge number and there is a linear-time factor-4 approximation algorithm for the feedback vertex number [1]. We mention in passing that the parameter vertex cover number, which is lower-bounded by the feedback vertex number, has been frequently studied for kernelization [3, 4]. In particular, Gupta and Peng [15] and Giannopoulou et al. [13] provided a linear-time computable quadratic-size kernel for Matching with respect to the parameter solution size (or equivalently vertex cover number). Coming to bipartite graphs, we parameterize by the vertex deletion distance to chain graphs which is motivated as follows. First, chain graphs form one of the most obvious easy cases for bipartite graphs where Matching can be solved in linear time [23]. Second, we show that the vertex deletion distance of any bipartite graph to a chain graph can be 2-approximated in linear time. Moreover, vertex deletion distance to chain graphs lower-bounds the vertex cover number of a bipartite graph, and thus gives a stronger parameterization [18] than vertex cover number.

An overview of our main results is given in Table 1. We study kernelization for Matching parameterized by the feedback vertex number, that is, the vertex deletion distance to a forest (see Section 2). As a warm-up we first show that a subset of our data reduction rules for the "feedback vertex set kernel" also yields a linear-time computable linear-size kernel for the typically much larger parameter feedback edge number (see Section 2.1). As for Bipartite Matching no faster algorithm is known than on general graphs, we kernelize Bipartite Matching with respect to the vertex deletion distance to chain graphs (see Section 3).

Seen from a high level, our two technical main results (Theorems 11 and 15, see Table 1) employ the same algorithmic strategy, namely upper-bounding (as a function of the parameter) the number of neighbors in the appropriate vertex deletion set X; that is, X being the feedback vertex set or in the deletion set to chain graphs, respectively. To achieve this we develop new *"irrelevant edge techniques"* tailored to these two kernelization problems. More specifically, whenever a vertex v of the deletion set X has large degree, we efficiently detect edges incident to v whose removal does not change the size of the maximum matching. Then the remaining graph can be further shrunk by scenario-specific data reduction rules. While this approach of removing irrelevant edges is natural, the technical details and the proofs of correctness become quite technical and combinatorially challenging.

Note that there exists a trivial $O(km)$-time solving (not only kernelization) algorithm, where k is the feedback vertex number. Our kernel has size $2^{O(k)}$. Therefore, only if $k = o(\log n)$ our kernelization algorithm *provably* shrinks the initial instance. However, our result

is still relevant: First, our data reduction rules might assist in proving a polynomial upper bound on the kernel size – so our result is a first step in this direction. Second, the running time $O(kn)$ of our kernelization algorithm is a kind of "half way" between $O(km)$ (which could be as bad as $O(k^2n)$) and $O(n+m)$ (which is best possible). Finally, note that this work focuses on theoretical and worst-case analysis; in practice, our kernelization algorithm might achieve much better upper bounds on real-world input instances.

As a technical side remark, we emphasize that in order to achieve a linear-time kernelization algorithm, we often need to use suitable data structures and to carefully design the appropriate data reduction rules to be exhaustively applicable in linear time, making this form of "algorithm engineering" much more relevant than in the classical setting of mere polynomial-time data reduction rules.

Notation and Observations. We use standard notation from graph theory. Merging two vertices u and v means to first introduce a new vertex w with $N(w) = N(u) \cup N(v)$ and then delete u and v. A feedback vertex (edge) set of a graph G is a set X of vertices (edges) such that $G - X$ is a tree or a forest. The feedback vertex (edge) number denotes the size of a minimum feedback vertex (edge) set. All paths we consider are simple paths. Two paths in a graph are called *internally vertex-disjoint* if they are either completely vertex-disjoint or they overlap only in their endpoints. A *matching* in a graph is a set of pairwise disjoint edges. Let $G = (V, E)$ be a graph and let $M \subseteq E$ be a matching in G. The degree of a vertex is denoted by $\deg(v)$. 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". An *alternating path* with respect to M is a path in G such that every second edge of the path is in M. An *augmenting path* is an alternating path whose endpoints are free. It is well known that a matching M is maximum if and only if there is no augmenting path for it. Let $M \subseteq E$ and $M' \subseteq E$ be two matchings in G. We denote by $G(M, M') := (V, M \triangle M')$ the graph containing only the edges in the symmetric difference of M and M', that is, $M \triangle M' := M \cup M' \setminus (M \cap M')$. Observe that every vertex in $G(M, M')$ has degree at most two.

▶ **Observation 1.** *Let $G = (V, E)$ be a graph with a maximum matching M_G, let $X \subseteq V$ be a vertex subset of size k, and let M_{G-X} be a maximum matching for $G - X$. Then, $|M_{G-X}| \leq |M_G| \leq |M_{G-X}| + k$.*

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 $f(k) \in k^{O(1)}$, 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 for Matching on General Graphs

In this section we first present as a warm-up a simple, linear-size kernel for Matching with respect to the parameter feedback edge number (see Section 2.1). Exploiting the data reduction rules and ideas used for this kernel, we then present the main result of this section: an exponential-size kernel for the typically much smaller parameter feedback vertex number (see Section 2.2).

2.1 Warm-up: Parameter feedback edge number

We provide a linear-time computable linear-size kernel for Matching parameterized by the feedback edge number, that is, the size of a minimum feedback edge set. Observe that a minimum feedback edge set can be computed in linear time via a simple depth-first search or breadth-first search. The kernel is based on the next two simple data reduction rules due to Karp and Sipser [17]. They deal with vertices of degree at most two.

▶ **Reduction Rule 2.1.** *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 (v is matched with its neighbor).*

▶ **Reduction Rule 2.2.** *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.*

Reduction Rules 2.1 and 2.2 are correct; however, it is not clear whether Reduction Rule 2.2 can be exhaustively applied in linear time. Fortunately, for our purpose it suffices to consider the following restricted version which we can exhaustively apply in linear time.

▶ **Reduction Rule 2.3.** *Let v be a vertex of degree two and u, w be its neighbors with u and w having degree at most two. Then remove v, merge u and w, and decrease s by one.*

▶ **Lemma 2.** *Reduction Rules 2.1 and 2.3 can be exhaustively applied in $O(n + m)$ time.*

▶ **Theorem 3.** Matching *admits a linear-time computable linear-size kernel with respect to the parameter feedback edge number k.*

Proof. Apply Reduction Rules 2.1 and 2.3 exhaustively in linear time (Lemma 2). We claim that the reduced graph $G = (V, E)$ has less than $12k$ vertices and less than $13k$ edges. Denote with $X \subseteq E$ a feedback edge set for G, $|X| \leq k$. Furthermore, 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 the $G - X$. Thus, $|V^1_{G-X}| \leq 2k$ 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 $|V^{\geq 3}_{G-X}| < 2k$. Finally, each degree-two vertex in G needs at least one neighbor of degree at least three since G is reduced with respect to Reduction Rule 2.3. Thus, the vertices in V^2_{G-X} are either incident to an edge in X or adjacent to one of the at most $|V^{\geq 3}_{G-X}| + 2k$ vertices in G that have degree at least three. Since the sum over all degrees of vertices in $V^{\geq 3}_{G-X}$ is at most $\sum_{v \in V^{\geq 3}_{G-X}} \deg_{G-X}(v) \leq 2|V^{\geq 3}_{G-X}| + |V^1_{G-X}| < 6k$, it follows that $|V^2_{G-X}| \leq 8k$. Thus, the number of vertices in G is $|V^1_{G-X}| + |V^2_{G-X}| + |V^{\geq 3}_{G-X}| \leq 12k$. Since $G - X$ is a forest, it follows that G has at most $|V| + k \leq 13k$ edges. ◀

Applying the $O(m\sqrt{n})$-time algorithm for Matching [20] on the kernel yields:

▶ **Corollary 4.** Matching *can be solved in $O(n + m + k^{1.5})$ time, where k is the feedback edge number.*

2.2 Parameter feedback vertex number

We next provide for MATCHING a kernel of size $2^{O(k)}$ computable in $O(kn)$ time where k is the feedback vertex number. Using a known linear-time factor 4-approximation algorithm [1], we can approximate feedback vertex set and use it in our kernelization algorithm.

Roughly speaking, our kernelization algorithm extends the linear-time computable kernel with respect to the parameter feedback edge set. Thus, Reduction Rules 2.1 and 2.3 play an important role in the kernelization. Compared to the other kernels presented in this paper, the kernel presented here comes at the price of higher running time $O(kn)$ and bigger kernel size (exponential size). It remains open whether MATCHING parameterized by the feedback vertex number admits a linear-time computable kernel (possibly of exponential size), and whether it admits a polynomial kernel computable in $O(kn)$ time.

Subsequently, we describe our kernelization algorithm which keeps in the kernel all vertices in the given feedback vertex set X and shrinks the size of $G - X$. Before doing so, we need some further notation. In this section, we assume that each tree is rooted at some arbitrary (but fixed) vertex such that we can refer to the parent and children of a vertex. A leaf in $G - X$ is called a *bottommost leaf* either if it has no siblings or if all its siblings are also leaves. (Here, bottommost refers to the subtree with the root being the parent of the considered leaf.) The outline of the algorithm is as follows (we assume throughout that $k < \log n$ since otherwise the input instance is already a kernel of size $O(2^k)$):

1. Reduce G with respect to Reduction Rules 2.1 and 2.3.
2. Compute a maximum matching M_{G-X} in $G - X$.
3. Modify M_{G-X} in linear time such that only the leaves of $G - X$ are free.
4. Bound the number of free leaves in $G - X$ by k^2.
5. Bound the number of bottommost leaves in $G - X$ by $O(k^2 2^k)$.
6. Bound the degree of each vertex in X by $O(k^2 2^k)$. Then, use Reduction Rules 2.1 and 2.3 to provide the kernel of size $2^{O(k)}$.

Whenever we reduce the graph at some step, we also show that the applied data reduction is *correct*. That is, the given instance is a yes-instance if and only if the reduced one is a yes-instance. The correctness of our kernelization algorithm then follows by the correctness of each step. We discuss in the following some details of each step.

2.2.1 Steps 1 to 3

By Lemma 2 we can perform Step 1 in linear time. A maximum matching in Step 2 can be computed by repeatedly matching a free leaf to its neighbor and by removing both vertices from the graph (thus effectively applying Reduction Rule 2.1 to $G - X$). By Lemma 2, this can be done in linear time. Step 3 can be done in $O(n)$ time by traversing each tree in M_{G-X} in a BFS manner starting from the root: If a visited inner vertex v is free, then observe that all children are matched since M_{G-X} is maximum. Pick an arbitrary child u of v and match it with v. The vertex w that was previously matched to u is now free and since it is a child of u, it will be visited in the future. Observe that Steps 2 and 3 do not change the graph but only the auxiliary matching M_{G-X}, and thus these steps are correct.

2.2.2 Step 4

Recall that our goal is to upper-bound the number of edges between vertices of X and $V \setminus X$, since we can then use a simple analysis as for the parameter feedback edge set. Observe that if a vertex $x \in X$ has at least k neighbors in $V \setminus X$ that are free wrt. M_{G-X}, then there exists a maximum matching where x is matched to one of these k vertices since at most $k - 1$

can be "blocked" by other matching edges. This means that we can delete all other edges incident to x. Formalizing this idea, we obtain the following data reduction rule.

▶ **Reduction Rule 2.4.** *Let $G = (V, E)$ be a graph, let $X \subseteq V$ be a subset of size k, and let M_{G-X} be a maximum matching for $G - X$. If there is a vertex $x \in X$ with at least k free neighbors $V_x = \{v_1, \ldots, v_k\} \subseteq V \setminus X$, then delete all edges from x to vertices in $V \setminus V_x$.*

To finish Step 4, we exhaustively apply Reduction Rule 2.4 in linear time. Afterwards, there are at most k^2 free (wrt. to M_{G-X}) leaves in $G - X$ that have at least one neighbor in X since each of the k vertices in X is adjacent to at most k free leaves. Thus, applying Reduction Rule 2.1 we can remove the remaining free leaves that have no neighbor in X. However, since for each degree-one vertex also its neighbor is removed, we might create new free leaves and need to again apply Reduction Rule 2.4 and update the matching (see Step 3). This process of alternating application of Reduction Rules 2.1 and 2.4 stops after at most k rounds since the neighborhood of each vertex in X can be changed by Reduction Rule 2.4 at most once. This shows the running time $O(k(n+m))$. We next show how to improve this to $O(n+m)$ time and arrive at the final lemma of this subsection.

▶ **Lemma 5.** *Given a matching instance (G, s) and a feedback vertex set X, one can compute in linear time an instance (G', s') with feedback vertex set X and a maximum matching $M_{G'-X}$ in $G' - X$ such that the following holds.*

- *There is a matching of size s in G if and only if there is a matching of size s' in G'.*
- *Each vertex that is free wrt. $M_{G'-X}$ is a leaf in $G' - X$.*
- *There are at most k^2 free leaves in $G' - X$.*

2.2.3 Step 5

Step 5 reduces the graph in $O(kn)$ time so that at most $k^2(2^k + 1)$ bottommost leaves will remain in the forest $G - X$. We restrict ourselves to consider leaves that are matched with their parent vertex in M_{G-X} and that do not have a sibling. Any sibling of a bottommost leaf is by definition also a leaf. Thus, at most one of these leaves (the bottommost leaf or its siblings) is matched with respect to M_{G-X} and all other leaves are free. Recall that in the previous step we upper-bounded the number of free leaves with respect to M_{G-X} by k^2. Hence there are at most k^2 bottommost leaves with siblings.

Our general strategy for this step is to extend the idea behind Reduction Rule 2.4: We want to keep for each pair of vertices $x, y \in X$ at most k different internally vertex-disjoint augmenting paths from x to y. (For ease of notation we keep k paths although keeping $k/2$ is sufficient.) In this step, we only consider augmenting paths of the form x, u, v, y where v is a bottommost leaf and u is v's parent in $G - X$. Assume that the parent u of v is adjacent to some vertex $x \in X$. Observe that in this case any augmenting path starting with the two vertices x and u has to continue to v and end in a neighbor of v. Thus, the edge $\{x, u\}$ can be only used in augmenting paths of length three. Furthermore, all these length-three augmenting paths are clearly internally vertex-disjoint. If we do not need the edge $\{x, u\}$ because we kept k augmenting paths from x already, then we can delete $\{x, u\}$. Furthermore, if we deleted the last edge from u to X (or u had no neighbors in X in the beginning), then u is a degree-two vertex in G and can be removed by applying Reduction Rule 2.2. As the child v of u is a leaf in $G - X$, it follows that v has at most $k + 1$ neighbors in G. Thus, an application of Reduction Rule 2.2 to remove u takes $O(k)$ time.

Counting for each pair $x \in N(u) \cap X$ and $y \in N(v) \cap X$ one augmenting path gives in a simple worst-case analysis $O(k^2)$ time per edge; this is too slow for our purposes. Instead, we

count for each vertex $x \in N(u) \cap X$ and for each *set* $Y = N(v) \cap X$ one augmenting path. In this way, we know that for each $y \in Y$ there is one augmenting path from x to y, without iterating through all $y \in Y$. We get an exponential factor in the bound of the bottommost leaves since there are 2^k subsets of X, but we can perform this step in $O(kn)$ time as follows.

▶ **Lemma 6.** *Let $(G = (V, E), s)$ be a matching instance, let $X \subseteq V$ be a feedback vertex set, and let M_{G-X} be a maximum matching for $G - X$ with at most k^2 free vertices in $G - X$ that are all leaves. Then, can compute in $O(kn)$ time an instance (G', s') with feedback vertex set X and a maximum matching $M_{G'-X}$ in $G' - X$ such that the following holds.*

- *There is a matching of size s in G if and only if there is a matching of size s' in G'.*
- *There are at most $k^2(2^k + 1)$ bottommost leaves in $G' - X$.*
- *There are at most k^2 free vertices in $G' - X$ and they are all leaves.*

2.2.4 Step 6

In this subsection, we provide the final step of our kernelization algorithm. Recall that in the previous steps we have upper-bounded the number of bottommost leaves in $G - X$ by $O(k^2 2^k)$, we computed a maximum matching M_{G-X} for $G - X$ such that at most k^2 vertices are free wrt. M_{G-X} and all free vertices are leaves in $G - X$. Using this, we next show how to reduce G to a graph of size $O(k^3 2^k)$. To this end we need some further notation. A leaf in $G - X$ that is not bottommost is called a *pendant*. We define T to be the *pendant-free tree (forest) of* $G - X$, that is, the tree (forest) obtained from $G - X$ by removing all pendants. The next observation shows that $G - X$ is not much larger than T. Together with the second observation, this allows us to restrict ourselves in the following on giving an upper bound on the size of T.

▶ **Observation 7.** *Let $G - X$ be as described above with vertex set $V \setminus X$ and let T be the pendant-free tree (forest) of $G - X$ with vertex set V_T. Then, $|V \setminus X| \leq 2|V_T| + k^2$.*

▶ **Observation 8.** *Let F be a forest, let F' be the pendant-free forest of F, and let B be the set of all bottommost leaves in F. Then, the set of leaves in F' is exactly B.*

From Observation 8 it follows that the set B of bottommost leaves in $G - X$ is exactly the set of leaves in T. In the previous step we reduced the graph such that $|B| \leq k^2(2^k + 1)$. Thus, T has at most $k^2(2^k + 1)$ vertices of degree one and, since T is a tree (a forest), T also has at most $k^2(2^k + 1)$ vertices of degree at least three. Let V_T^2 be the vertices of degree two in T and let $V_T^{\neq 2}$ be the remaining vertices in T. From the above it follows that $|V_T^{\neq 2}| \leq 2k^2(2^k + 1)$. Hence, it remains to bound the size of V_T^2. To this end, we will upper-bound the degree of each vertex in X by $O(k^2 2^k)$ and then use Reduction Rules 2.1 and 2.3. We will check for each edge $\{x, v\} \in E$ with $x \in X$ and $V \setminus X$ whether we "need" it. This check will use the idea from the previous subsection where each vertex in X needs to reach each subset $Y \in X$ at most k times via an augmenting path. Similarly as in the previous section, we want to keep "enough" of these augmenting paths. However, this time the augmenting paths might be long, while different augmenting paths might overlap. To still use the basic approach, we use the following lemma stating that we can still somehow replace augmenting paths.

▶ **Lemma 9.** *Let M_{G-X} be a maximum matching in the forest $G - X$. Let P_{uv} be an augmenting path for M_{G-X} in G from u to v. Let P_{wx}, P_{wy}, and P_{wz} be three internally vertex-disjoint augmenting paths from w to x, y, and z, respectively, such that P_{uv} intersects all of them. Then, there exist two vertex-disjoint augmenting paths with endpoints u, v, w, and one of the three vertices x, y, and z.*

Proof. Label the vertices in P_{uv} alternating as *odd* or *even* with respect to P_{uv} so that no two consecutive vertices have the same label, u is *odd*, and v is *even*. Analogously, label the vertices in P_{wx}, P_{wy}, and P_{wz} as odd and even with respect to P_{wx}, P_{wy}, and P_{wz} respectively so that w is always odd. Since all these paths are augmenting, it follows that each edge from an even vertex to its succeeding odd vertex is in the matching M_{G-X} and each edge from an odd vertex to its succeeding even vertex is not in the matching. Observe that P_{uv} intersects each of the other paths at least at two consecutive vertices, since every second edge must be an edge in M_{G-X}. Since $G-X$ is a forest and all vertices in X are free with respect to M_{G-X}, it follows that the intersection of two augmenting paths is connected and thus a path. Since P_{uv} intersects the three augmenting paths from w, it follows that at least two of these paths, say P_{wx} and P_{wy}, have a "fitting parity", that is, in the intersections of P_{uv} with P_{wx} and with P_{wy} the even vertices with respect to P_{uv} are either even or odd with respect to *both* P_{wx} and P_{wy}.

Assume w.l.o.g. that in the intersections of the paths the vertices have the same label with respect to the three paths (if the labels differ, then revert the ordering of the vertices in P_{uv}, that is, exchange the names of u and v and change all labels on P_{uv} to its opposite). Denote with v_s^1 and v_t^1 the first and the last vertex in the intersection of P_{uv} and P_{wx}. Analogously, denote with v_s^2 and v_t^2 the first and the last vertex in the intersection of P_{uv} and P_{wy}. Assume w.l.o.g. that P_{uv} intersects first with P_{wx} and then with P_{wy}. Observe that v_s^1 and v_s^2 are even vertices and v_t^1 and v_t^2 are odd vertices since the intersections have to start and end with edges in M_{G-X}. For an arbitrary path P and for two arbitrary vertices p_1, p_2 of P, denote by $p_1 - P - p_2$ the subpath of P from p_1 to p_2. Observe that $u - P_{uv} - v_t^1 - P_{wx} - x$ and $w - P_{wy} - v_t^2 - P_{uv} - v$ are vertex-disjoint augmenting paths. ◀

Algorithm description. We now provide the algorithm for Step 6 (see Algorithm 1 for a pseudocode). Algorithm 1 uses a table Tab which has an entry for each vertex $x \in X$ and each set $Y \subseteq X$. The table is filled in such a way that the algorithm detected for each $y \in Y$ at least $\mathrm{Tab}[x, Y]$ internally vertex-disjoint augmenting paths from x to y. The main part of the algorithm is the boolean function 'Keep-Edge' in Lines 13 to 22 which makes the decision on whether to delete an edge $\{x, v\}$ for $v \in V \setminus X$ and $x \in X$. The function works as follows for edge $\{x, v\}$: Starting at v the graph will be explored along possible augmenting paths until a "reason" for keeping the edge $\{x, v\}$ is found or further exploration is possible.

If the vertex v is free wrt. M_{G-X}, then $\{x, v\}$ is an augmenting path and we keep $\{x, v\}$ (see Line 14). Observe that in Step 4 we upper-bounded the number of free vertices by k^2 and all these vertices are leaves. Thus, we keep a bounded number of edges incident to x because the corresponding augmenting paths can end at a free leaf. We provide the exact bound below when discussing the size of the graph returned by Algorithm 1. In Line 14, the algorithm stops exploring the graph and keeps the edge $\{x, v\}$ if v has degree at least three in T. The reason is to keep the graph exploration simple by following only paths in T. This ensures that the running time for exploring the graph from x does not exceed $O(n)$. Since the number of vertices in T with degree at least three is bounded (see discussion after Observation 8), it follows that only a bounded number of such edges $\{x, v\}$ are kept.

If v is not free wrt. M_{G-X}, then it is matched with some vertex w. If w is adjacent to some leaf u in $G-X$ that is free wrt. M_{G-X}, then the path x, v, w, u is an augmenting path. Thus, the algorithm keeps in this case the edge $\{x, v\}$, see Line 16. Again, since the number of free leaves is bounded, only a bounded number of edges incident to x will be kept. If w has degree at least three in T, then the algorithm stops the graph exploration here and keeps the edge $\{x, v\}$, see Line 16. Again, this is to keep the running time at $O(kn)$ overall.

Algorithm 1: Algorithm for Step 6 of our kernelization.

Input: A matching instance $(G = (V, E), s)$, a feedback vertex set $X \subseteq V$ of size k for G with $k < \log n$ and at most $k^2(2^k + 1)$ bottommost leaves in $G - X$, and a maximum matching M_{G-X} for $G - X$ with at most k^2 free vertices in $G - X$ that are all leaves.

Output: An equivalent matching instance (G', s') such that G' contains at most $O(k^3 2^k)$ vertices and edges.

1 Fix an arbitrary bijection $f\colon 2^X \to \{1, \dots, 2^k\}$
2 **foreach** $v \in V \setminus X$ **do**
3 Set $f_X(v) \leftarrow f(N(v) \cap X)$ // The number $f_X(v) < n$ can be read in constant time.
4 Initialize a table Tab of size $k \cdot 2^k$ with $\mathrm{Tab}[x, f(Y)] \leftarrow 0$ for $x \in X, \emptyset \subsetneq Y \subseteq X$
5 $T \leftarrow$ pendant-free tree (forest) of $G - X$
6 $V_T^{\geq 3} \leftarrow$ vertices in T with degree ≥ 3
7 **foreach** $x \in X$ **do**
8 **foreach** $v \in N(x) \setminus X$ **do**
9 **if** *Keep-Edge*$(x, v) =$ *false* **then** // Is $\{x, v\}$ needed for an augmenting path?
10 delete $\{x, v\}$
11 exhaustively apply Reduction Rules 2.1 and 2.3
12 **return** (G, s).
13 **Function** *Keep-Edge*$(x \in X, v \in V \setminus X)$
14 **if** *v is free wrt. M_{G-X} or $v \in V_T^{\geq 3}$* **then return** *true*
15 $w \leftarrow$ matched neighbor of v in M_{G-X}
16 **if** *$w \in V_T^{\geq 3}$ or w is adjacent to free leaf in $G - X$* **then return** *true*
17 **if** *w has at least one neighbor in X and* $\mathrm{Tab}[x, f_X(w)] < 6k^2$ **then**
18 $\mathrm{Tab}[x, f_X(w)] \leftarrow \mathrm{Tab}[x, f_X(w)] + 1$
19 **return** *true*
20 **foreach** *neighbor $u \neq v$ of w that is matched wrt. M_{G-X} and fulfills $\{u, x\} \notin E$* **do**
21 **if** *Keep-Edge*$(u, x) =$ *true* **then return** *true*
22 **return** *false*

Let $Y \subseteq X$ denote the neighborhood of w in X. The partial augmenting path x, v, w can be extended to each vertex in Y. Thus, if the algorithm did not yet find $6k^2$ paths from x to vertices whose neighborhood in X is also Y, then the table entry $\mathrm{Tab}[x, f_X(w)]$ (where $f_X(w)$ encodes the set $Y = N(w) \cap X$) is increased by one and the edge $\{x, v\}$ will be kept (see Lines 18 and 19). The proof that $6k^2$ paths suffice is based on an exchange argument using Lemma 9. If the algorithm already found $6k^2$ "augmenting paths" from x to Y, then the neighborhood of w in X is irrelevant for x and the algorithm continues.

In Line 20, all above discussed cases to keep the edge $\{x, v\}$ do not apply and the algorithm extends the partial augmenting part x, v, w by considering the neighbors of w except v. Since the algorithm dealt with possible extensions to vertices in X in Lines 17 to 19 and with extensions to free vertices in $G - X$ in Line 14, it follows that the next vertex on this path has to be a vertex u that is matched wrt. M_{G-X}. Furthermore, since we want to extend a partial augmenting path from x, we require that u is not adjacent to x as otherwise x, u would be another, shorter partial augmenting path from x to u and we do not need the currently stored partial augmenting path.

The next lemma shows that Algorithm 1 is correct and runs in $O(kn)$ time.

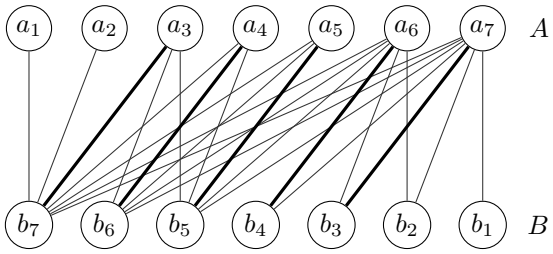

Figure 1 A chain graph. Note that the ordering of the vertices in A is going from left to right while the ordering of the vertices in B is going from right to left. The reason for these two orderings being drawn in different directions is that a maximum matching can be drawn as parallel edges, see e. g. the bold edges.

▶ **Lemma 10.** *Let $(G = (V, E), s)$ be a matching instance, let $X \subseteq V$ be a feedback vertex set of size k with $k < \log n$ and at most $k^2(2^k + 1)$ bottommost leaves in $G - X$, and let M_{G-X} be a maximum matching for $G - X$ with at most k^2 free vertices in $G - X$ that are all leaves. Then, Algorithm 1 computes in $O(kn)$ time an equivalent instance (G', s') of size $O(k^3 2^k)$.*

Simply performing Steps 1 to 6 yields the following kernel.

▶ **Theorem 11.** Matching *parameterized by the* feedback vertex number k *admits a kernel of size $2^{O(k)}$. It can be computed in $O(kn)$ time.*

Applying the $O(m\sqrt{n})$-time algorithm for Matching [20] on the kernel yields:

▶ **Corollary 12.** Matching *can be solved in $O(kn + 2^{O(k)})$ time, where k is the feedback vertex number.*

3 Kernelization for Matching on Bipartite Graphs

In this section, we investigate the possibility of efficient and effective preprocessing for Bipartite Matching. In particular, we show a linear-time computable polynomial-size kernel with respect to the parameter distance k to chain graphs. In the first part of this section, we provide the definition of chain graphs and describe how to compute the parameter. In the second part, we discuss the kernelization algorithm.

Definition and computation of the parameter. We first define chain graphs and we show that we can 4-approximate the parameter set in linear time.

▶ **Definition 13** ([5])**.** Let $G = (A, B, E)$ be a bipartite graph. Then G is a *chain graph* if each of its two color classes A, B admits a linear order w.r.t. neighborhood inclusion.

▶ **Lemma 14.** *There is a linear-time factor-4 approximation for the problem of deleting a minimum number of vertices in a bipartite graph in order to obtain a chain graph.*

Kernelization. Due to lack of space, we defer the details of our kernelization algorithm to the full version and provide in the following a high-level overview. In contrast to the kernelization in the previous section, here it is easy to bound the number of neighbors of each vertex in the deletion set X but complicated to shrink the remaining graph. Let $G = (A, B, E)$ be the given bipartite graph and let X a vertex subset such that $G - X$ is a chain graph. First compute a maximum matching M_{G-X} in $G - X$ where the edges in M_{G-X} are "parallel to each other", see Figure 1 for an illustration. Using Observation 1, we obtain the following.

▶ **Reduction Rule 3.1.** *If $|M_{G-X}| \geq s$, then return a trivial yes-instance; if $s > |M_{G-X}| + k$, then return a trivial no-instance.*

The next step of our kernelization algorithm is to bound the degree of each vertex in X. It suffices to keep for each $x \in X$ its k neighbors with smallest degree in A and in B, respectively. Due the small degree, such a vertex v is either free or matched to high-degree vertex u, see Figure 1. The correctness proof in the latter case uses the observation that there are a lot of possibilities to continue an augmenting path x, v, u as u has high degree. The reason for keeping k neighbors for each $x \in X$ is that at most $k-1$ neighbors might be "matched" (either directly or via an augmenting path) to other vertices in X.

After having bounded the degree of the vertices, we can show that for each vertex v in $G - X$ that is adjacent to a vertex in X we need to keep at most k neighbors right of v and k neighbors left of v (with respect to the linear ordering in each color class). Proving that this is indeed correct is the most technical part and relies heavily on the facts that $G - X$ is a chain graph and that the edges in M_{G-X} are "parallel".

Since for each of the k vertices in X we keep at most $2k$ neighbors in $G - X$, and for each of these neighbors, we keep $2k$ vertices, we arrive at the following.

▶ **Theorem 15.** Matching *on bipartite graphs admits a linear-time computable cubic-vertex kernel with respect to the vertex deletion distance to chain graphs.*

Applying the $O(n^{2.5})$-time algorithm for Bipartite Matching [16] on the kernel yields:

▶ **Corollary 16.** Matching *can be solved in $O(k^{7.5} + n + m)$ time, where k is the vertex deletion distance to chain graphs.*

Using the randomized $O(n^\omega)$-time bipartite matching algorithm based on matrix multiplication [21], one would obtain a randomized algorithm with running time $O(k^{3\omega} + n + m)$, where $\omega < 2.373$ is the matrix multiplication exponent.

4 Conclusion

We focused on kernelization results for Matching. In ongoing work, we are testing the practical relevance of our data reduction rules. There remain numerous challenges for future research as discussed in the second part of this concluding section. First, however, let us discuss the closely related issue of FPTP algorithms for Matching. There is a generic augmenting path-based approach to provide FPTP algorithms for Matching: One can find an augmenting path in linear time [2, 12, 20]. So the solving FPTP algorithm for Matching parameterized by some vertex deletion distance k works as follows:

1. Use a constant-factor linear-time (approximation) algorithm to compute a vertex set X such that $G - X$ is a "trivial" graph (where Matching is linear-time solvable).
2. Compute in linear time an initial maximum matching M in $G - X$.
3. Start with M as an *initial* matching in G and increase its size at most $|X| = k$ times to obtain in $O(k \cdot (n + m))$ time a *maximum* matching for G.

From this we can directly derive that Matching can be solved in $O(k(n+m))$ time, where k is one of the following parameters: feedback vertex number, feedback edge number, vertex cover number. Moreover, Bipartite Matching can be solved in $O(k(n+m))$ time, where k is the vertex deletion distance to chain graphs. Using our kernelization results, the multiplicative dependence of the running time on parameter k can now be made an additive one. For instance, in this way the running time for Bipartite Matching parameterized by vertex deletion distance to chain graphs "improves" from $O(k(n + m))$ to $O(k^{7.5} + n + m)$.

We conclude with listing some questions and tasks for future research. Can the running time of the kernelization with respect to feedback vertex set (see Section 2) be improved to linear time? Moreover, can the exponential upper bound on the kernel size be decreased to a polynomial upper bound? Is there a linear-time computable kernel for Matching parameterized by the treewidth t (assuming that t is given)? This would complement the recent randomized $O(t^4 n \log n)$ time algorithm [10]. Can one extend the kernel of Section 3 from Bipartite Matching to Matching parameterized by the distance to chain graphs?

References

1. Reuven Bar-Yehuda, Dan Geiger, Joseph Naor, and Ron M. Roth. Approximation algorithms for the feedback vertex set problem with applications to constraint satisfaction and Bayesian inference. *SIAM Journal on Computing*, 27(4):942–959, 1998. doi:10.1137/S0097539796305109.
2. Norbert Blum. A new approach to maximum matching in general graphs. In *Proceedings of the 17th International Colloquium on Automata, Languages, and Programming (ICALP '90)*, volume 443 of *LNCS*, pages 586–597. Springer, 1990. doi:10.1007/BFb0032060.
3. Hans L. Bodlaender, Bart M. P. Jansen, and Stefan Kratsch. Preprocessing for treewidth: A combinatorial analysis through kernelization. *SIAM Journal on Discrete Mathematics*, 27(4):2108–2142, 2013. doi:10.1137/120903518.
4. Hans L. Bodlaender, Bart M. P. Jansen, and Stefan Kratsch. Kernelization lower bounds by cross-composition. *SIAM Journal on Discrete Mathematics*, 28(1):277–305, 2014. doi:10.1137/120880240.
5. Andreas Brandstädt, Van Bang Le, and Jeremy P. Spinrad. *Graph Classes: a Survey*, volume 3 of *SIAM Monographs on Discrete Mathematics and Applications*. SIAM, 1999.
6. Leizhen Cai. Parameterized complexity of Vertex Colouring. *Discrete Applied Mathematics*, 127(1):415–429, 2003. doi:10.1016/S0166-218X(02)00242-1.
7. Maw-Shang Chang. Algorithms for maximum matching and minimum fill-in on chordal bipartite graphs. In *Proceedings of the 7th International Symposium on Algorithms and Computation (ISAAC '96)*, volume 1178 of *LNCS*, pages 146–155. Springer, 1996. doi:10.1007/BFb0009490.
8. Elias Dahlhaus and Marek Karpinski. Matching and multidimensional matching in chordal and strongly chordal graphs. *Discrete Applied Mathematics*, 84(1–3):79–91, 1998. doi:10.1016/S0166-218X(98)00006-7.
9. Ran Duan and Seth Pettie. Linear-time approximation for maximum weight matching. *Journal of the ACM*, 61(1):1:1–1:23, 2014. doi:10.1145/2529989.
10. Fedor V. Fomin, Daniel Lokshtanov, Michal Pilipczuk, Saket Saurabh, and Marcin Wrochna. Fully polynomial-time parameterized computations for graphs and matrices of low treewidth. In *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '17)*, pages 1419–1432. SIAM, 2017. doi:10.1137/1.9781611974782.92.
11. Harold N. Gabow and Robert Endre Tarjan. A linear-time algorithm for a special case of disjoint set union. *Journal of Computer and System Sciences*, 30(2):209–221, 1985. doi:10.1016/0022-0000(85)90014-5.
12. Harold N. Gabow and Robert Endre Tarjan. Faster scaling algorithms for general graph-matching problems. *Journal of the ACM*, 38(4):815–853, 1991. doi:10.1145/115234.115366.
13. Archontia C. Giannopoulou, George B. Mertzios, and Rolf Niedermeier. Polynomial fixed-parameter algorithms: A case study for longest path on interval graphs. *Theoretical Computer Science*, 2017. Available online. doi:10.1016/j.tcs.2017.05.017.

14 Jiong Guo, Falk Hüffner, and Rolf Niedermeier. A structural view on parameterizing problems: Distance from triviality. In *Proceedings of the 1st International Workshop on Parameterized and Exact Computation (IWPEC '04)*, volume 3162 of *LNCS*, pages 162–173. Springer, 2004. doi:10.1007/978-3-540-28639-4_15.

15 Manoj Gupta and Richard Peng. Fully dynamic (1+ e)-approximate matchings. In *Proceedings of the 54th Annual IEEE Symposium on Foundations of Computer Science (FOCS '13)*, pages 548–557. IEEE, 2013. doi:10.1109/FOCS.2013.65.

16 John E. Hopcroft and Richard M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2(4):225–231, 1973. doi:10.1137/0202019.

17 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. doi:10.1109/SFCS.1981.21.

18 Christian Komusiewicz and Rolf Niedermeier. New races in parameterized algorithmics. In *Proceedings of the 37th International Symposium on Mathematical Foundations of Computer Science (MFCS '12)*, volume 7464 of *LNCS*, pages 19–30. Springer, 2012. doi:10.1007/978-3-642-32589-2_2.

19 George B. Mertzios, André Nichterlein, and Rolf Niedermeier. Linear-time algorithm for maximum-cardinality matching on cocomparability graphs. *CoRR*, abs/1703.05598, 2017.

20 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. doi:10.1109/SFCS.1980.12.

21 Marcin Mucha and Piotr Sankowski. Maximum matchings via Gaussian elimination. In *Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science (FOCS '04)*, pages 248–255. IEEE, 2004. doi:10.1109/FOCS.2004.40.

22 Steven S. Skiena. *The Algorithm Design Manual*. Springer, 2010.

23 G. Steiner and J. S. Yeomans. A linear time algorithm for maximum matchings in convex bipartite graphs. *Comput. Math. Appl.*, 31:91–96, 1996. doi:10.1016/0898-1221(96)00079-X.

24 Ryan Williams, Carla P. Gomes, and Bart Selman. Backdoors to typical case complexity. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI '03)*, pages 1173–1178. Morgan Kaufmann, 2003.

25 Raphael Yuster. Maximum matching in regular and almost regular graphs. *Algorithmica*, 66(1):87–92, 2013. doi:10.1007/s00453-012-9625-7.