Technische Universität Berlin
Electrical Engineering and Computer Science
Institute of Software Engineering and Theoretical Computer Science
Algorithmics and Computational Complexity (AKT)

The Practical Power of Data Reduction for Maximum-Cardinality Matching

Masterarbeit

von Viatcheslav Korenwein

zur Erlangung des Grades „Master of Science“ (M. Sc.)
im Studiengang Computer Science (Informatik)

Erstgutachter:	Prof. Dr. Rolf Niedermeier
Zweitgutachter:	Prof. Dr. Thorsten Koch
Betreuer:	Dr. André Nichterlein

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und eigenhändig sowie ohne unerlaubte fremde Hilfe und ausschließlich unter Verwendung der aufgeführten Quellen und Hilfsmittel angefertigt habe.

Die selbständige und eigenhändige Anfertigung versichert an Eides statt:

Berlin, den .....

Zusammenfassung

Das Finden eines Matchings maximaler Kardinalität in ungerichteten Graphen ist ein breit untersuchtes Problem der algorithmischen Graphentheorie. Ein Matching ist eine Kantenmenge, so dass die Kanten keinen gemeinsamen Knoten haben. Seit 1980 ist bekannt, dass Matchings maximaler Kardinalität in Zeit $O(m\sqrt{n})$ gefunden werden können, wobei n die Anzahl der Knoten und m die Anzahl der Kanten in dem Graphen ist. Unlängst erschien eine Studie über den Einfluss von Datenreduktion auf die Laufzeit des Findens eines Matchings maximaler Kardinalität. Wir untersuchen die Anwendung von Datenreduktion für das Finden eines Matchings maximaler Kardinalität in der Praxis. Wir ermitteln, wie stark reale Instanzen reduziert werden, vergleichen Algorithmen mit und ohne Datenreduktion im Hinblick auf die Laufzeit und verwenden Datenreduktion auch in Kombination mit Heuristiken. Wir stellen fest, dass Datenreduktion die Ausführung von exakten Algorithmen und sogar von einigen Heuristiken beschleunigt, z. B. für einen aktuell schnellsten exakten Algorithmus ist die Beschleunigung auf verschiedenen Testgraphen bis zu einem Faktor von 90. Zudem beobachten wir eine Verbesserung der Größe von mit Heuristiken gefundenen Matchings. Außerdem übertragen wir Datenreduktionsregeln auf das Problem von Finden eines maximalen gewichteten Matchings. Insgesamt haben wir somit aus theoretischer Sicht (Quasi-)Linearzeitalgorithmen zur Berechnung von Problemkernen linearer Größe bezüglich des Parameters „Feedback Edge Number“ für die Probleme Maximum-Cardinality Matching und Maximum-Weight Matching.

Abstract

Finding a maximum-cardinality matching in undirected graphs is a well-studied problem in algorithmic graph theory. A matching is an edge set such that the edges have no common vertex. Since 1980 it is known that this problem is solvable in $O(m\sqrt{n})$ time, where n is the number of vertices and m is the number of edges in the graph. Recent work studied the influence of data reduction techniques on the running time of finding a maximum-cardinality matching. We research the applicability of data reduction for finding a maximum-cardinality matching in practice. We investigate to what extent real-world graphs are reduced in size, compare applications of algorithms with and without data reduction in terms of running time, and apply data reduction for heuristics. We ascertain that applying data reduction speeds up exact algorithms and even some heuristics, for instance the speed-up for a state-of-the-art exact algorithm on different test graphs is up to a factor of 90. Moreover, we observe that combining data reduction with heuristics improves the size of the found matching. Finally, we developed data reduction rules for the problem of finding a maximum edge-weighted matching. Altogether, on the theoretical side we thus have (quasi-)linear-time linear-size kernelizations with respect to the parameter feedback edge number both for computing maximum-cardinality matchings and maximum-weight matchings.

Contents

1. Introduction

The problem of matching objects or people occurs in many practical applications. Finding a maximum number of matching pairs of vertices is one of the central problems in algorithmic graph theory; it is known as the Maximum-Cardinality Matching problem: given an undirected graph, compute a maximum-cardinality set of pairwise disjoint edges. For example, assume that there is an international team and there is a task that should be done in pairs. The pairs can not be formed arbitrarily because not every member of the team can communicate to each other member because of language barriers. A pair is possible if two members of the team have at least one common language. All members of the team and all possible pairs can be represented with a graph. Maximum-Cardinality Matching represents the problem to find a maximum number of disjoint pairs.

An example for a graph and its two possible maximum-cardinality matchings is shown in Figure 1. Maximum-cardinality matchings have size three and all six vertices of the graph are matched. There are no other maximum-cardinality matchings in that graph, because we cannot take any edge of triangle $\{d, e\}, \{e, b\}, \{d, b\}$. If $\{d, e\}$ is in a matching, then vertex f cannot be matched and this matching cannot be maximum. Analogously matchings containing $\{e, b\}$ or $\{d, b\}$ cannot be maximum.

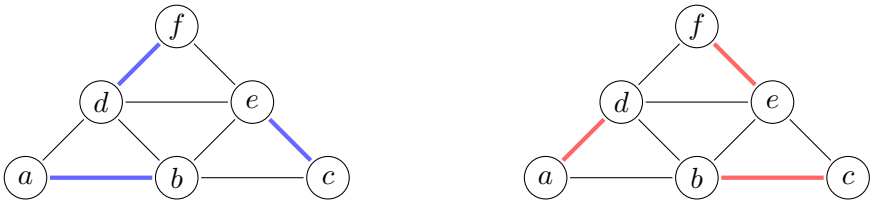
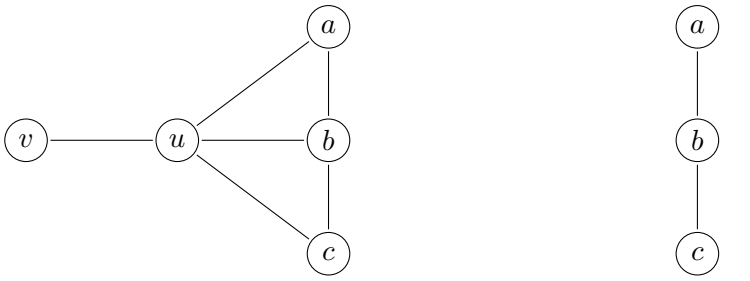


Figure 1: Sets of colored and thick edges are maximum-cardinality matchings.

We research in this work the usefulness of data reduction for Maximum-Cardinality Matching in practice. Our goal is to investigate what practical advantage data reduction rules bring. We compare applications of algorithms for finding a maximum-cardinality matching with and without data reduction in terms of overall running time. We also consider weighted graphs in the theoretical part of this work. It means that every edge of a graph has a weight. The problem to find a set of disjoint edges of maximum weight is called Maximum-Weight Matching. We research what data reduction rules could be applied for this.

Figure 2a shows an example graph on which we illustrate the application of an easy data reduction rule. It works as follows: if $\deg(v) = 1$, then delete v and its neighbor and decrease the solution size s by one. The example graph has five vertices; therefore, the maximum-cardinality matching contains at most two edges, for instance the edge set $\big\{\{a, u\}, \{b, c\}\big\}$ is a maximum-cardinality matching. We delete vertex v and its neighbor u and keep edge $\{v, u\}$ in mind. Figure 2b shows the resulting graph. This has obviously a maximum-cardinality matching of size one: $M_1 = \big\{\{a, b\}\big\}$ or $M_2 = \big\{\{b, c\}\big\}$. If we add the remembered edge $\{v, u\}$ to a maximum-cardinality matching of the resulting graph, then we get a maximum-cardinality matching of the original graph. Indeed,



(a) before (b) after

Figure 2: Example for data reduction rule.

$M_1 = \{\{a, b\}, \{v, u\}\}$ and $M_2 = \{\{b, c\}, \{v, u\}\}$ are maximum-cardinality matchings of the original graph.

Related work

For an m-edge and n-vertex graph, Edmonds [Edm65] invented a method to find a maximum-cardinality matching in an undirected graph in $O(mn^2)$ time. Subsequently Micali and Vazirani [MV80] improved this method and showed that one can find a maximum-cardinality matching in $O(m\sqrt{n})$ time. Blum [Blu90] and Gabow and Tarjan [GT91] presented alternative techniques for finding a maximum-cardinality matching, but both also run in $O(m\sqrt{n})$ time. Mucha and Sankowski [MS04] presented a randomized algorithm for finding maximum-cardinality matchings that runs in $O(n^\omega)$ time, where ω is the exponent of the best known matrix multiplication algorithm. Nowadays, ω is less than 2.3727 [Wil12].

Kolmogorov [Kol09] described a new implementation of Edmonds' algorithm for computing a perfect matching of minimum cost. This can be used to find a matching in an undirected unweighted graph.

Mertzios, Nichterlein, and Niedermeier [MNN17] investigated how to speed up the search for maximum-cardinality matchings in undirected graphs by employing a parametrized complexity analysis. Their goal was to achieve $O(f(k)(n + m))$ time, where k is some parameter. One of the considered methods was *kernelization*. This is a standard data reduction methodology to preprocess an NP-hard problem, where one "solves" the easy part of the problem in polynomial time. As we want to make use of kernelization, it has to be done faster than $O(m\sqrt{n})$ time, preferably in linear time.

Gabow [Gab90] gave an algorithm that solves Maximum-Weight Matching in $O(mn + n^2 \log n)$ time. For integer weights Gabow and Tarjan [GT91] gave an algorithm that takes $O(m\sqrt{n \log n} \log(nN))$ time, where N is the maximum edge weight. Drake and Hougardy [DH03], Maue and Sanders [MS07], and Preis [Pre99] discussed engineering heuristics for Maximum-Weight Matching.

Our results

In this thesis we study the practical side of kernelization with respect to finding a maximum-cardinality matching. We use real-world data from the Stanford Network Analysis Project (SNAP) [Les]. We implement (quasi-)linear-time data reduction rules to achieve a kernel. Our experiments show that data reduction speeds up the exact algorithms we considered. Among these exact algorithms we have a state-of-the-art implementation [Kol08]. Thus we recommend to use data reduction rules for all exact algorithms. Our experiments of

combining data reduction and heuristics show that the computation speeds up for one heuristic and slows down for another, but the sizes of the found matchings increase for both heuristics. The decision to apply data reduction rules for heuristics or not depends on the pursued aim and should be made for each heuristic separately. Moreover, we describe data reduction rules for finding a maximum-weight matching and show that Maximum-Weight Matching admits a quasi-linear-time computable linear-size kernel with respect to the parameter *feedback edge number*.

Overview of this work

Chapter 2 serves for reference of notion and notation, in particular we define matching and kernelization. We begin with the theoretical studies in Chapter 3. We describe the known theoretical results of Mertzios, Nichterlein, and Niedermeier [MNN17] for kernelization and suggest a new data reduction rule for Maximum-Cardinality Matching and give several data reduction rules for Maximum-Weight Matching. We implemented some of the theoretical results for Maximum-Cardinality Matching and derive three methods for kernelization that implement different data reduction rules or realize the same rule in different ways in Chapter 4. We do several experiments using these three methods of kernelization and describe our results in Chapter 5. In Chapter 6 we summarize and give an outlook for possible future research. Appendix A contains tables with exact results of our experiments.

2. Preliminaries

In this chapter we introduce concepts that we use in this thesis. The goal is to recall some important definitions and to have a list of notations and definitions so that the reader can use this for reference.

Notion and notation for graphs

Let $G = (V, E)$ be an undirected unweighted graph with vertex set V and edge set E. We write n for $|V|$ and m for $|E|$. We give a list of notations and definitions that are used in this thesis:

$V(G)$	The vertex set V of the graph G.
$E(G)$	The edge set E of the graph G.
Incidence	A vertex-edge pair such that the vertex is an endpoint of the edge.
Neighbors	Vertices u and v are neighbors if they are connected by edge $\{u, v\}$.
Adjacency	Two vertices are adjacent if they are neighbors.
Isolated vertex	A vertex without neighbors.
Degree	The number of neighbors of a vertex.
Δ	The *maximum degree* of G, formally, $\Delta := \max_{v \in V}\{\deg(v)\}$.
$\deg(v)$	The *degree* of vertex $v \in V$.
$N(v)$	The *neighborhood* of vertex $v \in V$, formally, $N(v) := \big\{u \in V \mid \{u, v\} \in E\big\}$.
$N(U)$	The set $\bigcup_{u \in U} N(u)$ for $U \subseteq V$.
$G - F$	The graph obtained from G by deleting the edges $F \subseteq E$, formally, $G - F := (V, E \setminus F)$.
$G - W$	The graph obtained from G by deleting the vertices $W \subseteq V$ and all edges with an endpoint in W.
$G - w$	Notation for $G - \{w\}$, where $w \in V$.
Path	A graph with vertex set $\{v_0, v_1, \ldots, v_i\}$ and edge set $\big\{\{v_0, v_1\}, \{v_1, v_2\}, \{v_2, v_3\}, \ldots, \{v_{i-1}, v_i\}\big\}$.
Cycle	A path starting and ending at the same vertex.
Connectivity	A graph is connected when there is a path between every pair of vertices.
Subgraph	A graph $G' = (V', E')$, where $V' \subseteq V$ and $E' \subseteq E$.
Connected component	A maximal connected subgraph.
Forest	An undirected acyclic graph.
Tree	A connected forest.

Independent set	A set of vertices that share no edges.
Bipartite graph	A graph whose vertex set V can be split into two disjoint and independent vertex sets.
$G = (A, B, E)$	A bipartite graph with $V = A \dot\cup B$.
Matching	A set of pairwise disjoint edges.
Matched vertex	A vertex $v \in V$ is called *matched* with respect to a matching M if v is an endpoint of an edge of M.
Free vertex	A vertex $v \in V$ is called $free$ with respect to a matching M if v is not matched with respect to M.

Graph problems

The optimization problem of matching is the following:

Maximum(-Cardinality) Matching
Input: An undirected graph $G = (V, E)$.
Task: Find a maximum-cardinality matching.

We use also the following decision version for matching:

Matching
Input: An undirected graph $G = (V, E)$ and $s \in \mathbb{N}$.
Question: Is there a matching of size at least s in G?

Besides, we introduce a weighted version of matching. We still have graph G, but each edge $e = \{u, v\}$ has a weight $w(e) = w(\{u, v\}) = w(uv) = w(vu)$ associated with it. The weight of a matching is the sum of the weights of the edges in the matching. Our goal is to find a matching with the maximum weight. We use in our work the following decision version:

Maximum-Weight Matching
Input: An undirected graph $G = (V, E)$, weights $w(e) \in \mathbb{Z}$ for all $e \in E$, and $s \in \mathbb{N}$.
Question: Is there a matching of weight at least s in G?

We give some more problems from graph theory that are used in this work.

Minimum Feedback Edge Set
Input: An undirected graph $G = (V, E)$.
Task: Find a minimum edge set whose deletion makes G acyclic.

The *feedback edge number* is the minimum number of edges whose deletion makes G acyclic.

Vertex Cover
Input: An undirected graph $G = (V, E)$ and $k \in \mathbb{N}$.
Question: Is there a subset $U \subseteq V$ of size at most k such that each edge $e \in E$ is incident to at least one vertex $u \in U$?

Kernelization

We want to make use of kernelization. This is a data reduction technique from parametrized complexity theory that is usually applied to attack NP-hard problems. We use notation from parametrized complexity [Cyg+15; DF13; FG06; Nie06]. Let (I, k) be an instance of a parametrized problem where $I \in \Sigma^*$ for a finite alphabet Σ and k is the parameter. The instances (I, k) and (I', k') are *equivalent* if (I, k) is a yes-instance if and only if (I', k') is a yes-instance.

Definition 2.1. A *kernelization* is a polynomial-time reduction from an instance (I, k) to an instance (I', k') such that

1. (I, k) and (I', k') are equivalent and
2. $|I'|, k' \leq f(k)$ for some function f.

(I', k') is called the *problem kernel*, $f(k)$ is the *size of the kernel*. A problem admits a *polynomial kernel* if $f(k) \in k^{O(1)}$.

The usual way to achieve a kernel is by applying polynomial-time *data reduction rules*. A data reduction rule is *correct* if the new instance (I', k') that results from applying the data reduction rule to an instance (I, k) is equivalent to (I, k).

3. Data reduction rules and their performance

We now survey known data reduction rules for Matching in Section 3.1. We give a linear-time computable linear-size kernel with respect to the parameter *feedback edge number*. In Section 3.2 we present and discuss a new data reduction rule for Matching based on the crown reduction rule for Vertex Cover. In Section 3.3 we develop data reduction rules for Maximum-Weight Matching. We provide a quasi-linear-time computable linear-size kernel for Maximum-Weight Matching parametrized by the *feedback edge number*.

3.1. Known data reductions rules

Mertzios, Nichterlein, and Niedermeier [MNN17] considered the following three rules for Matching:

Data Reduction Rule 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.*

Data Reduction Rule 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.*

Data Reduction Rule 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 the solution size s by one.*

An example for Data Reduction Rule 2 is visualized in Figure 3. Vertex v is deleted and vertices u and w are merged to vertex uw.

(a) Graph before the data reduction (b) Graph after the data reduction

Figure 3: Applying Data Reduction Rule 2 on vertex v.

Mertzios, Nichterlein, and Niedermeier [MNN17] proved that Data Reduction Rule 1 and Data Reduction Rule 3 can be exhaustively applied in linear time. Whether this also holds for Data Reduction Rule 2 was unknown to them. However, Bartha and Kresz [BK09] already showed that Data Reduction Rule 2 can be also exhaustively applied in linear time. In Chapter 5 we evaluate these data reduction rules experimentally.

Mertzios, Nichterlein, and Niedermeier [MNN17] showed that the reduced graph has less than $12k$ vertices and $13k$ edges, where k is the feedback edge number. With the

knowledge that Data Reduction Rule 2 can be exhaustively applied in linear time, we improve this bound and simplify their proof as follows:

Theorem 3.1. MATCHING *admits a linear-time computable* $4k$*-vertex and* $5k$*-edge kernel with respect to the parameter* feedback edge number k.

Proof. Exhaustively apply Data Reduction Rules 1 and 2 in linear time [BK09; MNN17]. Let $G = (V, E)$ be the reduced graph and F be a minimum feedback edge set of the original graph with $|F| \leq k$. We denote with V^1_{G-F}, V^2_{G-F}, and $V^{\geq 3}_{G-F}$ the sets of vertices in $G - F$ that have degree one, two, and more than two respectively. We assume that there are no vertices of degree zero in $G - F$. Otherwise, either Data Reduction Rule 1 is not exhaustively applied or the vertex with degree zero in $G - F$ is incident to an edge $f \in F$ in G. However, $F \setminus \{f\}$ is a smaller feedback edge set.

Since G has no vertices of degree zero, one, and two, $G - F$ can have vertices of degree one or two only if they are incident to an edge from F. Hence, $|V^1_{G-F}| + |V^2_{G-F}| \leq 2k$. Since $G - F$ is a forest, we have $|V^{\geq 3}_{G-F}| < |V^1_{G-F}| \leq 2k$. Thus the number of vertices in G is $|V^1_{G-F}| + |V^2_{G-F}| + |V^{\geq 3}_{G-F}| < 4k$. The number of edges in $G - F$ is $|V| - 1$ because it is a forest. Therefore, $|E| < |V| + |F| < 5k$. □

Note that the feedback edge number is usually not small. For the tested graphs in our experiments the theoretical bound for the number of vertices and edges in the kernel is larger than the number of vertices and edges in the original graph. However, the data reduction rules are quite effective in our experiments; see more about in Section 5.2, in particular Diagram 13 on page 44. To explain the experimental results, we either need to improve the theoretical bound in the above theorem or provide problem kernels with respect to smaller parameters.

3.2. A new data reduction rule

The *crown reduction rule* for VERTEX COVER [Abu+07] can be used to get a kernel with $O(k)$ vertices, where k is the size of the vertex cover. We apply it for MATCHING.

Definition 3.2. A *crown* of a graph $G = (V, E)$ is an ordered pair (H, I) of disjoint vertex subsets $H, I \subseteq V$ such that

1. $H = N(I)$,
2. I is an independent set, and
3. there is a matching between H and I in which all vertices of H are matched.

If we have such a crown in a graph, then we can apply the following data reduction rule.

Data Reduction Rule 4. *Let (H, I) be a crown. Then remove H and I and decrease the solution size s by $|H|$.*

Lemma 3.3. *Data Reduction Rule 4 is correct, that is, a reduced graph with the solution size $s - |H|$ is a yes-instance if and only if the original graph with the solution size s is a yes-instance.*

Proof. Let M be a maximum matching of size at least s for the given graph G. We exchange in M all matched edges with an endpoint in H by a maximum matching between H and I. Then we still have a maximum matching because of Property 3 of Definition 3.2.

Hence, $G[V \setminus (H \cup I)]$ has a maximum matching of size at least $s - |H|$. Conversely, let M' be a maximum matching of size $s - |H|$ for graph $G[V \setminus (H \cup I)]$. Let M'' be a maximum matching between H and I. Then $M' \dot\cup M''$ is a matching for G. From Property 3 of Definition 3.2 it follows that $|M''| = |H|$ and thus $M' \dot\cup M''$ has size at least s. □

We now give an algorithm to find a crown in a graph $G = (V, E)$. First, compute a maximal matching M in G. The endpoints of M form the set A. Let B be the complement of A. Construct a bipartite graph $G' = (A, B, E')$ with $E' \subseteq E$ and find there a maximum matching M'. Using M' compute a vertex cover X in G'. Return crown $(X \cap A, B \setminus X)$.

Algorithm 1.

1. Compute a maximal matching $M \subseteq E$.
2. If $|M| \geq s$, then return YES.
3. $A := \{v \in e \mid e \in M\}$,
   $B := V \setminus A$.
4. Construct the bipartite graph $G' = (A, B, E')$ with $E' = \big\{\{u, v\} \in E \mid u \in A,$
   $v \in B\big\}$.
5. Compute in G' a maximum matching M'.
6. If $|M'| \geq s$, then return YES.
7. Using M' compute in G' a vertex cover X.
8. $H := X \cap A$,
   $I := B \setminus X$.
9. Return crown (H, I).

Lemma 3.4. *Algorithm 1 finds a crown.*

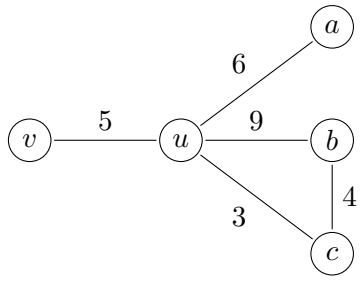
Proof. We show for (H, I) three properties from Definition 3.2. By definition, A is a set of endpoints of the maximal matching M, and B is the complement of A. Hence, B is an independent set, as otherwise the matching M would be not maximal. As $I \subseteq B$, I is an independent set as well.

We show now that all edges with an endpoint in I have second endpoint in H. Let $\{u, v\}$ be an edge with $u \in I$. Then $v \in X$ since X is a vertex cover in G' and X and I are disjoint. As $B \supseteq I$ is an independent set, $v \notin B$. Hence, $v \in A$. By definition we have $H = X \cap A$, therefore $v \in H$.

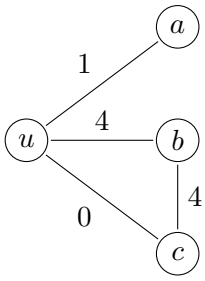
It remains to show that there is a matching between H and I in which all vertices of H are matched. Since G' is a bipartite graph, the vertex cover X contains exactly one endpoint of each edge in M'. Therefore, all vertices in H are matched by M'. Let $\{u, v\}$ be an edge in M' with $u \in H$. Then $u \in X$ and $u \in A$. Because of the partition of G' we have $v \in B$. As X is a vertex cover, $v \notin X$. Hence, $v \in I$. Thus, every edge from M' having one endpoint in H has another endpoint in I. □

Lemma 3.5. *Algorithm 1 has $O(m\sqrt{n})$ time.*

Proof. To compute a maximal matching of G, we can use a greedy algorithm that adds an edge with free endpoints to the matching. It takes linear time. A maximum matching of G' can be computed by the Hopcroft-Karp algorithm that has $O(m\sqrt{n})$ running time [HK73]. There is a linear-time algorithm to get a vertex cover from a maximum matching in a bipartite graph [BM76]. The rest takes linear time. □



(a) Graph before the data reduction



(b) Graph after the data reduction

Figure 4: Applying Data Reduction Rule 6 on vertex v.

Theorem 3.6. *Data Reduction Rule 4 is correct and can be applied once in $O(m\sqrt{n})$ time.*

Proof. The theorem follows from Lemma 3.3, Lemma 3.4, and Lemma 3.5. □

Recall that there are algorithms for finding a maximum matching in $O(m\sqrt{n})$ time [Blu90; GT91; MV80]. Thus the application of Data Reduction Rule 4 through Algorithm 1 does not improve this running time. The only time-critical step in Algorithm 1 is the computation of a maximum matching in a bipartite graph. The rest takes linear time. By using the Hopcroft-Karp algorithm [HK73], we get $O(m\sqrt{n})$ time for finding a maximum matching in a bipartite graph and consequently for Algorithm 1. If a faster algorithm for finding a maximum matching in a bipartite graph appears sometime, then Data Reduction Rule 4 will be useful. Until we have not a better running time, we subsequently do not consider Data Reduction Rule 4.

3.3. Maximum-weight matching

Now we try to extend Data Reduction Rules 1 to 3 to Maximum-Weight Matching. Recall that we only consider integer weights. We remark that we still call a matching that is inclusion-wise maximal a *maximal matching*.

Data Reduction Rule 5. *If $\deg(v) = 0$ for a vertex $v \in V$, then delete v. If $w(e) \leq 0$ for an edge $e \in E$, then delete e.*

This data reduction rule is clearly correct: If v has degree zero, then no matching can contain an edge with endpoint v and we can remove v. If e has nonpositive weight and is contained in any matching, then its deletion does not decrease the total weight of the matching. Hence, e cannot be contained in any matching and we can remove it.

Data Reduction Rule 6. *Let G be a graph with non-negative edge weights. Let v be a vertex of degree one and let u be its neighbor. Then delete v and set the weight of every edge e with an endpoint u to $\max\{0, w(e) - w(vu)\}$. Decrease the solution value s by $w(vu)$.*

Figure 4 visualizes Data Reduction Rule 6. Figure 4a shows a graph before applying the rule. The maximum-weight matching in this graph is $\{\{u, a\}, \{b, c\}\}$ and has value 10. The graph after data reduction is drawn in Figure 4b. The maximum-weight matching is again $\{\{u, a\}, \{b, c\}\}$. It has value 5, but we decrease s by $w(vu) = 5$. We show now that Data Reduction Rule 6 is correct.

Lemma 3.7. *Data Reduction Rule 6 is correct, that is, the reduced graph G' with the solution value $s - w(vu)$ is a yes-instance if and only if the original graph G with the solution value s is a yes-instance.*

Proof. Let v be a vertex of degree one and let u be its neighbor. Let M be a matching with weight at least s for the given graph G. By assumption, all edges of G have non-negative weights. We assume without loss of generality that M is inclusion-wise maximal. If v is matched, then $\{v, u\} \in M$ and the deletion of v decreases the weight of the matching by $w(vu)$. If v is not matched, then $\{v, u\} \notin M$. Then M is contained in G'. We show that the weight of M in G' is at least $s - w(vu)$. As v is not matched, u is matched. Otherwise, we could add $\{v, u\}$ to M, but this would contradict that M is maximal. Thus an edge of M loses at most $w(vu)$ on weight in G'. Hence, G' has a matching of size at least $s - w(vu)$.

Conversely, let M' be a matching in the reduced graph G' with weight at least $s - w(vu)$. If u is matched by an edge e, then set $M'' := (M' \setminus \{e\}) \cup \big\{\{u, v\}\big\}$, otherwise set $M'' := M' \cup \big\{\{u, v\}\big\}$. In both cases M'' is a matching in G and has weight at least s. □

Lemma 3.8. *Data Reduction Rule 6 can be exhaustively applied in $O(m \log \Delta)$ time.*

Proof. To reach $O(m \log \Delta)$ time, do not immediately decrease weight on all incident edges of the neighbor u of a deleted degree-one vertex v, but store this weight in a variable $r(u)$ and decrease weights after deleting all vertices of degree one at once. Otherwise, if we delete many neighbors of u during exhaustive applying of the data reduction rule, then we change weights of incident edges of u many times. An algorithm avoiding this issue performs the following steps:

1. Sort all incident edges for every vertex by weight.
2. Initialize variables $r(u)$ with zero.
3. Add all vertices of degree one to a queue.
4. Loop over the queue until the queue is empty.
   - 4.1 Take a vertex v from the queue.
   - 4.2 Let u be the neighbor of v. Add $w(vu) - r(v)$ to the value $r(u)$.
   - 4.3 Delete edges whose weight is at most $r(u)$ from the sorted list of incident edges of u and set their weights to zero.
   - 4.4 Remove v.
   - 4.5 Check if u has now degree zero or one.
     - 4.5.1 If $\deg(u) = 0$, then remove u.
     - 4.5.2 If $\deg(u) = 1$, then add u to the queue.
5. Loop over all remaining vertices $v \in V$ and set the weight of every incident edge e to $\max\{0, w(e) - r(v)\}$.

We next show that this algorithm runs in $O(m \log \Delta)$ time. The sorting of all incident edges for all vertices in step 1 takes the following time:

$$O\big(\sum_{v \in V} \deg(v) \log(\deg(v))\big) \leq O\big(\sum_{v \in V} \deg(v) \log(\Delta)\big)$$
$$= O\big(\log(\Delta) \cdot \sum_{v \in V} \deg(v)\big) = O\big(\log(\Delta) \cdot m\big).$$

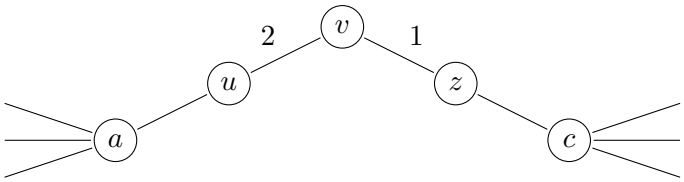


Figure 5: Example for failed attempt to extend Data Reduction Rule 2 to Maximum-Weight Matching. See also explanation after Lemma 3.8.

Since the incident edges are sorted for every vertex, the removal of all vertices of degree one in step 4 can be done in $O(m)$ time. Changing the weights of all incident edges for all remaining vertices $v \in V$ in step 5 takes also $O(m)$ time. □

Note that edges with weight zero might exist after applying Data Reduction Rule 6. The attempt to extend Data Reduction Rule 2 for degree-two vertices to Maximum-Weight Matching failed even if we have only two weights, for instance 1 and 2. Consider Figure 5. Vertex v has degree two and we want to delete v and merge its neighbors u and z. Let M be a maximal matching with weight at least s and let v be matched. Then we have to decrease the solution value s, but we do not know whether we decrease s by $w(vu)$ or by $w(vz)$. In our case these weights are different. To decrease s by 2 is wrong because it can be that $w(au) = w(uv) = w(vz) = w(zc) = 1$. To overcome this problem, we give some other data reduction rules for vertices of degree two, but before that we introduce new definitions.

Definition 3.9. A path in graph G is a *degree-2 path* if it contains at least two inner vertices, all its inner vertices have degree two in G and both its endpoints have degree at least three in G.

Definition 3.10. A cycle in graph G is a *degree-2 cycle* if all its vertices have degree two in G.

Definition 3.11. A cycle in graph G is an *almost degree-2 cycle* if exactly one of its vertices has degree at least three in G and all other vertices have degree two in G.

We denote a maximum-weight matching on path P with endpoints u and v with M^P_{uv} and its weight with m^P_{uv}. If P is clear from the context, then we omit the superscript P. Now, we introduce data reduction rules for degree 2-paths and degree-2 cycles.

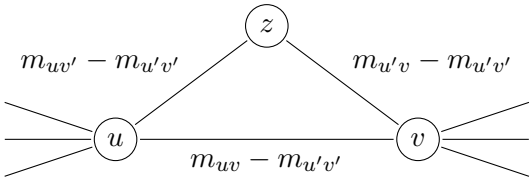
Data Reduction Rule 7. *Let G be a graph with non-negative edge weights. Let P be a degree-2 path in graph G with endpoints u and v. Let u' be the neighbor of u and v' be the neighbor of v on P. Then replace P by a triangle with vertices u, v, and a new vertex z, set $w(uz) := m_{uv'} - m_{u'v'}$, $w(zv) := m_{u'v} - m_{u'v'}$, and $w(uv) := m_{uv} - m_{u'v'}$, and decrease the solution value s by $m_{u'v'}$.*

Figure 6 visualizes Data Reduction Rule 7. We show that Data Reduction Rule 7 is correct.

Lemma 3.12. *Data Reduction Rule 7 is correct, that is, a reduced graph G' with the solution value $s - m_{u'v'}$ is a yes-instance if and only if the original graph G with the solution value s is a yes-instance.*

Proof. Let path P and vertices u, v, u', v', and z be defined as in the data reduction rule. Let M be a maximal matching with weight at least s for the given graph G. We define $\overline{M} := M \setminus E(P)$. If u and v are matched with respect to $\overline{M}$, then $\overline{M}$ is a maximal matching

(a) Graph before the data reduction



(b) Graph after the data reduction

Figure 6: Applying Data Reduction Rule 7, where m_{ij} is the value of a maximum-weight matching on the path between vertices i and j.

in G' with weight at least $s - m_{u'v'}$. If u is matched and v is free with respect to $\overline{M}$, then $\overline{M} \cup \{\{z, v\}\}$ is a maximal matching in G' with weight at least $(s - m_{u'v}) + (m_{u'v} - m_{u'v'}) = s - m_{u'v'}$. If u is free and v is matched with respect to $\overline{M}$, then $\overline{M} \cup \{\{u, z\}\}$ is a maximal matching in G' with weight at least $(s - m_{uv'}) + (m_{uv'} - m_{u'v'}) = s - m_{u'v'}$. Lastly, if vertices u and v are free with respect to $\overline{M}$, then $\overline{M} \cup \{\{u, v\}\}$ is a maximal matching in G' with weight at least $(s - m_{uv}) + (m_{uv} - m_{u'v'}) = s - m_{u'v'}$. Thus in every case there is a matching of weight at least $s - m_{u'v'}$ in G'.

Conversely, let M' be a maximal matching with weight at least $s - m_{u'v'}$ for the reduced graph G'. We define $\overline{M'} := M' \setminus \{\{u, z\}, \{z, v\}, \{u, v\}\}$. If u and v are matched with respect to $\overline{M'}$, then $\overline{M'} = M'$ and has weight $s - m_{u'v'}$. Hence, $\overline{M'} \dot\cup M_{u'v'}$ is a maximal matching in G with weight at least $(s - m_{u'v'}) + m_{u'v'} = s$. If u is matched and v is free with respect to $\overline{M'}$, then $\{z, v\} \in M'$. Hence, $\overline{M'}$ has weight at least $(s - m_{u'v'}) - (m_{u'v} - m_{u'v'}) = s - m_{u'v}$ and $\overline{M'} \dot\cup M_{u'v}$ is a maximal matching in G with weight at least $(s - m_{u'v}) + m_{u'v} = s$. If u is free and v is matched with respect to $\overline{M'}$, then $\{u, z\} \in M'$. Hence, $\overline{M'}$ has weight at least $(s - m_{u'v'}) - (m_{uv'} - m_{u'v'}) = s - m_{uv'}$ and $\overline{M'} \dot\cup M_{uv'}$ is a maximal matching in G with weight at least $(s - m_{uv'}) + m_{uv'} = s$. Lastly, if u and v are free with respect to $\overline{M'}$, then either $\{u, z\}$ or $\{z, v\}$ or $\{u, v\}$ is in M'. Hence, $\overline{M'}$ has weight at least $(s - m_{u'v'}) - \max\{w(uz), w(zv), w(uv)\} = (s - m_{u'v'}) - \max\{m_{uv'} - m_{u'v'}, m_{u'v} - m_{u'v'}, m_{uv} - m_{u'v'}\}$. As $m_{u'v'} \leq m_{uv'} \leq m_{uv}$ and $m_{u'v'} \leq m_{u'v} \leq m_{uv}$, it follows that $m_{uv} - m_{u'v'} \geq m_{uv'} - m_{u'v'}$ and $m_{uv} - m_{u'v'} \geq m_{u'v} - m_{u'v'}$. Thus $\overline{M'}$ has weight at least $(s - m_{u'v'}) - (m_{uv} - m_{u'v'}) = s - m_{uv}$ and $\overline{M'} \dot\cup M_{uv}$ is a maximal matching in G with weight at least $(s - m_{uv}) + m_{uv} = s$. Hence, in all cases there is a matching of weight at least s in G. □

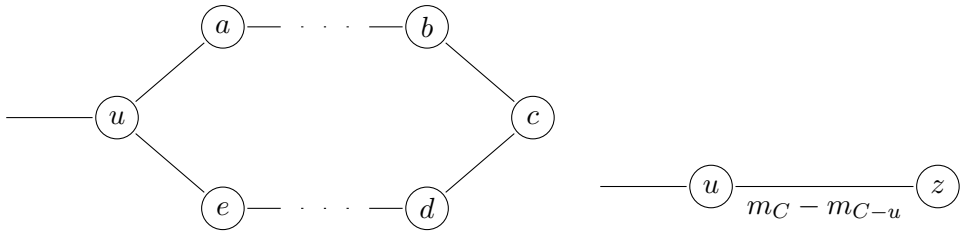
We denote in the following a maximum-weight matching of graph G with M_G and the weight of a maximum-weight matching of graph G with m_G.

Data Reduction Rule 8. *Let G be a graph with non-negative edge weights. Let C be a degree-2 cycle in graph G. Then remove C and decrease the solution value s by m_C.*

Data Reduction Rule 9. *Let G be a graph with non-negative edge weights. Let C be an almost degree-2 cycle in graph G, where $u \in C$ has degree at least three in G. Then replace C by an edge $\{u, z\}$ with $w(uz) = m_C - m_{C-u}$ and decrease the solution value s by m_{C-u}.*

Figure 7 visualizes Data Reduction Rule 9.

Lemma 3.13. *Data Reduction Rules 8 and 9 are correct.*



(a) Graph before the data reduction (b) Graph after the data reduction

Figure 7: Applying Data Reduction Rule 9, where C is the almost degree-2 cycle with vertices $a, \ldots, b, c, d, \ldots, e$, and u.

Proof. Data Reduction Rule 8 is clearly correct. Let now C be an almost degree-2 cycle in G where $u \in C$ has degree at least three in G and let G' be graph G after replacing C by an edge $\{u, z\}$ with weight $w(uz) = m_C - m_{C-u}$. We show that G' with the solution value $s - m_{C-u}$ is a yes-instance if and only if G with the solution value s is a yes-instance.

Let M be a maximal matching in G with weight at least s. We define $\overline{M} := M \setminus E(C)$. If u is matched with respect to $\overline{M}$, then $\overline{M}$ has weight at least $s - m_{C-u}$. As G' contains $\overline{M}$, G' has a matching with weight at least $s - m_{C-u}$. If u is free with respect to $\overline{M}$, then $\overline{M} \cup \big\{\{u, z\}\big\}$ is a matching in G' with weight at least $(s - m_C) + (m_C - m_{C-u}) = s - m_{C-u}$.

Conversely, let M' be a maximal matching in G' with weight at least $s - m_{C-u}$. If $\{u, z\} \in M'$, then $(M' \setminus \big\{\{u, z\}\big\}) \dot\cup M_C$ is a matching in G with weight at least $(s - m_{C-u}) - (m_C - m_{C-u}) + m_C = s$. If $\{u, z\} \notin M'$, then $M' \dot\cup M_{C-u}$ is a matching in G with weight at least $(s - m_{C-u}) + m_{C-u} = s$. □

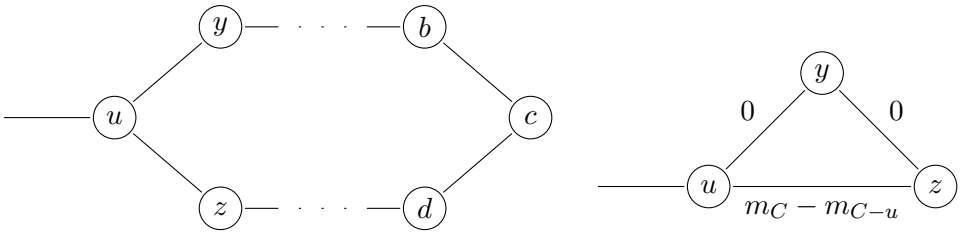
Lemma 3.14. *Data Reduction Rules 7 to 9 can be exhaustively applied in $O(n)$ time.*

Proof. First, collect in $O(n)$ time all vertices with degree two in a list. Then observe that we have to look at at most two neighbors of every vertex of degree two to find degree-2 paths, degree-2 cycles, and almost degree-2 cycles. It takes $O(|\{v \mid \deg(v) = 2\}|) \leq O(n)$ time. To apply Data Reduction Rule 7 for degree-2 paths, we have to determine weights of maximum-weight matchings for four paths. As the endpoints of a path P have degree one and all inner vertices have degree two, we can repeatedly apply Data Reduction Rule 6. Thus by Lemma 3.8, we find maximum-weight matchings in $O(|E(P)|) = O(|V(P)|)$ time.

To apply Data Reduction Rule 8 we have to determine the weight of a maximum-weight matching in a degree-2 cycle. Let $\{u, v\}$ be an edge of degree-2 cycle C. We distinguish two cases. First, let $\{u, v\}$ be in a maximum-weight matching of C. Then delete vertices u and v and keep $w(uv)$ in mind. Second, let $\{u, v\}$ not be in a maximum-weight matching of C. Then delete edge $\{u, v\}$. In both cases we get paths and can find maximum-weight matchings in $O(|V(C)|)$ time applying Data Reduction Rule 6 as before. Finally, we choose the greater weight over both cases.

To apply Data Reduction Rule 9 we have to determine the weights of maximum-weight matchings in the degree-2 cycle C and in the path $C - u$, where $u \in C$. This also take $O(|V(C)|)$ time. Thus we can apply Data Reduction Rules 7 to 9 for all degree-2 paths, degree-2 cycles, and almost degree-2 cycles in $O(n)$ time. □

Although every of Data Reduction Rules 5 to 9 is correct and can be exhaustively applied in linear or quasi-linear time, we do not know whether all these data reduction rules can be exhaustively applied in quasi-linear time. Note that after applying Data Reduction Rule 9 we can again apply Data Reduction Rule 6. To avoid this issue with the running time, we artificially weaken Data Reduction Rule 9:



(a) Graph before the data reduction (b) Graph after the data reduction

Figure 8: Applying Data Reduction Rule 10, where C is the almost degree-2 cycle with vertices $y, \dots, b, c, d, \dots, z$, and u.

Data Reduction Rule 10. *Let G be a graph with non-negative edge weights. Let C be an almost degree-2 cycle in graph G, where $u \in C$ has degree at least three in G, and let $y, z \in C$ be neighbors of u. Then replace C by an edge $\{u, z\}$ with $w(uz) = m_C - m_{C-u}$ and edges $\{u, y\}$ and $\{z, y\}$ with weights zero and decrease the solution value s by m_{C-u}.*

Figure 8 visualizes Data Reduction Rule 10. The correctness is clear since the new added edges have weight zero and do not contribute to any maximum-weight matching. Next we want to show a problem kernel. To this end we need the following auxiliary lemma:

Lemma 3.15. *Let $T = (V, E)$ be a forest without isolated vertices. Let V^1, V^2, and $V^{\geq 3}$ be the sets of vertices in T that have degree one, two, and more than two respectively. Then the sum over all degrees of vertices in $V^{\geq 3}$ is at most* $\sum_{v \in V^{\geq 3}} \deg(v) \leq 2|V^{\geq 3}| + |V^1| - 2$.

Proof. As T is a forest, it holds

$$\begin{aligned} 2(n-1) \geq 2m &= \sum_{v \in V} \deg(v) \\ &= \sum_{v \in V^1} \deg(v) + \sum_{v \in V^2} \deg(v) + \sum_{v \in V^{\geq 3}} \deg(v) \\ &= |V^1| + 2|V^2| + \sum_{v \in V^{\geq 3}} \deg(v). \end{aligned}$$

Then we have:

$$\begin{aligned} \sum_{v \in V^{\geq 3}} \deg(v) &\leq 2n - |V^1| - 2|V^2| - 2 \\ &= 2\big(|V^1| + |V^2| + |V^{\geq 3}|\big) - |V^1| - 2|V^2| - 2 \\ &= 2|V^{\geq 3}| + |V^1| - 2. \end{aligned}$$

□

We remark that if T is a tree, then we have equality $\sum_{v \in V^{\geq 3}} \deg(v) = 2|V^{\geq 3}| + |V^1| - 2$.

Now we have all tools to show a problem kernel:

Theorem 3.16. Maximum-Weight Matching *admits an $O(m \log \Delta)$-time computable $18k$-vertex and $28k$-edge kernel with respect to the parameter* feedback edge number *k.*

Proof. The kernelization algorithm is as follows:
Exhaustively apply Data Reduction Rule 5 in linear time. Then exhaustively apply Data Reduction Rule 6 in $O(m \log \Delta)$ time (see Lemma 3.8). After that exhaustively apply Data Reduction Rules 7, 8 and 10 in $O(n)$ time (see Lemma 3.14).

Let $G^* = (V^*, E^*)$ be the original graph and $G = (V \cup Z, E)$ be the reduced graph, where Z is a set of new vertices that arise due to Data Reduction Rule 7 and $V \subseteq V^*$. Let $F \subseteq E^*$ be a minimum feedback edge set in G^* with $|F| \leq k$. Observe that F has exactly one edge of each almost degree-2 cycle C. Without loss of generality, let this edge be $\{u, z\} \in C$, where u has degree at least three in G^* and $z \in N(u)$.

We denote with V_F the set of endpoints of F. We define $G' = (V, E \setminus F)$ and denote with $V^1_{G'}$, $V^2_{G'}$, and $V^{\geq 3}_{G'}$ the sets of vertices in G' that have degree one, two, and more than two respectively. Observe the following:

(1) $V_F \cap Z = \emptyset$.

(2) G' is a forest.

(3) There are no vertices of degree zero in G'.

(4) $V = V^1_{G'} \cup V^2_{G'} \cup V^{\geq 3}_{G'}$.

We show observation (3) as the other ones are obvious. Let $v \in V$ be a vertex of degree zero in G'. Then either Data Reduction Rule 5 was not exhaustively applied or v is incident to an edge $f \in F$ in G. However, the latter implies that $F \setminus \{f\}$ is a smaller feedback edge set; a contradiction.

Now we calculate the number of vertices in G'. Since G has no vertices of degree zero and one because of Data Reduction Rules 5 and 6, G' can have vertices of degree one only if they are incident to an edge from F. Hence, $V^1_{G'} \subseteq V_F$ and $|V^1_{G'}| \leq |V_F| \leq 2|F| = 2k$. Because of observation (2), we have $|V^{\geq 3}_{G'}| < |V^1_{G'}| \leq 2k$. We split set $V^2_{G'}$ into the set of vertices that are incident to an edge from F and the set of vertices that are not incident to any edge from F, that is, $V^2_{G'} = (V^2_{G'} \cap V_F) \dot\cup (V^2_{G'} \setminus V_F)$. Observe that every vertex $v \in V^2_{G'} \setminus V_F$ has degree two in the reduced graph G. Hence, v does not have neighbors in $V^2_{G'} \setminus V_F$; otherwise, Data Reduction Rules 7, 8, or 10 could be applied. Thus both neighbors of v are in V_F or in $V^{\geq 3}_{G'}$. We upper-bound $|V^2_{G'} \setminus V_F|$ with the number of neighbors of vertices in $V_F \cup V^{\geq 3}_{G'}$, divided by 2.

$$\begin{aligned}
|V^2_{G'} \setminus V_F| &\leq \frac{1}{2}\Big(N(V_F \setminus V^{\geq 3}_{G'}) + N(V^{\geq 3}_{G'})\Big) \\
&\leq \frac{1}{2}\Big(2|V_F| + \sum_{v \in V^{\geq 3}_{G'}} \deg(v)\Big) \\
&\overset{\text{Lemma 3.15}}{\leq} \frac{1}{2}\Big(2 \cdot 2|F| + 2|V^{\geq 3}| + |V^1|\Big) \\
&\leq \frac{1}{2}\Big(2 \cdot 2k + 2 \cdot 2k + 2k\Big) \\
&= 5k
\end{aligned}$$

Hence, we have the following number of vertices in G':

$$\begin{aligned}
|V| &\overset{(4)}{=} |V^1_{G'} \cup (V^2_{G'} \cap V_F) \cup (V^2_{G'} \setminus V_F) \cup V^{\geq 3}_{G'}| \\
&\leq |V^1_{G'} \cup (V^2_{G'} \cap V_F)| + |V^2_{G'} \setminus V_F| + |V^{\geq 3}_{G'}| \\
&\overset{(*)}{\leq} |V_F| + |V^2_{G'} \setminus V_F| + |V^{\geq 3}_{G'}| \\
&\leq 2k + 5k + 2k = 9k,
\end{aligned}$$

where inequality $(*)$ follows from the fact that $V^1_{G'} \subseteq V_F$ and $(V^2_{G'} \cap V_F) \subseteq V_F$.

To calculate the number of vertices in G, we still need to know the number of new vertices Z that arise due to Data Reduction Rule 7. Every vertex in Z has two adjacent neighbors in V; therefore, we can upper-bound the number of vertices in Z by number of edges in G':

$$|Z| \leq |E(G')| \overset{(2)}{\leq} |V| \leq 9k.$$

We reach now our goal and give the number of vertices and edges in G:

$$|V(G)| = |V \cup Z| = |V| + |Z| = 9k + 9k = 18k,$$

$$\begin{aligned} |E(G)| = |E| &= |E(G')| + |\big\{\{v, z\} \mid v \in V, z \in Z\big\}| + |F| \\ &\overset{(2)}{\leq} |V| + 2|Z| + |F| \\ &\leq 9k + 2 \cdot 9k + k = 28k. \end{aligned}$$

□

4. Implementation

For vertices of degree zero, one, and two we implemented Data Reduction Rules 1, 2, and 3. In this chapter we describe our implementation without linkage to a programming language. We derive three methods for kernelization that implement different data reduction rules or realize the same rule in different ways. We use these three methods for kernelization later for our experiments. The main part is explained in Section 4.1. Executing this leads to a kernel (an instance with $O(k)$ vertices and edges, where k is the feedback edge number). If we apply some algorithm to find not only the size of a maximum matching, but also the edge set in a kernel, then we have to do a postprocessing. This part is explained in Section 4.2.

4.1. Data reduction rules

In the theory we have four data reduction rules for Matching. We refrain from using Data Reduction Rule 4 because we cannot yet implement it faster than best known maximum matching algorithms (see Section 3.2). Data Reduction Rule 3 is a special case of Data Reduction Rule 2. We want to compare them with each other. Data Reduction Rule 1 can be combined with both Data Reduction Rule 2 and Data Reduction Rule 3. As we want reduce as much as possible, we do this combinations. Thus we implement two variants of kernelization:

1. Data Reduction Rule 1 and Data Reduction Rule 2.
2. Data Reduction Rule 1 and Data Reduction Rule 3.

We want to compare both variants and analyze their respective impact on different matching algorithms.

Data structure

Mertzios, Nichterlein, and Niedermeier [MNN17] showed how an algorithm which exhaustively applies Data Reduction Rule 1 and Data Reduction Rule 3 can be implemented to run in linear time. They suggested to use bucket sort to sort the vertices by degree and to keep three lists containing all degree-zero/one/two vertices. After that, Data Reduction Rule 1 and Data Reduction Rule 3 can be applied in a straightforward way but it is important to observe whether a neighbor of a deleted vertex has now degree zero, one or two. All those neighbors should be added to the corresponding lists.

We use a queue data structure and collect all vertices with degree zero, one, or two. Then we loop until the queue is empty. When the queue is empty, then we have no more vertices with degree zero, one, or two; therefore, no data reduction rule will be applicable. If we had three different lists, then we would have to move some vertices between the lists. For one common queue we do not need to update the degree of vertices which are already there. These vertices are anyway in the queue and will be removed at some point.

Case distinction and Data Reduction Rule 1

At the beginning of the loop we take a vertex v from the queue. If v was deleted from the graph in a previous iteration of the loop, then we continue with the next loop iteration. If v is in the graph, then we look at the degree of v and perform a case distinction.

If the degree of v is zero, then we remove v. This corresponds to Data Reduction Rule 1.

If the degree of v is one, then we have to remove v and its neighbor, say w. It also corresponds to Data Reduction Rule 1. Besides that, this rule says that the solution size has to be reduced by one. As we search for an optimal solution, we store the edge $\{v, w\}$. We will add $\{v, w\}$ to the matching later.

Before we remove v and w, we look at the neighbors of w. If any of them has degree three, then we add it to the queue. Otherwise, if the degree is less than three, then this vertex is already in the queue, and if it is greater than three, then we still do not need it.

If the degree of v is two, then we distinguish between Data Reduction Rule 2 and Data Reduction Rule 3. Let u and w be the neighbors of v.

Data Reduction Rule 2

We start with Data Reduction Rule 2. At first, we have to merge u and w. Assume without loss of generality that the degree of u is at least the degree of w. Then we refer to the merged vertex by u. According to the rule, the solution size has to be reduced by one. Instead, we have to store either the edge $\{v, u\}$ or the edge $\{v, w\}$. At this moment we do not know which one we can add later to the matching, therefore we store both edges. Then we remove v and the neighbor which is not a new merged vertex.

If the merged vertex has degree less than three and is not already in the queue, then we add it to the queue. To find out whether the merged vertex was already in the queue, we need only its degree before merging. If it was three or more, then the vertex was not in the queue. This is true because we merge the neighbor of v which has the greater degree. Assume that u will be the merged vertex and that it is already in the queue. This means that u has degree one or two. After deletion of v, the degree of u will be zero or one. As the degree of w was not greater than the degree of u, after deletion of v the degree of w will be zero or one too. Hence, the merged vertex will have degree at most two and will be still in the queue. Thus we add the merged vertex to the queue only if its degree was more than two before merging and is less than three now.

Data Reduction Rule 3

For Data Reduction Rule 3 we first check whether the degrees of u and w are at most two. Then we can act in two ways: we can use the method for Data Reduction Rule 2 described above or we can develop a special method for it. The difference between the special method and the method above is the way to implement the merging of u and w. In the following we describe the special method.

First, we remove the vertex v. After that, if both u and w have one neighbor, then we have the following case distinction. If the neighbor of u is w, then we remove vertex w and store the edge $\{v, w\}$. If u and w have a common neighbor, then we remove vertex w, store the edge $\{v, w\}$, and check whether the common neighbor of u and w has to be added to the queue. It can happen only if its degree became two. If u and w have different neighbors, then we merge both vertices and store the edge $\{v, w\}$.

If u has a neighbor after the deletion of v, and w has not, then we remove w and store the edge $\{v, w\}$. In the symmetric case, if w has a neighbor after the deletion of v, and u has not, and in the case if u and w have no other neighbors except v, then we remove u and store the edge $\{v, u\}$.

Methods of kernelization

Thus we have the following three methods of kernelization:

Kernelization Method 1. *Exhaustively apply Data Reduction Rule 1 and Data Reduction Rule 2.*

Kernelization Method 2. *Exhaustively apply Data Reduction Rule 1 and Data Reduction Rule 3. Use for the merging the approach of Kernelization Method 1.*

Kernelization Method 3. *Exhaustively apply Data Reduction Rule 1 and Data Reduction Rule 3. Use for the merging the special approach.*

Care has to be taken with respect to an interesting case for Data Reduction Rule 3. At the beginning all vertices of degree two are in the queue. When we delete one of them from the queue, say v, it can be that a neighbor of v has degree more than two. Then we do nothing. After deletion of its neighbor or a neighbor of its neighbor it is possible that the degree of v becomes zero or one or remains two, but both neighbors of v have degree at most two. Then we have to apply a corresponding data reduction rule for v. To do this we can easily add v to the queue again. We do it if we execute Kernelization Method 2 or 3.

While Kernelization Methods 2 and 3 have linear running time, we do not know whether our implementation of Kernelization Method 1 has a linear running time because of the merging. Experiments showed that Kernelization Method 1 was almost as fast as Kernelization Methods 2 and 3 (see Section 5.2).

4.2. Postprocessing

After the data reduction rules have been exhaustively applied, we use an algorithm or heuristic for finding a matching in the kernel. After that we have to add some edges that we stored when we reduced the graph. We have two kinds of stored edges:

1. Edges that we stored alone and wanted to add them to the matching later.
2. Pairs of edges where exactly one of the two can be added to the matching, but at the moment of storage we did not know which of them.

Edges of the first kind were stored during implementation of Data Reduction Rule 1 and of the special method for Data Reduction Rule 3. There are no obstacles to add those edges to the matching. We add easily all edges of the first kind.

Edges of the second kind were stored during implementation of Data Reduction Rule 2. We stored there two edges $e_1 = \{v, u\}$ and $e_2 = \{v, w\}$, removed the vertex v, and merged u and w. We should now decide which of two edges e_1 and e_2 can be added to the matching. If u is not matched, then we add e_1 to the matching because the common vertex v was removed from the graph and could not be matched. Otherwise, u is matched. As u and w were merged, w is not matched, and we add e_2 to the matching.

5. Experimental evaluation

In this chapter we describe our experiments and compare the kernelization methods with each other with respect to the number of reduced vertices and edges and with respect to the computation time. After that we discuss the computation of finding a maximum matching with two exact algorithms and two heuristics. We compare the computation times with and without kernelization. For heuristics we additionally compare the sizes of the found matchings, with and without kernelization.

5.1. Setup of the experiments

Hardware and software specification

We implemented the algorithms in Java using the JGraphT library [Nav79] and ran them under the OpenJDK runtime environment in version 1.8.0_131. Besides that, we used C++ 98/03 for a maximum matching algorithm implemented by Kolmogorov [Kol08] (see Section 5.4). 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. Our program can be found on `http://fpt.akt.tu-berlin.de/software/MatchingKernelization.zip`.

Test instances

We tested our implementation on graphs from the Stanford Network Analysis Project (SNAP) [Les] because it is an established data set for graphs that was often used and there are different graphs from different applications there. We first converted the files, that is, we did changes on files in order to read a graph with a method from the JGraphT library. In this conversion we deleted loops, i. e. edges with the same vertex on both endpoints. Table 1 contains all graphs that we used for our experiments, with exact numbers of their vertices and edges. Table 8 in Appendix A contains the conversion time and time needed for creating a graph from the conversion file.

Program parameters

In our program we use several input parameters. A directory with files of test graphs will be given with the parameter *list*. In the parameter *output file* we provide a path and a name of the file that will contain results after the computation.

We apply different algorithms and heuristics to find a maximum matching. With the parameter *algorithm* we fix which one we use.

We set the limit in milliseconds for the computation time of one graph with the parameter *time limit*. If the time exceeds the given limit, then the program will abort. To process as many graphs as possible before an abort, we sort the test graphs on file sizes in ascending order. Table 1 contains this sorting. We do not sort on number of vertices or edges because we should first create graph to find out how many vertices and edges this has.

Table 1: Test graphs

graph	file	vertices	edges	file size [KB]
G_1	as20000102.txt	6,474	12,572	112
G_2	p2p-Gnutella08.txt	6,301	20,777	210
G_3	p2p-Gnutella09.txt	8,114	26,013	266
G_4	p2p-Gnutella06.txt	8,717	31,525	325
G_5	p2p-Gnutella05.txt	8,846	31,839	329
G_6	CA-GrQc.txt	5,241	14,484	343
G_7	p2p-Gnutella04.txt	10,876	39,994	421
G_8	p2p-Gnutella25.txt	22,687	54,705	631
G_9	CA-HepTh.txt	9,875	25,973	643
G_{10}	p2p-Gnutella24.txt	26,518	65,369	761
G_{11}	p2p-Gnutella30.txt	36,682	88,328	1,052
G_{12}	Wiki-Vote.txt	7,115	100,762	1,069
G_{13}	p2p-Gnutella31.txt	62,586	147,892	1,809
G_{14}	CA-CondMat.txt	23,133	93,439	2,363
G_{15}	CA-HepPh.txt	12,006	118,489	2,949
G_{16}	Email-Enron.txt	36,692	183,831	3,954
G_{17}	Email-EuAll.txt	265,009	364,481	4,884
G_{18}	CA-AstroPh.txt	18,771	198,050	5,160
G_{19}	Cit-HepTh.txt	27,769	352,285	5,446
G_{20}	soc-Epinions1.txt	75,879	405,740	5,530
G_{21}	Cit-HepPh.txt	34,546	420,877	6,547
G_{22}	Slashdot0902.txt	82,168	504,230	11,010
G_{23}	com-amazon.ungraph.txt	334,863	925,872	12,290
G_{24}	com-dblp.ungraph.txt	317,080	1,049,866	13,604
G_{25}	Amazon0302.txt	262,111	899,792	16,839
G_{26}	web-NotreDame.txt	325,729	1,090,108	21,049
G_{27}	web-Stanford.txt	281,903	1,992,636	32,117
G_{28}	com-youtube.ungraph.txt	1,134,890	2,987,624	37,813
G_{29}	roadNet-PA.txt	1,088,092	1,541,898	45,020
G_{30}	Amazon0601.txt	403,394	2,443,408	46,731
G_{31}	roadNet-TX.txt	1,379,917	1,921,660	57,846
G_{32}	WikiTalk.txt	2,394,385	4,659,565	64,910
G_{33}	web-Google.txt	875,713	4,322,051	73,613
G_{34}	roadNet-CA.txt	1,965,206	2,766,607	85,729
G_{35}	web-BerkStan.txt	685,230	6,649,470	107,554
G_{36}	as-skitter.txt	1,696,415	11,095,298	145,611
G_{37}	cit-Patents.txt	3,774,768	16,518,947	273,963

5.2. Comparison of kernelization methods

Before we begin to apply any matching algorithms to compare the computation time of finding a maximum matching with and without kernelization, we look by how much and how fast the graphs are reduced by the three kernelization methods (see Section 4.1).

Power of data reduction

Since Kernelization Methods 2 and 3 applied the same data reduction rules and differ only in the implementation of merging method, the number of reduced vertices and edges is the same. In the following we do not consider the data reduction of Kernelization Method 3 separately. All results and conclusions for data reduction with respect to Kernelization Method 2 hold for Kernelization Method 3. However, note that the resulting kernels may vary in their structure.

Diagram 9 shows the number of vertices and edges in every tested graph and number of vertices and edges after Kernelization Methods 1 and 2. For a better view vertices and edges are drawn separately. We see that some graphs are reduced a lot, e. g. G_{32}, and some a little, e. g. G_{25}. Kernelization Method 1 is more powerful. This is not surprising because it applies Data Reduction Rule 2 and Kernelization Method 2 applies Data Reduction Rule 3 which is a special case of Data Reduction Rule 2.

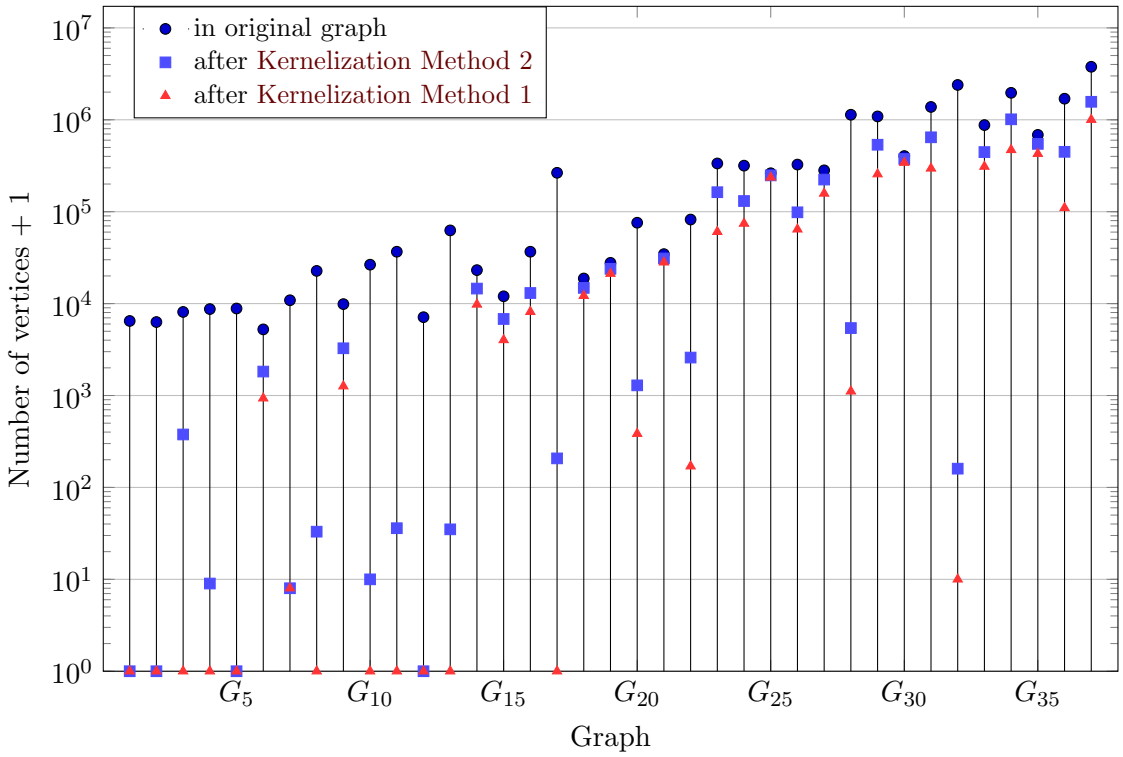
Diagram 10 shows the percentage of vertices and edges remaining after Kernelization Methods 1 and 2. We can see that the kernel of many graphs after Kernelization Method 1 is an empty set. It happens for relatively small graphs $G_1, \ldots, G_5$, but also for a bigger graph G_{17}. The kernel of the relatively big graph G_{32} has only 9 vertices and 15 edges or less than 0.001% of all vertices and edges. Kernelization Method 2 is less effective, but it gives a powerfull reduction for some instances as well. For instance, look again at graph G_{32}. The kernel has only 0.01% vertices and 0.01% edges of the original graph.

Some instances cannot be reduced successfully neither with Kernelization Method 1 nor with Kernelization Method 2. This can be observed for graph G_{25}, for which more than 90% vertices and edges remain after Kernelization Method 1 and about 95% vertices and edges remain after Kernelization Method 2. Recall that we implemented data reduction rules for vertices with degree 0, 1, and 2. However, not every graph has such vertices. Thus it can happen that our kernelization methods have no effect.

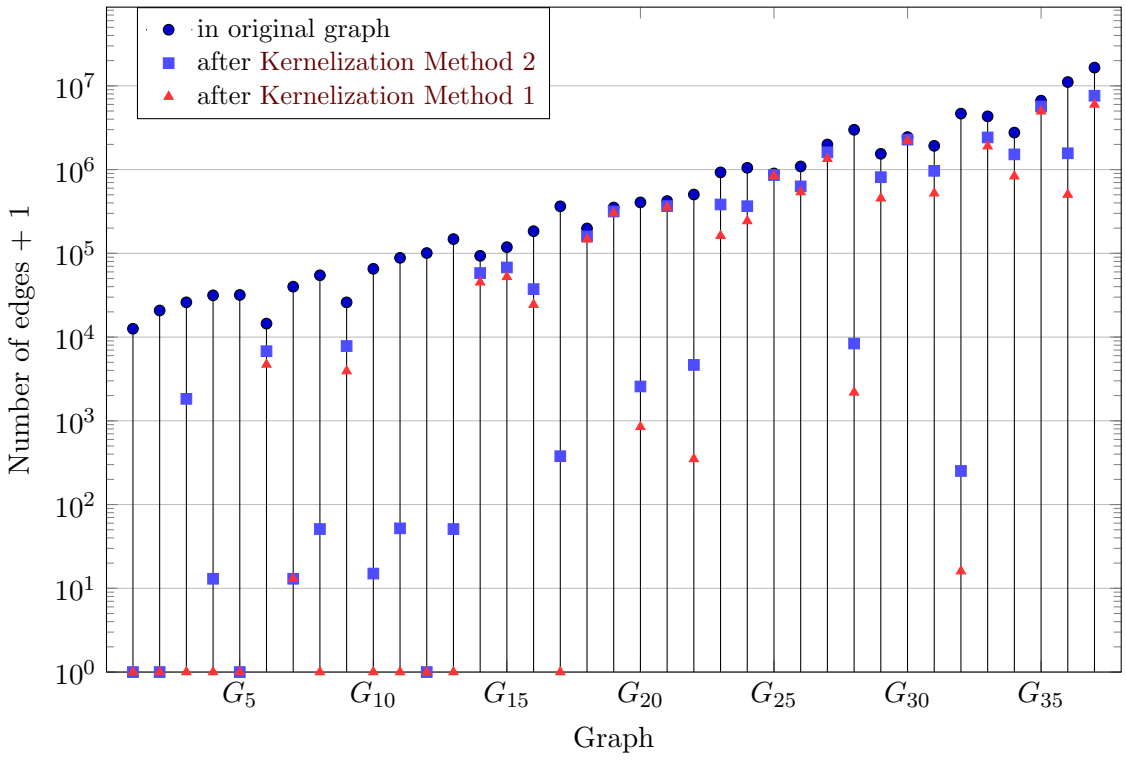
Computation time

Now we compare the computation times of all three kernelization methods. We know from Section 4.1 that Kernelization Methods 2 and 3 have a linear running time and we do not know whether our implementation of Kernelization Method 1 has a linear running time. We expect that executing Kernelization Method 1 needs more time because Kernelization Methods 2 and 3 only apply in a subset of cases where Kernelization Method 1 applies.

Diagram 11 visualizes the computation time for all test graphs. Indeed, Kernelization Method 1 is almost everywhere slower than Kernelization Methods 2 and 3. The only exceptions are graphs G_{17}, G_{28}, and G_{32}. The computation time of Kernelization Method 1 for G_{17} is 20 milliseconds faster than the computation time of Kernelization Method 2 and 3 milliseconds faster than the computation time of Kernelization Method 3. The kernel of G_{17} has no vertices and of course no edges after Kernelization Method 1 and only 0.08% vertices and 0.1% edges after Kernelization Methods 2 or 3. Thus the time consumption of all three kernelization methods is almost equal. A similar behavior can be observed for G_{28} and G_{32}.

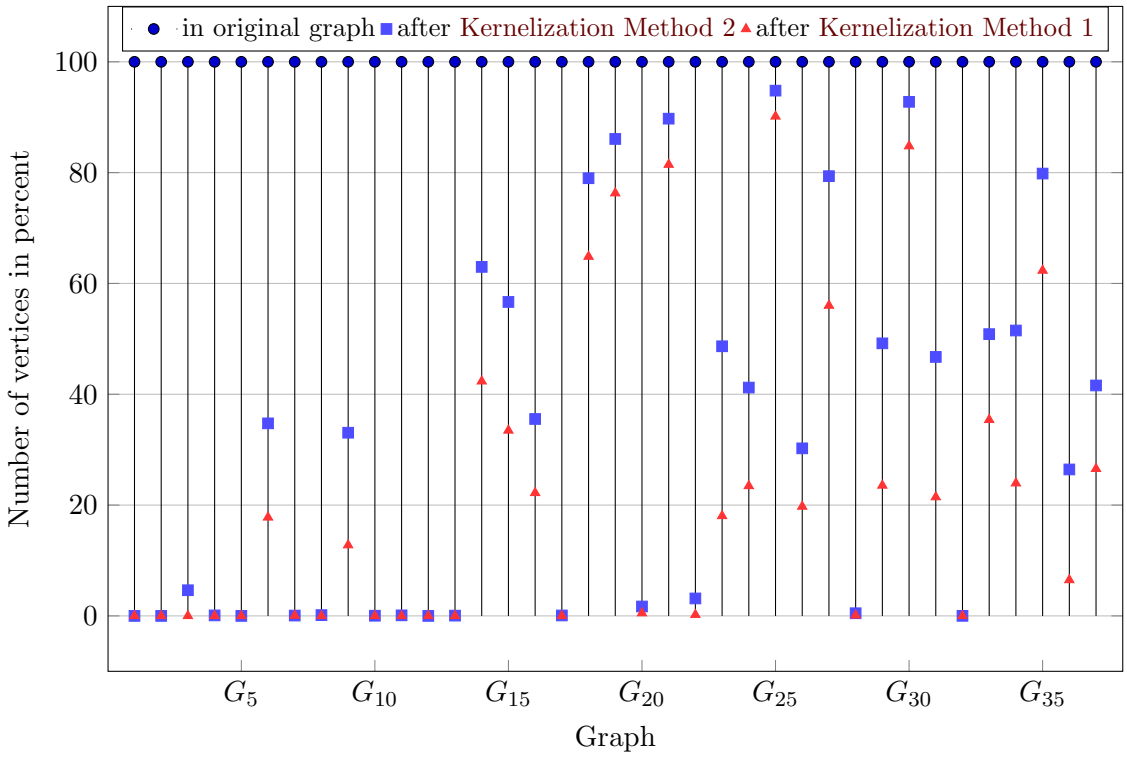


(a) Vertices

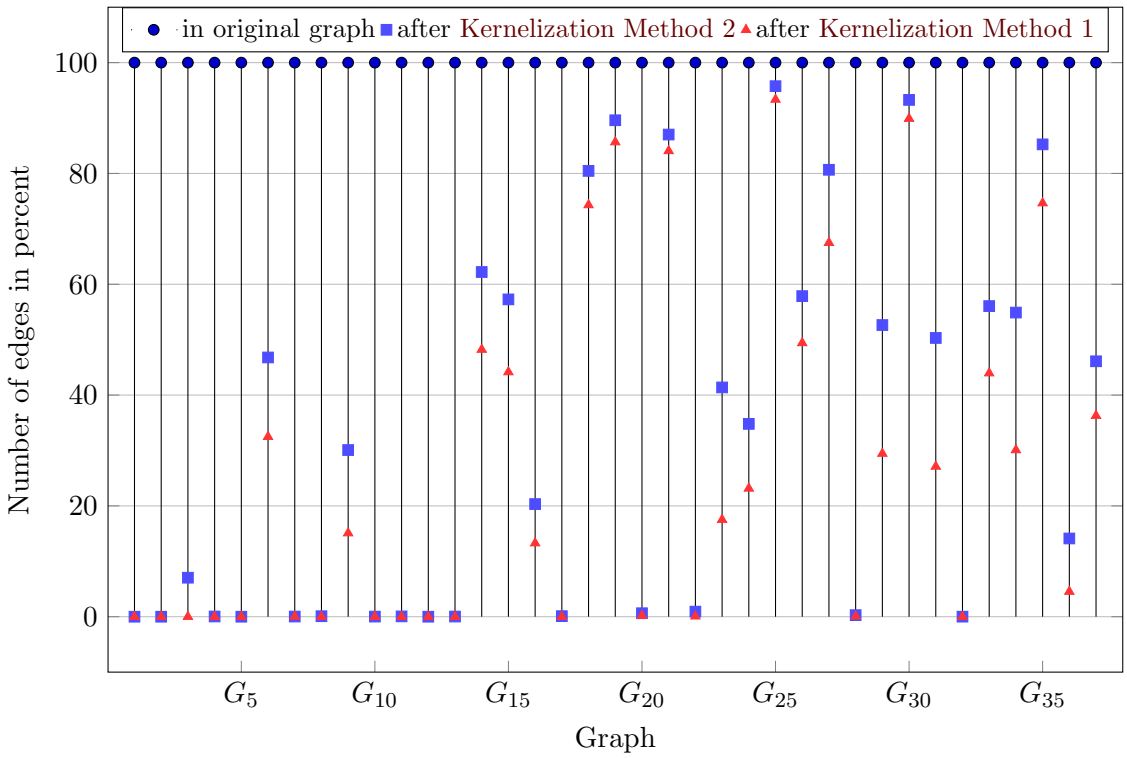


(b) Edges

Diagram 9: Number of vertices and edges before and after kernelization. Since points with value 0 cannot be drawn on logarithmic axis, we add 1 to all values. The exact numbers are in Table 9 in Appendix A. See also Diagram 10 for the data reduction in percent.



(a) Vertices



(b) Edges

Diagram 10: Number of vertices and edges before and after kernelization methods in percent, where 100 percent is the number of vertices or the number of edges in a graph. The exact numbers are in Table 10 in Appendix A.

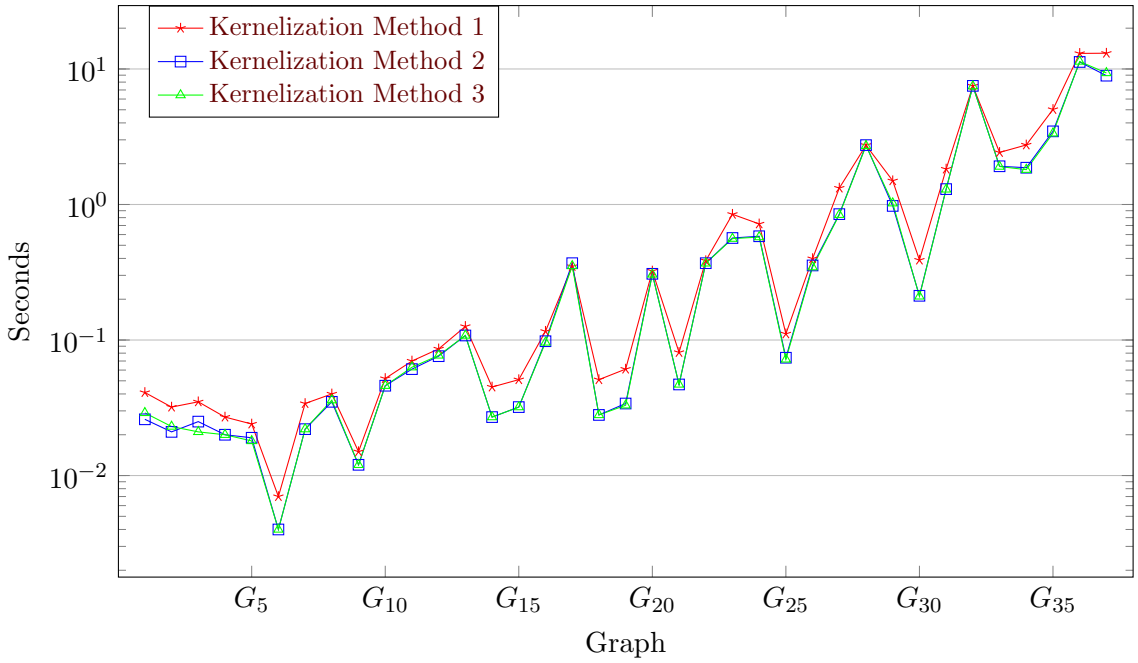


Diagram 11: Computation time of kernelization. The exact numbers are in Table 11 in Appendix A.

For all 37 instances executing Kernelization Method 1 is less than 1.9 times slower than executing Kernelization Methods 2 or 3. We can see in Diagram 11 that for many graphs the computation of a kernel lasts less than one second. The longest time consumption of Kernelization Method 1 amounts 13.07 seconds for graph G_{37}. This graph has 3.7 million vertices and 16.5 million edges and 26.6% of these vertices and 36.3% of these edges remain in the kernel. We observe a similar picture for Kernelization Methods 2 and 3. The longest time consumption for both kernelization methods is 11 seconds for the graph G_{36}, which has almost 1.7 million vertices and almost 11.1 million edges. Its kernel has 26.4% vertices and 14.1% edges. In the next sections we evaluate whether this effort pays off.

Postprocessing

Together with the computation time of the kernelization we have to measure the computation time of the postprocessing (see Section 4.2) because it is a part of the kernelization. However, the postprocessing depends also on the found matchings for the kernels. Thus the postprocessing depends on the matching algorithms used. In the next sections we evaluate several algorithms which find a maximum matching. We will see that the postprocessing lasts less than 0.4 seconds for all tested instances (see Table 6 on page 49 and Tables 18, 19 and 24 to 26 in Appendix A). Therefore, we can consider the computation time of postprocessing as negligible.

After executing any of the three kernelization methods we know how many edges will be added to the matching in the postprocessing. We find the sizes of maximum matchings in the tested graphs with exact algorithms and use them to estimate how many edges of a maximum matching are found in the kernelization. Exact algorithms will be considered thoroughly in Sections 5.3 and 5.4. Table 2 contains for every tested graph how many edges are in the graph, how many edges are in a maximum matching and how many edges of a maximum matching are found and reserved during every kernelization method.

Table 2: Matchings. See also Diagram 12.

graph	number of edges	size of a maximum matching	edges of a maximum matching found during Kernelization Method 1	edges of a maximum matching found during Kernelization Methods 2 or 3
G_1	12,572	1,048	1,048	1,048
G_2	20,777	2,054	2,054	2,054
G_3	26,013	2,574	2,574	2,523
G_4	31,525	3,405	3,405	3,403
G_5	31,839	3,428	3,428	3,428
G_6	14,484	2,329	1,876	1,448
G_7	39,994	4,348	4,345	4,345
G_8	54,705	6,017	6,017	6,008
G_9	25,973	4,457	3,838	2,864
G_{10}	65,369	7,208	7,208	7,206
G_{11}	88,328	9,268	9,268	9,255
G_{12}	100,762	2,249	2,249	2,249
G_{13}	147,892	15,693	15,693	15,681
G_{14}	93,439	10,970	6,114	3,779
G_{15}	118,489	5,649	3,655	2,284
G_{16}	183,831	12,198	8,236	5,848
G_{17}	364,481	18,315	18,315	18,297
G_{18}	198,050	9,150	3,072	1,753
G_{19}	352,285	13,746	3,153	1,800
G_{20}	405,740	21,960	21,780	21,365
G_{21}	420,877	17,150	3,083	1,652
G_{22}	504,230	25,565	25,488	24,696
G_{23}	925,872	148,485	119,471	71,653
G_{24}	1,049,866	138,074	101,878	74,736
G_{25}	899,792	130,822	12,685	6,597
G_{26}	1,090,108	66,160	38,341	30,418
G_{27}	1,992,636	108,120	44,594	18,040
G_{28}	2,987,624	274,331	273,809	271,817
G_{29}	1 541 898	528,802	400,773	261,689
G_{30}	2,443,408	201,272	30,280	14,158
G_{31}	1,921,660	669,753	521,943	348,177
G_{32}	4,659,565	56,063	56,059	56,011
G_{33}	4,322,051	301,210	169,559	121,079
G_{34}	2,766,607	955,206	720,115	450,116
G_{35}	6,649,470	245,311	83,361	36,212
G_{36}	11,095,298	513,304	474,868	355,760
G_{37}	16,518,947	1,591,723	1,093,667	827,787

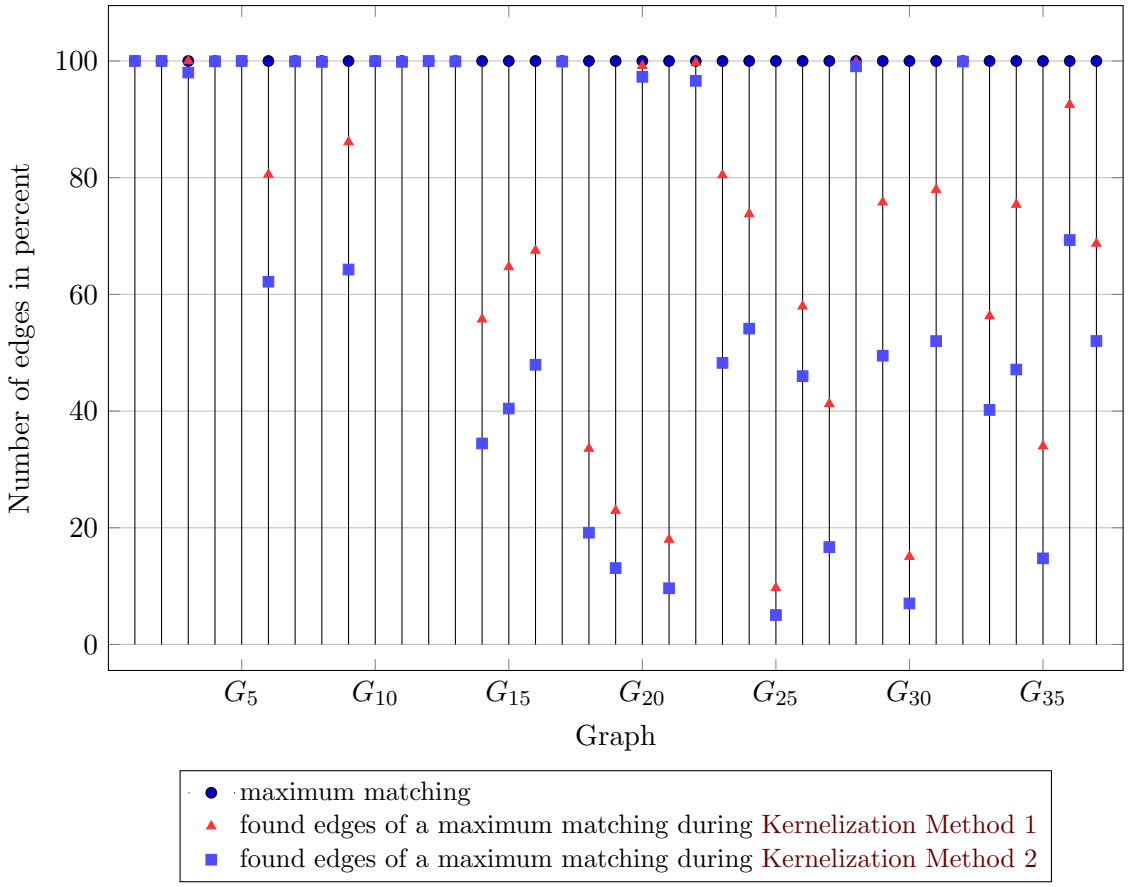


Diagram 12: Percent of edges of a maximum matching found during kernelizations. The exact numbers are in Table 12 in Appendix A.

Diagram 12 visualizes the percentage of edges of maximum matchings Kernelization Methods 1 and 2 find. Recall that Kernelization Methods 2 and 3 reduce equally. Hence, the found numbers of edges during Kernelization Methods 2 and 3 are the same. Diagram 12 shows similar results as Diagram 10. It is clear that for the graphs which have an empty kernel, all edges of a maximum matching are found during the kernelization. The graphs which are reduced a little have a lot edges in the kernel. Hence, a great part of a maximum matching is in the kernel and has to be found with any matching algorithm. For instance, graph G_{25} has 93,34% of edges in the kernel after Kernelization Method 1 and only 9,7% of edges of a maximum matching are found during the kernelization.

Thus our observations for the data reduction and for the computation time of the three kernelization methods hold for the whole procedure of kernelization including postprocessing. We finish this section with concluding that executing Kernelization Method 1 is not significantly slower than executing Kernelization Methods 2 or 3, but its kernel is smaller for all tested graphs.

Graph parameters

We measured the following graph parameters in the original graphs and in the kernels after Kernelization Method 1:

Vertex cover number	The size of a minimum vertex cover.
Feedback edge number	The size of a minimum feedback edge set.

Feedback vertex number	The size of a minimum set of vertices whose deletion makes G acyclic.
Degeneracy	The minimum number k such that G is k-degenerate, where a k-degenerate graph is an undirected graph in which every subgraph has a vertex of degree at most k.

and the number of connected components.

We used *Graphana* tool to compute or estimate graph parameters. The software is available from `http://fpt.akt.tu-berlin.de/graphana`. Values for feedback edge number, number of connected components, and degeneracy are optimal while values for vertex cover number and feedback vertex number are approximated and can be twice as large as the optimum values. All computed values are in Tables 3 and 4.

We used the parameter feedback edge number in Theorem 3.1 (page 18) and showed that MATCHING admits a $4k$-vertex and $5k$-edge kernel with respect to feedback edge number k. We now calculate this theoretical bound for the kernel size and compare this with the actual kernel size. For the comparison we take kernels after Kernelization Method 1 because their sizes are smaller than sizes of kernels computed with Kernelization Methods 2 or 3. This comparison is visualized in Diagram 13. We also drew there the number of vertices and edges in the original graphs. We see that the theoretical bound is not useful for our real-world instances. For each of the tested graphs the theoretical bound for the number of vertices and edges in the kernel is even larger than the number of vertices and edges in the original graph.

5.3. Edmonds' Blossom Shrinking

We begin to describe the results of executing the kernelization in combination with a matching algorithms. The JGraphT library has several methods to compute a matching. Only one of them is intended for constructing an optimal matching on general graphs. It is Edmonds' Blossom shrinking algorithm [Edm65] that runs in $O(n^4)$ time. We applied it for our first experiments.

We compare finding a maximum matching with and without kernelization. We first measured the computation time needed to find a maximum matching only with Edmonds' Blossom Shrinking algorithm. Then we measured the computation time needed for kernelization, finding a maximum matching in the kernel with Edmonds' Blossom Shrinking algorithm and postprocessing. We used one by one all three methods of kernelization (see Section 4.1).

We set the time limit to 32 hours per instance which includes four ways for the computation of a maximum matching: without kernelization and with the three kernelization methods. We got results for graphs $G_1, \ldots, G_{28}$. The computation for the remaining graphs did not finish within the time limit. Results are in Table 5.

Comparison between computations with and without kernelization

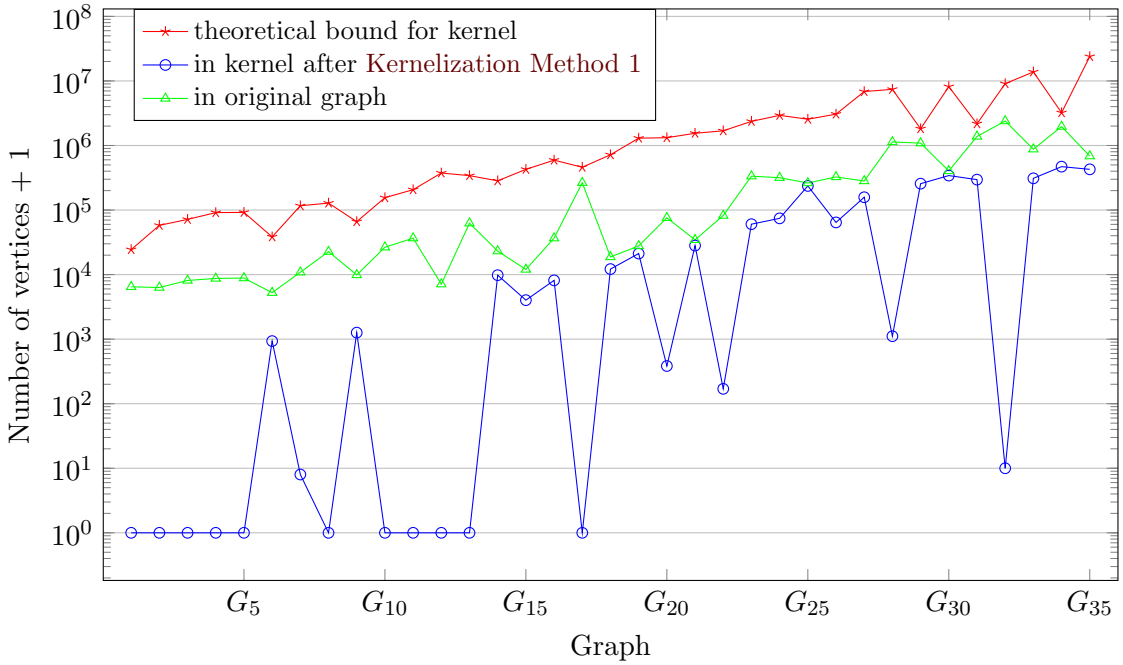
We see that the computations with any of the kernelization methods are faster than the computation without kernelization as visualized in Diagram 14. Four ratios are drawn here: the computation times of finding a maximum matching without kernelization and with Kernelization Methods 1, 2, and 3 divided by the computation time of finding a maximum matching without kernelization. The ratio for the computation without kernelization is of course constant and has value 1. The three other ratios for the computations

Table 3: Graph parameters in the original graphs and in the kernels after Kernelization Method 1. We denote with VCN the vertex cover number, with FEN the feedback edge number, with FVN the feedback vertex number, with CC the number of connected components, with DEG the degeneracy, and with ∅ an empty kernel. Vertex cover number and feedback vertex number are upper bounded by factor 2. Parameters for graphs $G_{19}, \ldots, G_{35}$ are in Table 4.

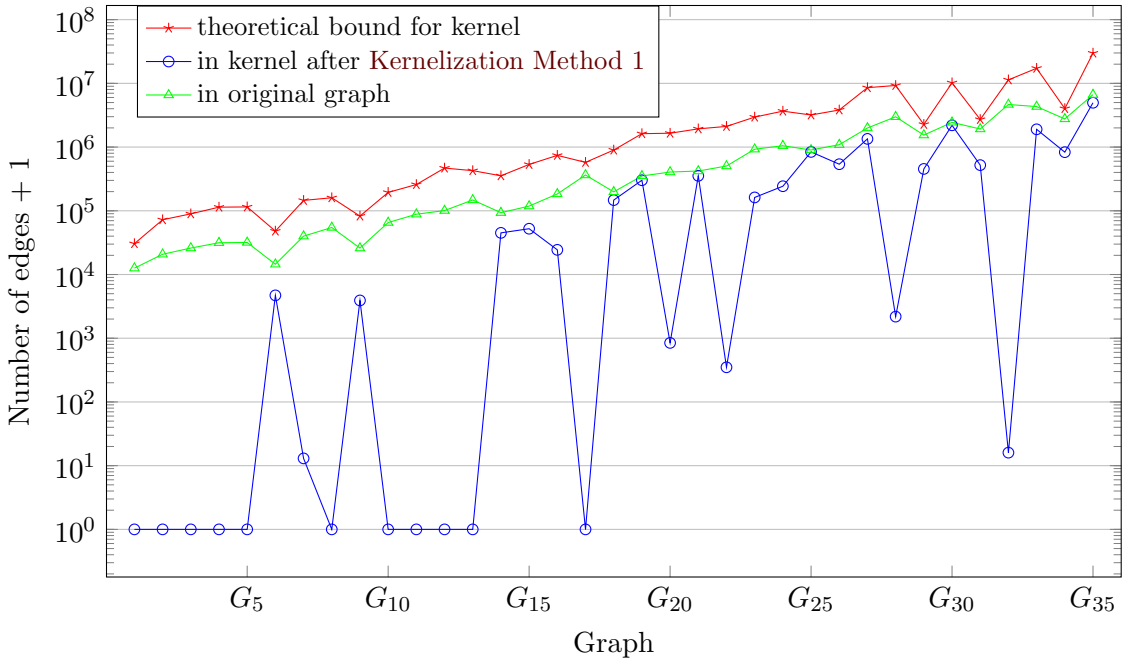
Graph	in	VC	FEN	FVS	CC	DEG
G_1	original	1,236	6,099	218	1	12
	kernel			∅		
G_2	original	2,091	14,478	1,288	2	10
	kernel			∅		
G_3	original	2,619	17,905	1,651	6	10
	kernel			∅		
G_4	original	3,425	22,809	2,259	1	9
	kernel			∅		
G_5	original	3,469	22,996	2,248	3	9
	kernel			∅		
G_6	original	3,139	9,597	1,450	354	43
	kernel	730	3,840	557	67	34
G_7	original	4,411	29,119	2,836	1	7
	kernel	3	6	2	1	3
G_8	original	6,137	32,031	3,724	13	5
	kernel			∅		
G_9	original	5,532	16,525	2,505	427	31
	kernel	891	2,733	597	76	29
G_{10}	original	7,360	38,862	4,423	11	5
	kernel			∅		
G_{11}	original	9,437	51,658	5,728	12	7
	kernel			∅		
G_{12}	original	2,277	93,671	1,666	24	53
	kernel			∅		
G_{13}	original	15,969	85,318	9,552	12	6
	kernel			∅		
G_{14}	original	14,196	70,873	8,316	567	25
	kernel	6,995	35,447	4,904	194	24
G_{15}	original	7,374	106,759	4,624	276	238
	kernel	3,157	48,370	2,523	71	152
G_{16}	original	16,090	148,204	7,851	1,065	43
	kernel	5,579	16,799	3,599	559	14
G_{17}	original	31,667	115,103	1,189	15,631	37
	kernel			∅		
G_{18}	original	12,287	179,568	8,627	289	56
	kernel	8,790	135,048	6,724	64	53

Table 4: Graph parameters in the original graphs and in the kernels after Kernelization Method 1. We denote with VCN the vertex cover number, with FEN the feedback edge number, with FVN the feedback vertex number, with CC the number of connected components, with DEG the degeneracy, and with – no results within the time limit. Vertex cover number and feedback vertex number are upper bounded by factor 2. Parameters for graphs $G_1, \ldots, G_{18}$ are in Table 3.

Graph	in	VC	FEN	FVS	CC	DEG
G_{19}	original	18,322	324,658	12,718	142	37
	kernel	14,966	280,539	11,019	1	36
G_{20}	original	23,845	329,863	7,857	2	67
	kernel	269	523	187	62	7
G_{21}	original	22,055	386,392	15,502	61	30
	kernel	18,828	325,641	13,521	2	27
G_{22}	original	26,774	422,063	10,683	1	55
	kernel	105	209	72	29	6
G_{23}	original	163,669	591,010	–	1	6
	kernel	36,844	103,805	23,335	2,098	6
G_{24}	original	178,594	732,787	–	1	113
	kernel	55,475	172,574	39,285	3,940	94
G_{25}	original	172,380	637,682	–	1	6
	kernel	159,085	603,634	–	30	6
G_{26}	original	76,858	764,380	35,484	1	155
	kernel	37,017	474,799	23,107	596	155
G_{27}	original	125,052	1,711,098	–	365	71
	kernel	77,014	1,187,918	–	1,066	68
G_{28}	original	297,420	1,852,735	–	1	51
	kernel	789	1,300	543	228	8
G_{29}	original	579,396	454,012	–	206	3
	kernel	155,923	197,993	76,350	269	3
G_{30}	original	271,250	2,040,021	–	7	10
	kernel	239,556	1,854,514	–	12	10
G_{31}	original	733,153	542,167	–	424	3
	kernel	180,755	225,614	–	588	3
G_{32}	original	67,567	2,267,735	–	2,555	131
	kernel	6	8	4	2	3
G_{33}	original	355,364	3,449,084	–	2,746	44
	kernel	174,552	1,594,437	–	4,730	42
G_{34}	original	1,052,587	804,039	–	2,638	3
	kernel	287,781	362,230	–	534	3
G_{35}	original	288,261	5,964,916	–	676	201
	kernel	198,587	4,538,229	–	2,168	198



(a) Vertices



(b) Edges

Diagram 13: The number of vertices and edges in kernel: the theoretical bound (see Theorem 3.1 on page 18) and the empirically measured values with Kernelization Method 1. For comparison the number of vertices and edges in the original graphs are also given. Since points with value 0 cannot be drawn on logarithmic axis, we add 1 to all values. The exact numbers are in Table 13 in Appendix A.

Table 5: Computation time for Edmonds' Blossom shrinking algorithm

graph	matching size	without kernelization [s]	with Kernelization Method 1 [s]	with Kernelization Method 2 [s]	with Kernelization Method 3 [s]
G_1	1,048	0.836	0.009	0.008	0.008
G_2	2,054	1.203	0.020	0.016	0.016
G_3	2,574	2.041	0.036	0.066	0.051
G_4	3,405	2.900	0.033	0.024	0.024
G_5	3,428	2.998	0.029	0.023	0.023
G_6	2,329	1.112	0.092	0.256	0.257
G_7	4,348	4.049	0.032	0.026	0.025
G_8	6,017	10.680	0.045	0.040	0.056
G_9	4,457	3.968	0.101	0.581	0.574
G_{10}	7,208	12.447	0.055	0.049	0.048
G_{11}	9,268	35.318	0.075	0.065	0.063
G_{12}	2,249	6.915	0.122	0.060	0.059
G_{13}	15,693	183.713	0.128	0.118	0.111
G_{14}	10,970	75.063	9.826	23.510	24.037
G_{15}	5,649	46.807	4.036	12.846	13.838
G_{16}	12,198	87.563	4.884	11.897	13.317
G_{17}	18,315	2,758.893	0.384	0.377	0.364
G_{18}	9,150	159.548	47.647	77.075	81.523
G_{19}	13,746	288.904	87.265	162.144	180.394
G_{20}	21,960	310.607	0.350	0.367	0.360
G_{21}	17,150	461.986	169.990	321.192	390.546
G_{22}	25,565	345.124	0.405	0.536	0.530
G_{23}	148,485	26,186.697	601.115	4,340.600	4,564.900
G_{24}	138,074	24,493.126	1,683.429	5,277.361	5,531.336
G_{25}	130,822	17,505.910	13,034.260	12,461.524	12,716.637
G_{26}	66,160	8,124.839	728.225	1,678.382	1,719.043
G_{27}	108,120	29,461.539	8,467.501	16,043.266	15,882.701
G_{28}	274,331	89,774.879	3.057	3.807	3.683

with kernelization are everywhere below this line. It means that all three kernelization methods bring benefit. This results from the fact that our implementations of Kernelization Methods 2 and 3 take linear running time and Edmonds' Blossom shrinking algorithm takes $O(n^4)$ time. That is, we do not solve the whole problem in $O(n^4)$ time but only a reduced part of it. Although we do not know whether our implementation of Kernelization Method 1 runs in linear time, this is obviously not significant slower than our implementations of Kernelization Methods 2 and 3. We next compare the computations with kernelization methods with each other.

Comparison between kernelization methods

Diagram 15 shows the differences between the kernelization methods. There are three curves on this diagram showing the ratios of the computation times of finding a maximum matching with Kernelization Methods 1, 2, and 3 divided by the computation time of finding a maximum matching with Kernelization Method 1. For some instances the curves of the computations with Kernelization Methods 2 and 3 are below the curve of the

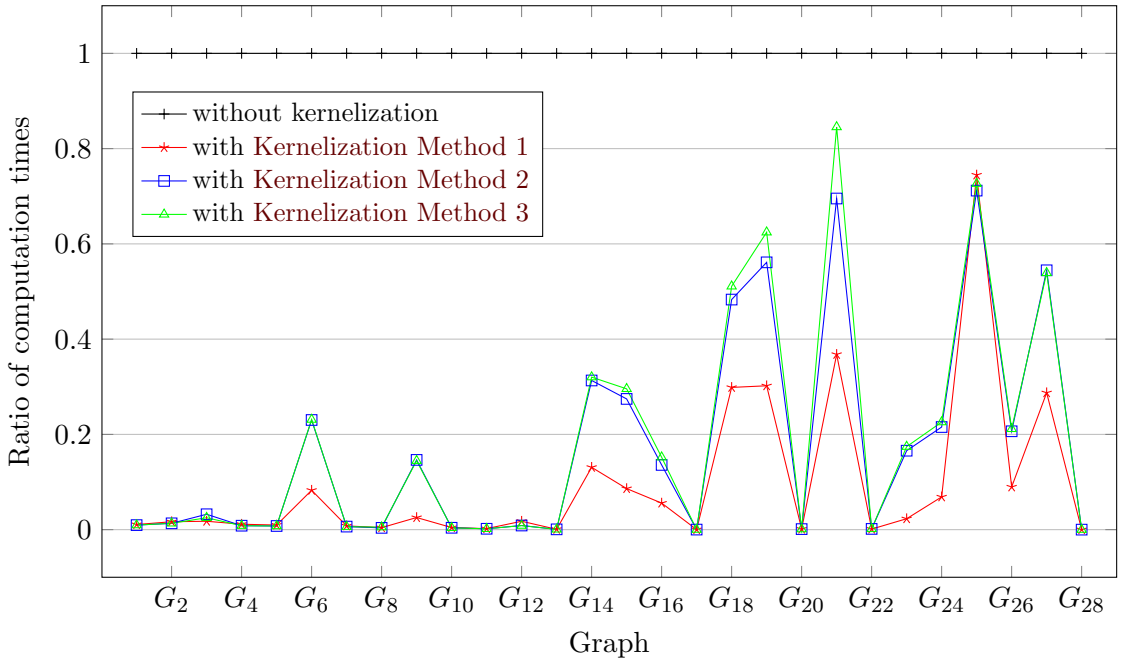


Diagram 14: Computation time of finding a maximum matching with kernelization methods in comparison to computation time of finding a maximum matching without kernelization. Every computation time of finding a maximum matching without kernelization is set to 1 and other computation times are scaled accordingly. For more details see Diagrams 15 and 16. The exact numbers are in Table 14 in Appendix A.

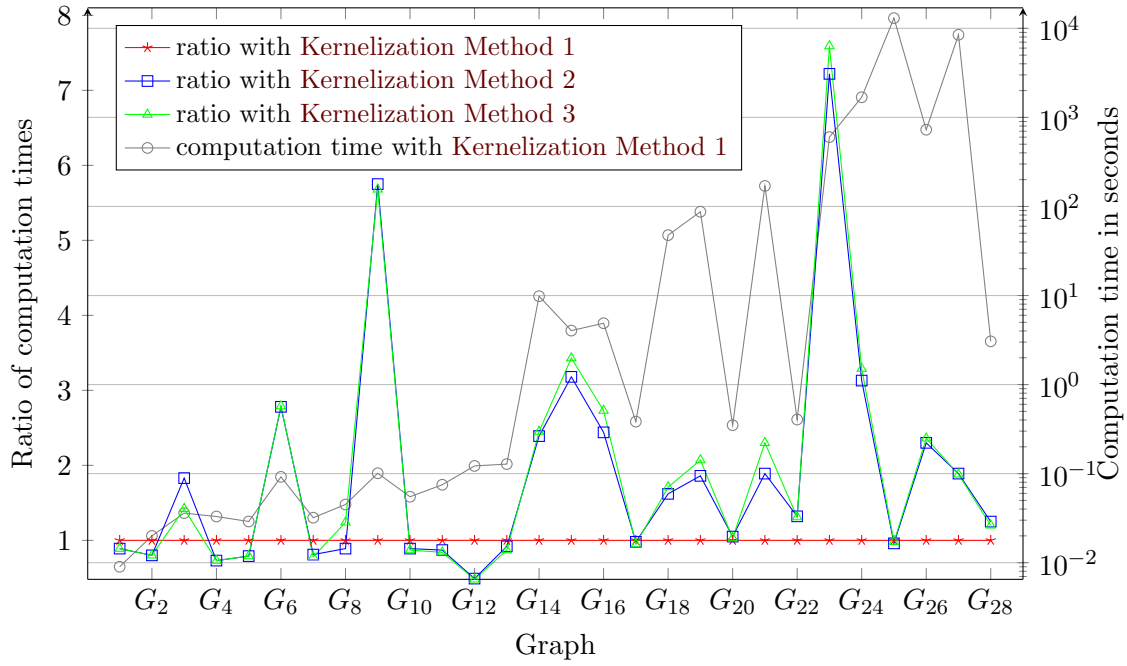


Diagram 15: Computation time of finding a maximum matching with Kernelization Method 1 and ratios of computation time of finding a maximum matching with kernelization methods to computation time of finding a maximum matching with Kernelization Method 1. The exact numbers are in Table 15 in Appendix A and in Table 5.

computation with Kernelization Method 1. The forth curve that represents the computation time of Kernelization Method 1 shows that all these instances except G_{25} could be solved in less then one second for every kernelization method. Therefore, we can neglect these instances. More importantly, for instances for which much more time is needed to solve them Kernelization Method 1 is almost always faster. For the only exception G_{25} Kernelization Method 2 is just 4% faster and Kernelization Method 3 is even only 2% faster than Kernelization Method 1.

For this reason we consider in the following only Kernelization Method 1, but before we compare Kernelization Methods 2 and 3 with each other. Although Kernelization Method 2 applies a standardized way for a merging of two vertices and Kernelization Method 3 makes it in an optimized way, the computation with Kernelization Method 2 is faster for 10 of 12 instances which could not be solved in less then one second for these kernelization methods. The first idea for the reason of it can be that there are many case distinctions in the implementation of the special merging approach for Kernelization Method 3. Sometimes the costs for the search a right case could outweigh the benefit of the optimal merging. However, we saw in Section 5.2 that the computation of a kernel lasts few seconds and one can see in Diagram 11 on page 38 that the difference between the computation times of Kernelization Methods 2 and 3 is insignificant. Thus the first idea is incorrect. The only possible reason is that Kernelization Methods 2 and 3 compute different kernels (of the same size) and then Edmonds' Blossom shrinking algorithm causes this difference in performance.

Graph size and speed-up of kernelization

The size of a graph does not influence the ratio between computations with kernelization and without. This can be seen on Diagram 16 for Kernelization Method 1. The number of vertices and edges in the graphs and a ratio between the computation time of finding a maximum matching with Kernelization Method 1 to the computation time of finding a maximum matching without kernelization are drawn here. This ratio is the pure speed-up of using Kernelization Method 1. The minimal speed-up on the tested graphs is 1.3 for G_{25} and the maximal speed-up is 29,367 for G_{28}. For 79% of the tested graphs the computation with Kernelization Method 1 is at least 10 times faster than the computation without any kernelization, for 36% even at least 100 times faster! The kernelization requires only a few seconds (see Diagram 11 on page 38), but leads to big saving of time for finding a maximum matching.

Diagram 17 visualises the effect of Kernelization Method 1. In particular, it is shown, how much the computation time of finding a maximum matching is reduced in comparison with the computation time of finding a maximum matching without kernelization. We can easily see that for non empty kernels the time reduction is everywhere higher than the graph reduction. Especially graphs G_{18}, G_{19}, and G_{21} stand out. We save a lot of time even if graph reduction is rather small.

The more we could reduce a graph the more time we could save. The only condition for this nice statement is that a data reduction has to be done faster than the computation of maximum matching with Edmonds' Blossom shrinking algorithm that takes $O(n^4)$ time. For instance, Data Reduction Rule 4 can be applied if there is a faster implementation for finding a maximum matching in a bipartite graph (see Section 3.2). Then it could speed up the computation of finding a maximum matching.

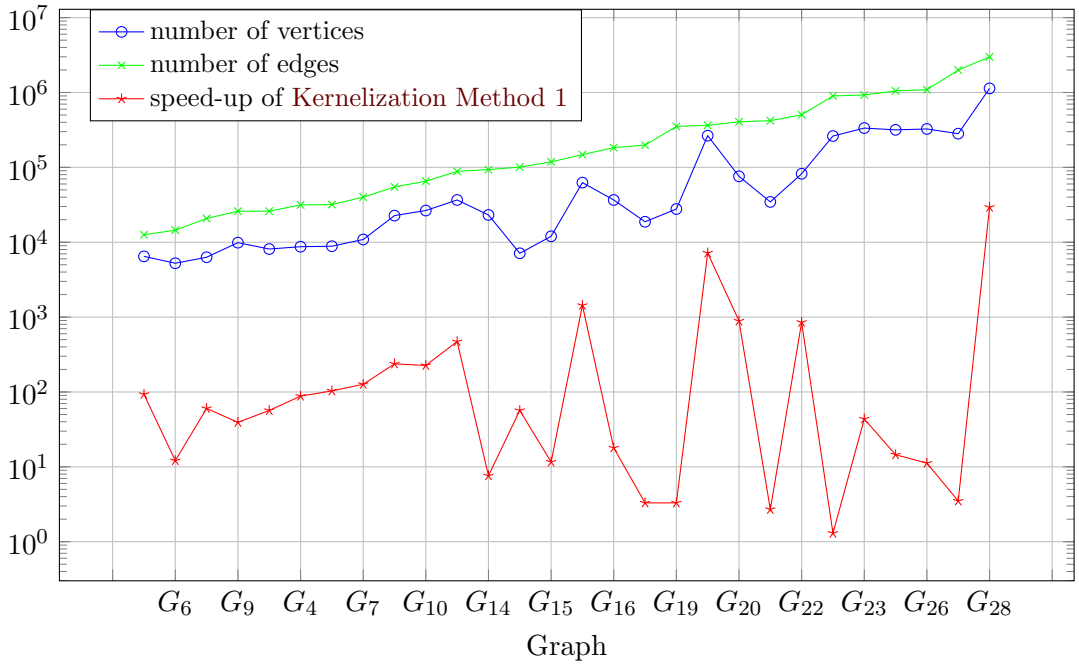


Diagram 16: Size of graphs and speed-up of computation of finding a maximum matching with Kernelization Method 1 compared to the computation of finding a maximum matching without kernelization. The graphs are sorted on the number of edges from the smallest left to the biggest right. The exact numbers are in Table 16 in Appendix A.

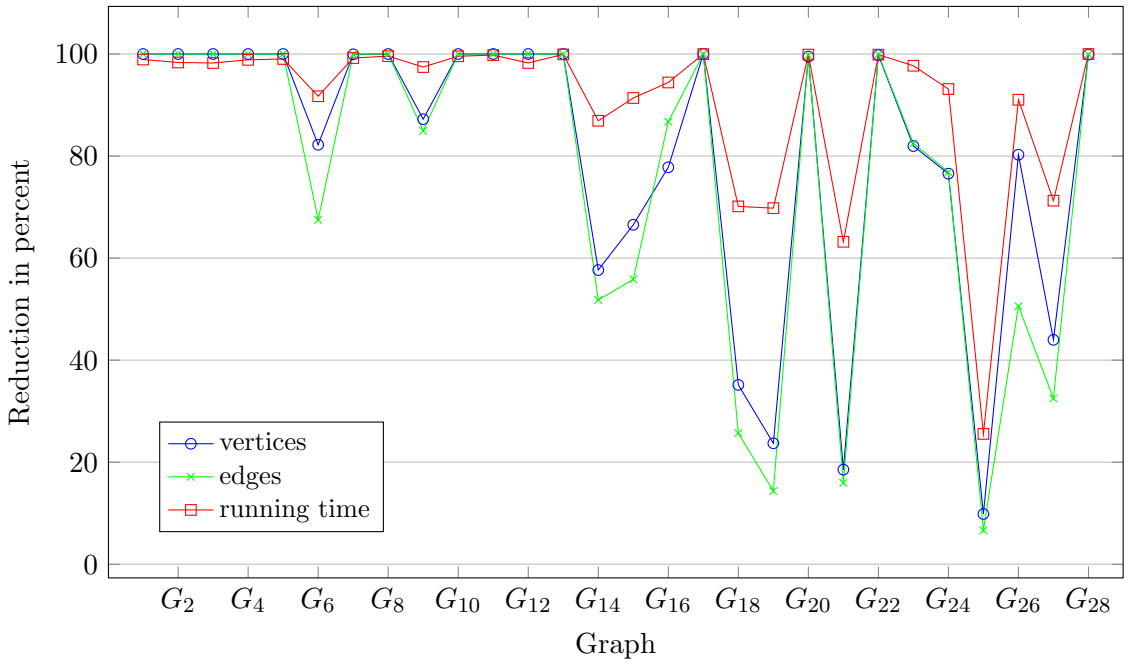


Diagram 17: Reduction of graphs with Kernelization Method 1 and reduction of computation times of finding a maximum matching with Kernelization Method 1 and Edmonds' Blossom shrinking algorithm in comparison to Edmonds' Blossom shrinking algorithm without kernelization. The exact numbers are in Table 17 in Appendix A.

Computation time with kernelization in detail

Now we consider the computation times with Kernelization Method 1 in detail. For the instances which were not finished within the time limit, we started the experiment again. We set the time limit to 32 hours per instance as before but we executed only Edmonds' Blossom shrinking algorithm with Kernelization Method 1. Only G_{35} and G_{37} lacked time. Table 6 contains the total computation time for finding a maximum matching with Kernelization Method 1 and its three components: the computation times of the kernelization, the finding a maximum matching in the kernel and the postprocessing.

Table 6: Computation time of finding a maximum matching with Kernelization Method 1 and Edmonds' Blossom shrinking algorithm in detail. Analogous tables for Kernelization Methods 2 and 3 are Tables 18 and 19 in Appendix A.

graph	kernel [s]	matching [s]	postprocessing [s]	total [s]
G_1	0.008	0	0.001	0.009
G_2	0.019	0	0.001	0.020
G_3	0.035	0	0.001	0.036
G_4	0.031	0	0.002	0.033
G_5	0.027	0	0.002	0.029
G_6	0.007	0.083	0.002	0.092
G_7	0.030	0	0.002	0.032
G_8	0.042	0.001	0.002	0.045
G_9	0.015	0.084	0.002	0.101
G_{10}	0.053	0	0.002	0.055
G_{11}	0.071	0	0.004	0.075
G_{12}	0.122	0	0	0.122
G_{13}	0.125	0	0.003	0.128
G_{14}	0.043	9.780	0.003	9.826
G_{15}	0.050	3.984	0.002	4.036
G_{16}	0.116	4.764	0.004	4.884
G_{17}	0.380	0	0.004	0.384
G_{18}	0.052	47.592	0.003	47.647
G_{19}	0.063	87.200	0.002	87.265
G_{20}	0.341	0.004	0.005	0.350
G_{21}	0.084	169.902	0.004	169.990
G_{22}	0.398	0.001	0.006	0.405
G_{23}	0.825	600.255	0.035	601.115
G_{24}	0.758	1,682.641	0.030	1,683.429
G_{25}	0.112	13,034.091	0.057	13,034.260
G_{26}	0.410	727.801	0.013	728.224
G_{27}	1.362	8,466.103	0.036	8,467.501
G_{28}	2.930	0.038	0.089	3.057
G_{29}	1.650	9,982.338	0.231	9,984.219
G_{30}	0.502	54,760.604	0.185	54,761.294
G_{31}	2.227	11,363.179	0.258	11,365.665
G_{32}	7.563	0	0.021	7.584
G_{33}	2.540	43,363.358	0.151	43,366.049
G_{34}	2.790	33,996.467	0.317	33,999.574
G_{36}	13.265	1,595.557	0.144	1,608.966

The computation time to find a kernel was considered thoroughly in Section 5.2. The computation time for the postprocessing needs only a few milliseconds. Graph G_{34} needs the most time for the postprocessing but even this case only takes 0.317 seconds.

It is interesting to compare G_{25} and G_{28}. They need for the postprocessing 0.057 and 0.089 seconds respectively. We know from Section 4.2 that there are edges that we store alone during the kernelization. Those edges are easily added to the matching in the postprocessing. There are also pairs of edges, where exactly one of the two has to be added to the matching. In this case it is necessary to look which of the edges has already a matched endpoint. We can see in Table 2 on page 39 that G_{28} reserves 273 thousand edges of a maximum matching. In contrast, the number of reserved edges of G_{25} is only 12 thousand. But on the other hand, a maximum matching of the kernel of G_{25} has 118 thousand edges and a maximum matching of the kernel of G_{28} only 0.5 thousand edges. The number of matched vertices in the graph before the postprocessing is twice more. Therefore, the postprocessing needs less efforts to add the reserved edges to G_{28}, but has to add much more edges.

The total computation time of finding a maximal matching with Kernelization Method 1 is less than one second for 46% of the instances. We can see in Table 5 on page 45 that only G_1 could be solved in less than one second without kernelization. For 46% of the instances the computation time of finding a maximum matching in the kernel is less than the computation time of the kernelization, e. g. for instances G_{20}, G_{22}, or G_{28}. In Diagram 10 on page 37 we see that for all such instances the kernel contains less than 1% of vertices and edges of the original graph. Then it is clear that the computation of a maximum matching in the kernel of such graphs lasts only few milliseconds. For all other graphs, for which the kernel has a bit more vertices and edges, most time is spent on finding a maximum matching in the kernel.

Diagram 18 visualizes the computation time for finding a maximum matching with Kernelization Method 1, where the computation times for kernelization and postprocessing are combined, and additionally the computation time for finding a maximum matching without kernelization. We see again that the computation time for the kernelization takes few seconds or even few split seconds, but it saves a lot of time for finding a maximum matching.

5.4. Blossom V

Our next experiments use a C++ implementation of Blossom V [Kol08]. This is a state-of-the-art implementation of Edmonds' algorithm for computing a perfect matching of minimum cost. Details are described in the article "Blossom V: A new implementation of a minimum cost perfect matching algorithm" by Kolmogorov [Kol09]. We called this C++ implementation by means of Java Native Interface in such way that we got a maximal matching. To this end we employed a standard reduction from maximum matching to perfect matching [LP86].

Comparison between Blossom V and Edmonds' Blossom Shrinking

First of all, we compared the computation times of finding a maximum matching with Blossom V and Edmonds' Blossom Shrinking to find out whether applying Blossom V has an advantage. We set the time limit to 6 hours per instance. Diagram 19 shows both computation times without kernelization and we see that Blossom V is clearly better than Edmonds' Blossom shrinking algorithm. The minimum speed-up amounts to 20.2 for G_7.

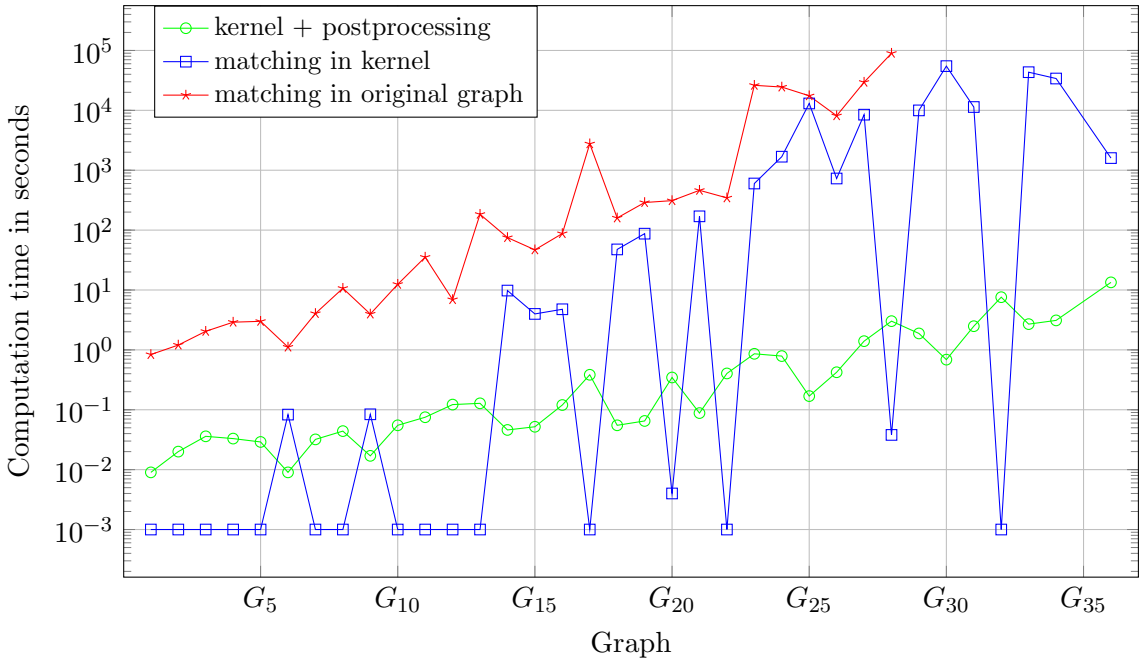


Diagram 18: Computation time of finding a maximum matching algorithm with Kernelization Method 1 and Edmonds' Blossom shrinking algorithm. Since points with value 0 cannot be drawn on logarithmic axis, we drew in such cases the value 0.001. We plot the computation time of Edmonds' Blossom shrinking algorithm without kernelization only for $G_1, \ldots, G_{28}$ because the remaining graphs could not be solved within the time limit. The exact numbers are in Table 20 in Appendix A.

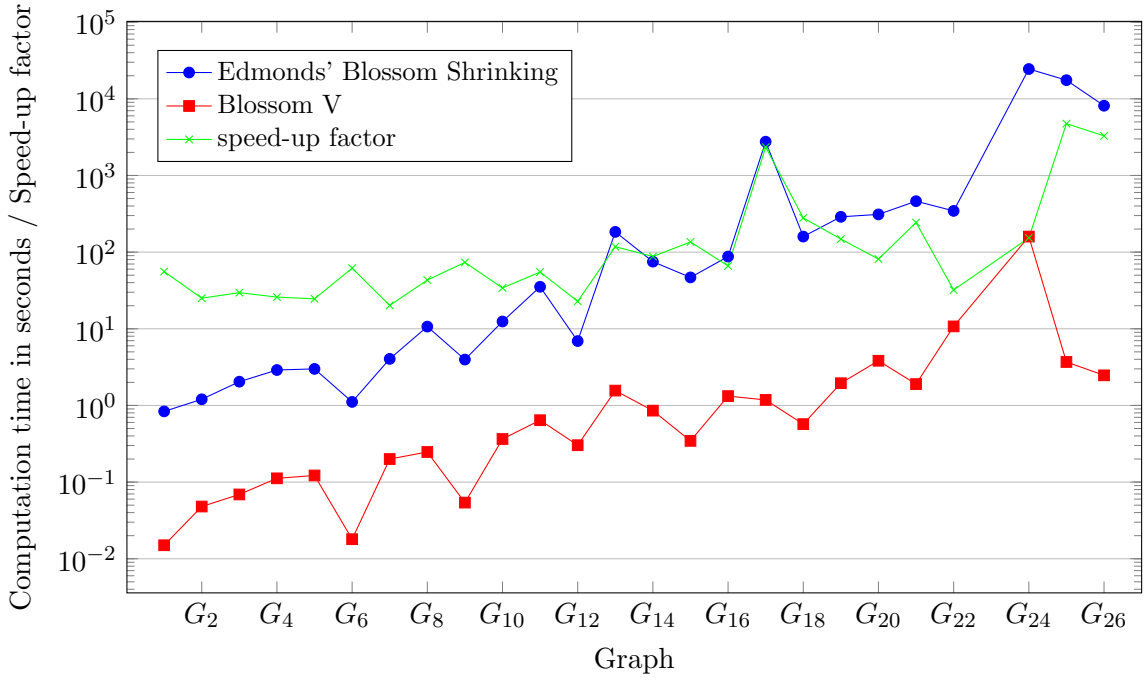


Diagram 19: Computation times of finding a maximum matching without kernelization. The exact numbers are in Table 21 in Appendix A.

The speed-up for G_{25} and G_{26} is over 3.000. The computation of bigger instances and G_{23} was stopped. The advantage of applying Blossom V is obvious and we did not increase the time limit to compute the bigger instances.

Comparison between computations with and without kernelization

We return now to our aim and compare the computation time of finding a maximum matching with Blossom V without kernelization and computation times of finding a maximum matching with our three methods of kernelization using Blossom V. We set the time limit again to 32 hours per instance which include all four ways for the computation a maximum matching: without kernelization and with the three kernelization methods. Our results are in Table 7. The computation of missing instances was broken with the error message "Segmentation fault (core dumped)" in the C++ program. After many tests we could not find the reason for this error.

Table 7: Computation time of finding a maximum matching with Blossom V

graph	matching size	without kernelization [s]	with Kernelization Method 1 [s]	with Kernelization Method 2 [s]	with Kernelization Method 3 [s]
G_1	1,048	0.015	0.008	0.008	0.009
G_2	2,054	0.057	0.030	0.014	0.014
G_3	2,574	0.066	0.027	0.028	0.025
G_4	3,405	0.152	0.025	0.022	0.024
G_5	3,428	0.127	0.024	0.020	0.020
G_6	2,329	0.015	0.016	0.015	0.016
G_7	4,348	0.203	0.031	0.025	0.025
G_8	6,017	0.256	0.046	0.041	0.039
G_9	4,457	0.054	0.031	0.035	0.035
G_{10}	7,208	0.373	0.053	0.066	0.048
G_{11}	9,268	0.627	0.073	0.067	0.066
G_{12}	2,249	0.307	0.088	0.067	0.058
G_{13}	15,693	1.636	0.132	0.126	0.122
G_{14}	10,970	0.859	0.306	0.236	0.237
G_{15}	5,649	0.344	0.147	0.346	0.347
G_{16}	12,198	1.333	0.195	0.217	0.214
G_{17}	18,315	1.180	0.358	0.405	0.394
G_{18}	9,150	0.476	0.282	0.344	0.344
G_{19}	13,746	1.954	1.652	1.639	1.633
G_{20}	21,960	3.867	0.347	0.329	0.319
G_{21}	17,150	1.911	1.751	1.769	1.762
G_{22}	25,565	10.907	0.410	0.397	0.386
G_{23}	148,485	64.054	1.335	3.278	3.234
G_{24}	138,074	159.783	2.090	4.205	4.179
G_{25}	130,822	3.811	3.359	3.006	2.991
G_{26}	66,160	2.421	0.782	0.848	0.836
G_{27}	108,120	21.559	7.561	8.822	8.785
G_{30}	201,272	3,627.744	391.166	40.046	39.907
G_{32}	56,063	218.854	8.848	7.886	7.837
G_{35}	245,311	84.004	22.343	32.372	31.968

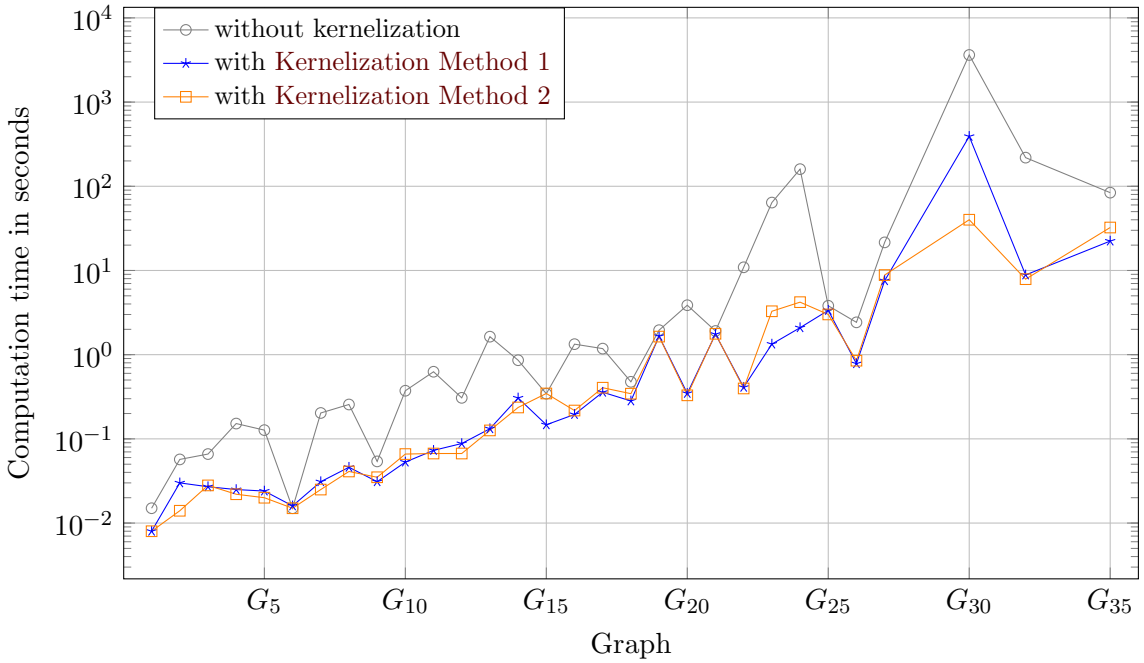


Diagram 20: Computation times of finding a maximum matching with Blossom V with and without kernelization

In contrast to Edmonds' Blossom Shrinking the computation times for finding a maximum matching with Kernelization Methods 2 and 3 are almost identical. Because of this we subsequently do not consider Kernelization Method 3 separately.

Diagram 20 visualizes the computation times from Table 7. We see that any method of kernelization speeds up the finding of a maximum matching. There are two exceptions: G_6 and G_{15}, but the time difference for both instances is at most 3 milliseconds. Hence, we can say that the computation with kernelization pays off. The speed-up of executing kernelization is drawn in Diagram 21.

Our kernelization methods only apply data reduction rules for vertices of degree zero, one, and two. A natural question is whether there are data reduction rules for vertices of degree three or more which reduce graph still more.

Comparison between kernelization methods

Diagram 21 also shows that for some instances, e. g. G_{23}, Kernelization Method 1 is faster and for some instances, e. g. G_{30}, Kernelization Method 2 is faster. We now compare both methods in detail. Diagram 22 visualizes the computation time for finding a maximum matching with Kernelization Methods 1 and 2 and Blossom V, where the computation times for kernelization and postprocessing are combined. The computation of kernelization and postprocessing for Kernelization Method 1 is for 90% of the instances longer than for Kernelization Method 2. However, we can see that the difference is not significant. A difference occurs in the computation of the maximum matchings in the kernels. We know from Section 5.2 that the kernel after Kernelization Method 1 is smaller than after Kernelization Method 2 (see for example Diagram 10). One might conjecture that the computation of a maximum matching in the kernel after Kernelization Method 1 will be faster. For most instances this conjecture is correct, however G_{30} disproves it distinctly. Thus we cannot definitely say that some kernelization method is better.

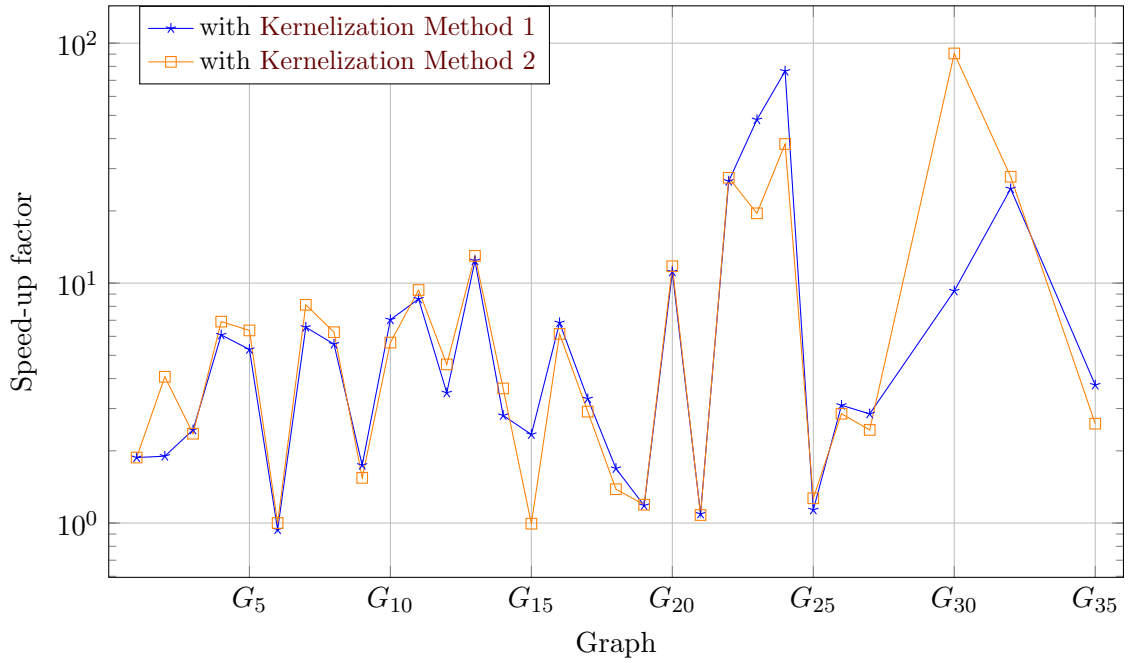


Diagram 21: Speed-up of computation time of finding a maximum matching with kernelization compared to computation time of finding a maximum matching without kernelization. The exact numbers are in Table 22 in Appendix A.

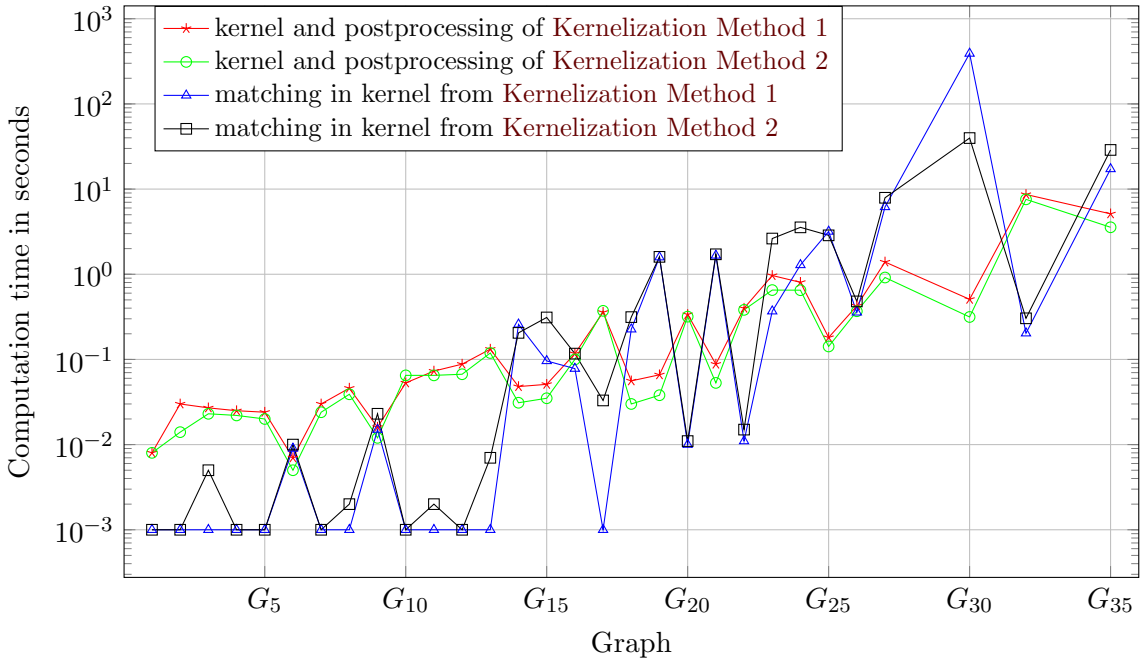


Diagram 22: Computation times of finding a maximum matching with kernelization and Blossom V. Since points with value 0 cannot be drawn on logarithmic axis, we drew in such cases the value 0.001. The exact numbers are in Table 23 in Appendix A.

The example of instance G_{30} leads to a question whether the kernel after Kernelization Method 1 is a subgraph of the kernel after Kernelization Method 2. Both kernelization methods apply Data Reduction Rule 1, but furthermore Kernelization Method 1 applies Data Reduction Rule 2 and Kernelization Method 2 applies Data Reduction Rule 3 that is a special case of Data Reduction Rule 2. If the kernel after Kernelization Method 1 is a subgraph of the kernel after Kernelization Method 2, then we would have an example for which Blossom V computes a maximum matching in a subgraph slower than in the graph itself. In this case we could not state that data reduction reduces the computation of finding a maximum matching with Blossom V.

Comparison of computation with kernelization between Blossom V and Edmonds' Blossom Shrinking

We compared the computation times of finding a maximum matching with Blossom V and Edmonds' Blossom Shrinking without kernelization and saw in Diagram 19 on page 51 that Blossom V is clearly better. Now we compare both algorithms with executing Kernelization Method 1. Diagram 23 visualizes this comparison. We chose Kernelization Method 1 because it was the best method for Edmonds' Blossom Shrinking.

At first sight it seems that there are many instances where the difference between Blossom V and Edmonds' Blossom Shrinking is not significant. However, we can see in Diagram 10 on page 37 that all those instances for which the difference is not significant have less than 1% of vertices and edges in the kernel. For all other instances Blossom V is clearly better than Edmond Blossom Shrinking as before.

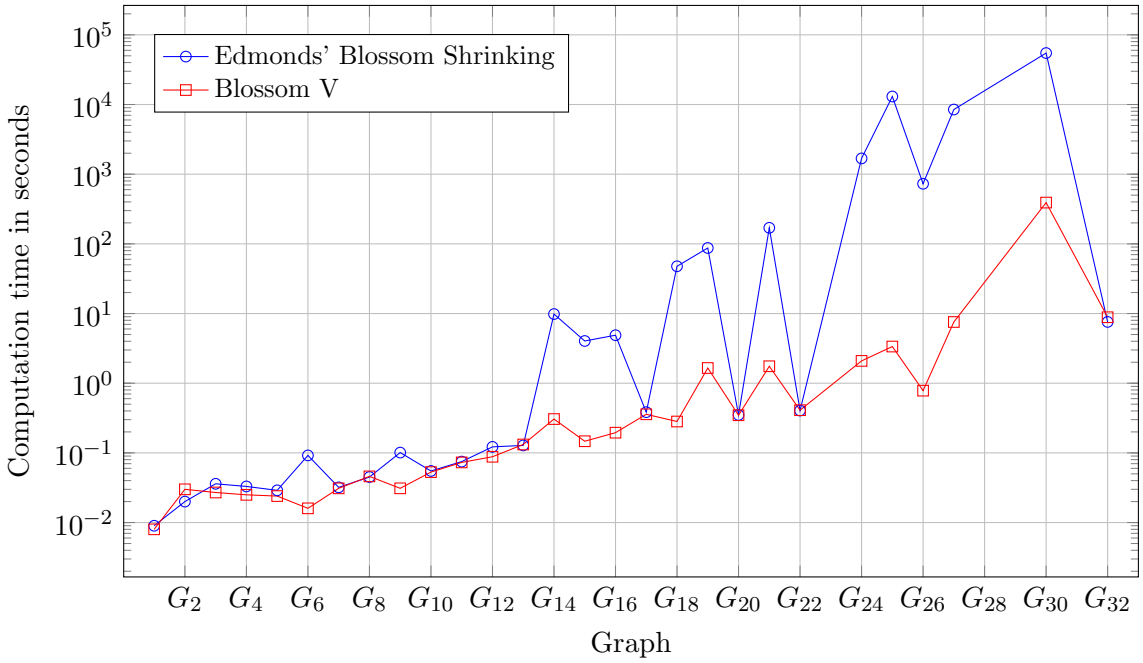


Diagram 23: Computation times of finding a maximum matching with Kernelization Method 1. The exact numbers are in Table 27 in Appendix A.

5.5. Heuristics

There are two factor 1/2 approximation algorithms for finding a maximum-weight matching in an arbitrary graph in the JGraphT library: Path Growing Weighted Matching [DH03] and Greedy Weighted Matching [DH03; Pre99]. They take respectively $O(m)$ and $O(n + m \log n)$ time. We executed both heuristics with and without kernelization for our test graphs. The received weights are the sizes of the found matchings. In our case factor 1/2 means that the found matching may be not optimal but its size is at least half of the size of a maximum matching. We analyze two aspects of the application of the kernelization: reduction of running time and improvement of found matchings.

Matchings

We begin with the sizes of the found matchings. Diagram 24 shows the sizes in percent of maximum matchings found with Path Growing Weighted Matching and with Greedy Weighted Matching without kernelization and with Kernelization Methods 1 and 2. Recall that Kernelization Methods 2 and 3 applied the same data reduction rules. Although the number of reduced vertices and edges is the same, the structure of the resulting kernels can be different. Hence, the heuristics can find matchings of different sizes on these kernels. Despite this, the matching sizes found with Kernelization Methods 2 and 3 are equal for all 37 instances by applying both Path Growing Weighted Matching and Greedy Weighted Matching. For this reason there is no extra curve for Kernelization Method 3 in Diagram 24.

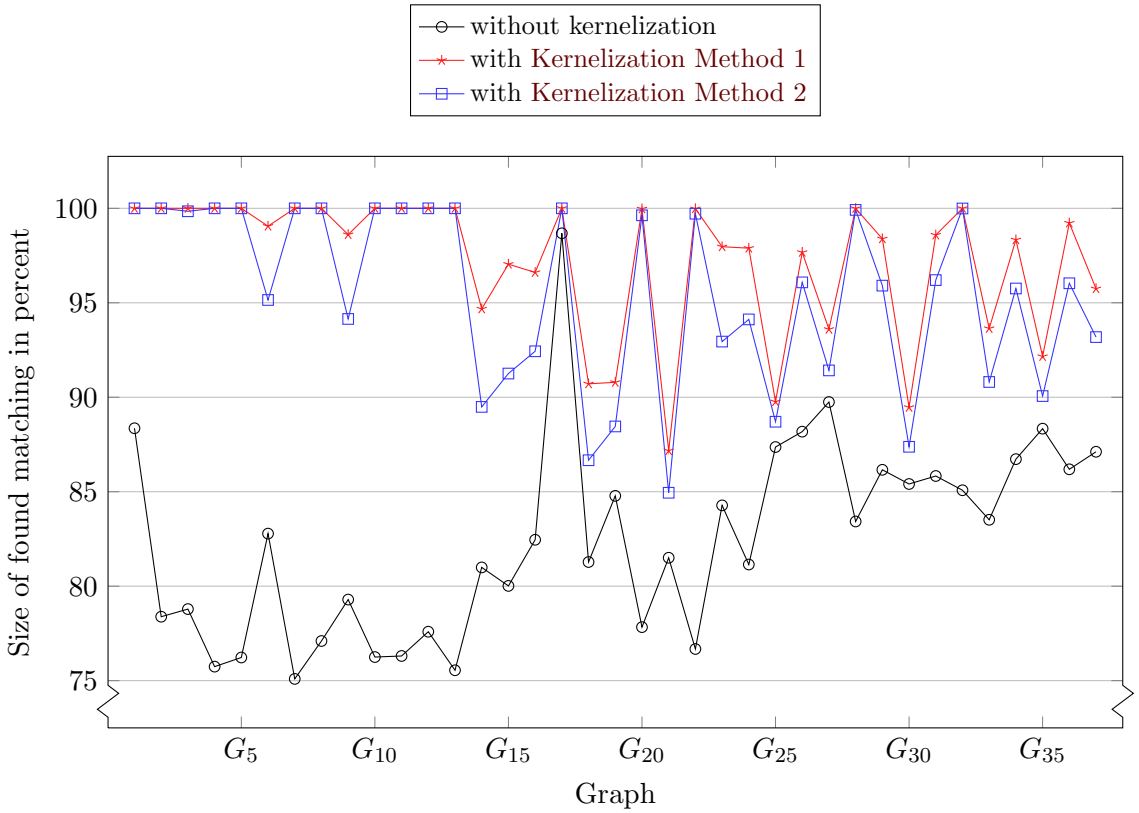
We see that both heuristics found suboptimal matchings. For most of the instances the found matching has size between 75% and 90% of the size of a maximum matching for both Path Growing Weighted Matching and Greedy Weighted Matching. Only for G_{17} the found matching has size 98,7% and 98,8% respectively. We see distinctly that executing kernelization improves the size of the found matching for all instances for all kernelization methods for both heuristics, where executing Kernelization Method 1 brings a bit better results. However, the application of our kernelization methods cannot improve the approximation ratio 1/2 because there are graphs for which our three kernelization methods have no effect.

A very interesting result is that the matchings found with the kernelization, have almost the same size for both heuristics, independently what for kernelization method we use. The biggest difference between two heuristics amounts to 1.3% for instance G_{18} for Kernelization Method 2.

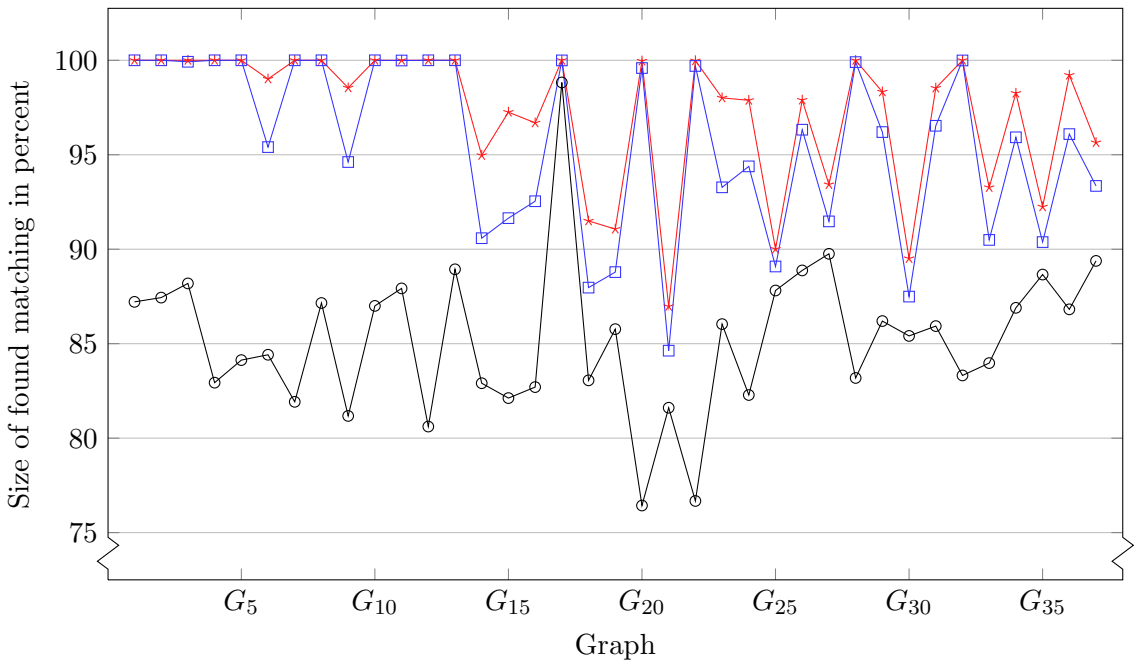
Thus when we search a matching with a heuristic, the execution of the kernelization seems to bring mostly an improvement of the matching size. We consider now what influence on the running time has the kernelization.

Computation time

We can see in Diagram 25 that the computation time for finding a matching with Greedy Weighted Matching without kernelization is distinctly faster than with Path Growing Weighted Matching. The application of kernelization reduced the computation time for finding a matching with Path Growing Weighted Matching and increased the computation time for finding a matching with Greedy Weighted Matching. There are almost no difference between finding a matching with Kernelization Methods 2 and 3. For Path Growing Weighted Matching finding a matching with Kernelization Method 1 is up to seven times faster on the tested instances than with Kernelization Methods 2 or 3 or the difference

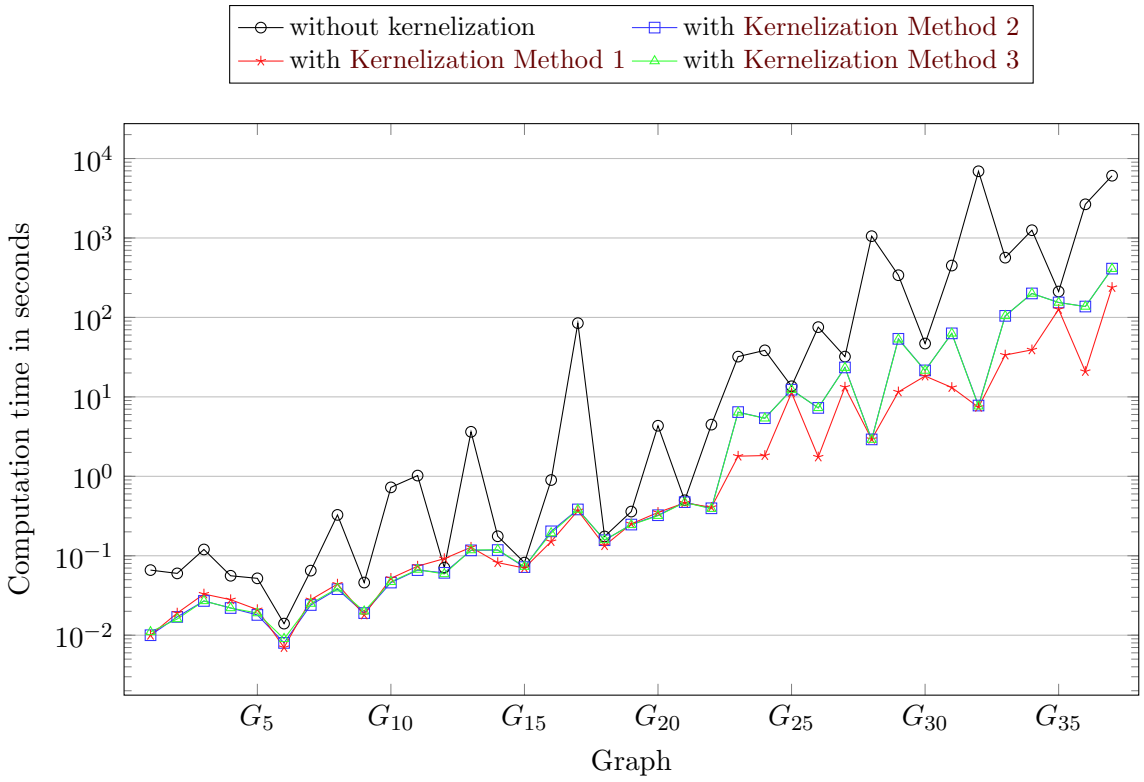


(a) Path Growing Weighted Matching

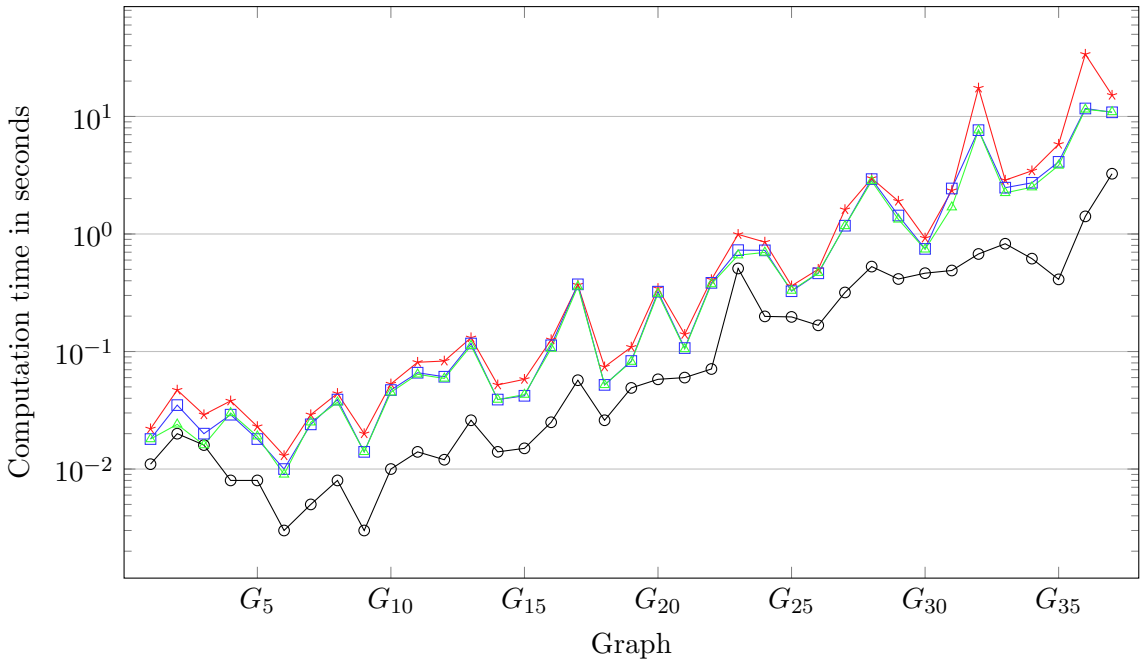


(b) Greedy Weighted Matching

Diagram 24: Size of found matching in percent, where 100 percent is the size of a maximum matching. The exact numbers are in Table 28 in Appendix A.



(a) Path Growing Weighted Matching



(b) Greedy Weighted Matching

Diagram 25: Computation time of finding a matching with heuristics with and without kernelization. The exact numbers are in Tables 29 and 30 in Appendix A.

is not significant. For Greedy Weighted Matching finding a matching with Kernelization Method 1 is in contrast up to three times slower.

Kernelization in detail

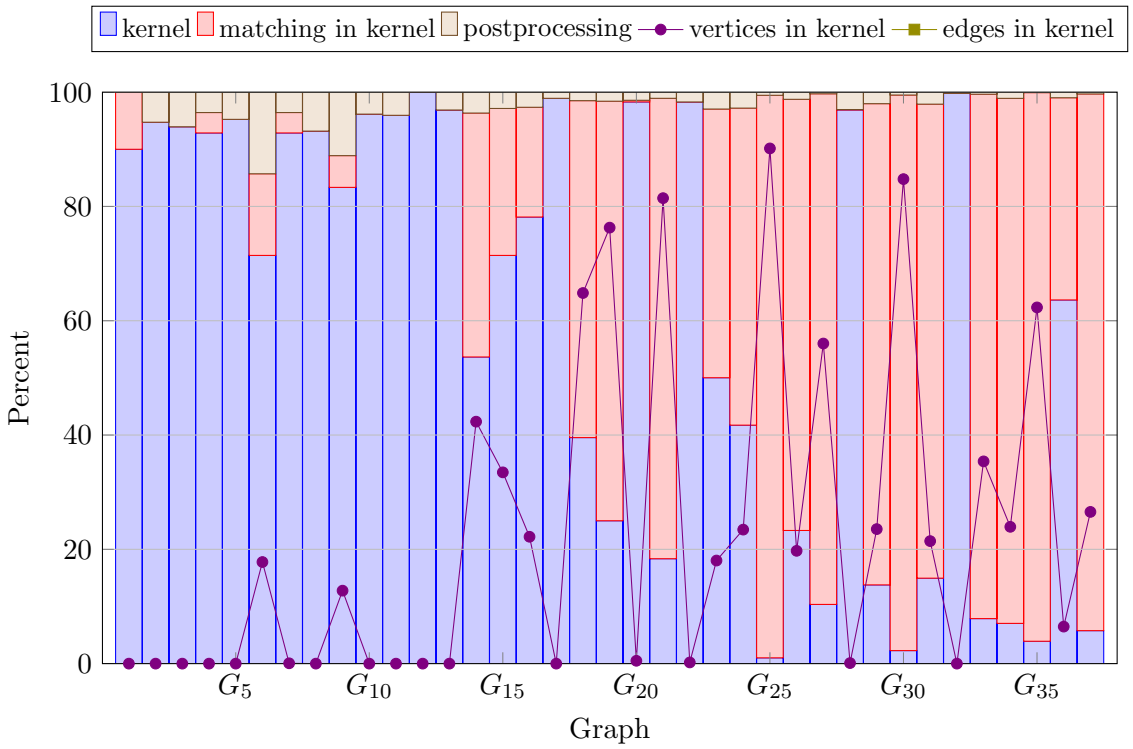
We consider now the computation time of finding a matching with Kernelization Method 1 in detail. The data reduction to a kernel is always the same, but finding a matching in a kernel takes different time for different heuristics. As the postprocessing depends on the found matching, it takes different time as well. In Diagram 26 we can easily see that finding a matching in the kernel with Greedy Weighted Matching is faster than with Path Growing Weighted Matching. The last one needs the most computation time of finding a matching in quite big instances for finding a matching in the kernel.

In contrast, for all instances except G_{25} and G_{30} the data reduction to the kernel takes over 50 percent of the computation time of finding a matching with Greedy Weighted Matching. The kernel of G_{25} has 90% vertices and 93% edges of the original graph and the kernel of G_{30} has 85% vertices and 90% edges of the original graph. Thus the computation time for finding a matching in these kernels takes almost as much time as finding a matching in the original graphs. Indeed, Greedy Weighted Matching needs 197 milliseconds for G_{25} and 184 milliseconds for its kernel and 464 milliseconds for G_{30} and 422 milliseconds for its kernel. Executing Kernelization Method 1 doubles the computation time for these two instances. For the other instances the expansion of computation time by applying Kernelization Method 1 is much more.

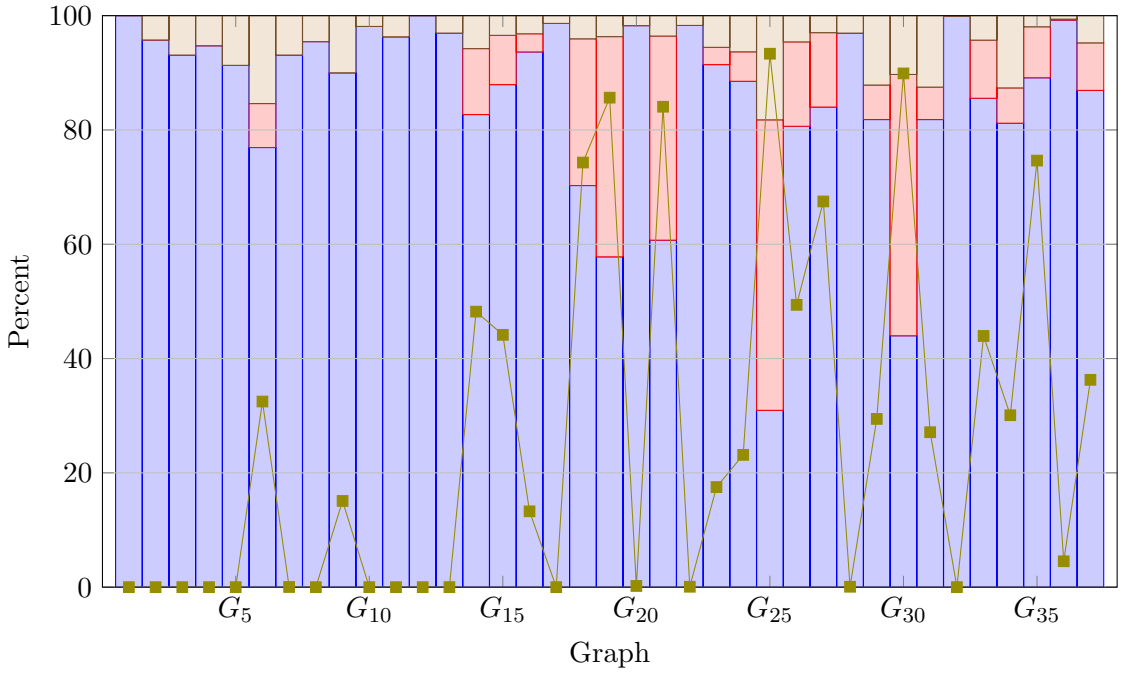
Pros and cons executing kernelization for heuristics

We saw that Path Growing Weighted Matching is the slower of the two considered heuristics. For this heuristic the execution of the kernelization reduces the computation time of finding a matching and improves the matching size. There is no doubt that applying the kernelization brings only advantages. Kernelization Method 1 is the best of our three kernelization methods for both criteria.

In contrast, it is not so clear for the faster heuristic Greedy Weighted Matching whether applying the kernelization is good or bad. The kernelization increases the matching size, but this heuristic is so fast that the execution of the kernelization increases the computation time. If one needs only a rough matching, for example as an initial instance, then it is recommended to use Greedy Weighted Matching without kernelization. If the computation time is not such a critical point, then the big advantage of the kernelization is the larger size of the found matching. Executing Kernelization Method 1 brings the biggest improvement of the matching size, but its execution needs even more time. Kernelization Methods 2 and 3 are a bit faster. Besides that, for some graph the computation with Path Growing Weighted Matching with Kernelization Method 1 is faster than with Greedy Weighted Matching with Kernelization Method 1. We can see this in Diagram 27. Nevertheless, there are many instances for which the computation with Path Growing Weighted Matching with Kernelization Method 1 is much slower.



(a) Path Growing Weighted Matching



(b) Greedy Weighted Matching

Diagram 26: Computation time of finding a matching with heuristics with Kernelization Method 1 in detail and remaining vertices and edges after Kernelization Method 1 in percent. The exact numbers are in Tables 10, 31 and 32 in Appendix A.

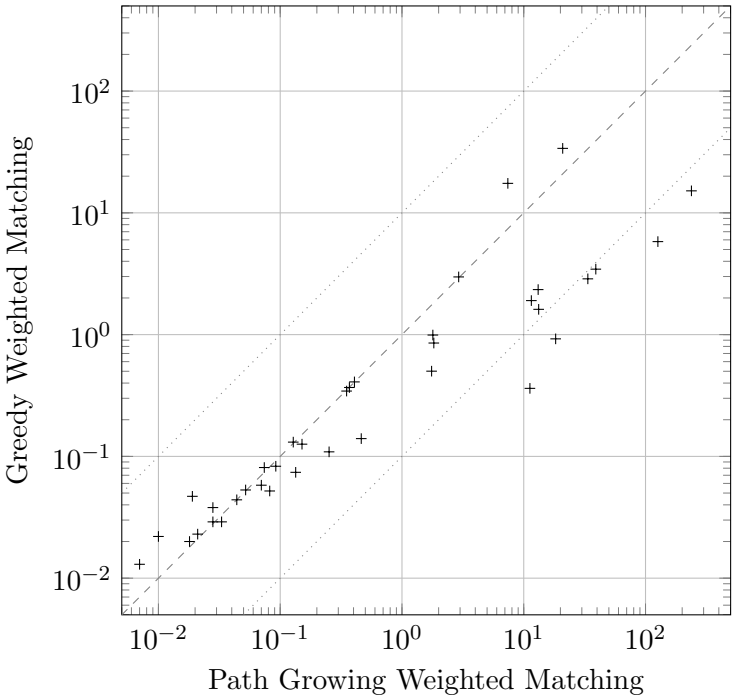


Diagram 27: Comparison of the computation times of the heuristics with Kernelization Method 1. The dashed line denotes the values where both heuristics would have the same computation time. Marks above the dashed line indicate an instance where Path Growing Weighted Matching with Kernelization Method 1 is faster, marks below the dashed line indicate an instance where Greedy Weighted Matching with Kernelization Method 1 is faster. The dotted lines mark the factor 10 increase/decrease. The exact numbers are in Table 33 in Appendix A.

6. Outlook

In this thesis we investigated the application of data reduction for Matching in practice. The implementation of the data reduction rules led to simple program code. Correspondingly, they could be executed in a fast way. As a result we could speed up all exact algorithms that we studied. Thus the data reduction rules should be incorporated in all exact algorithms. The decision to apply the data reduction rules for heuristics or not should be made for each heuristic separately. The data reduction rules sped up the computation for one heuristic and improved the size of the found matching, but slowed down the computation for another heuristic while improving the size of the found matching as well. Thus it is important what aim is pursued.

Moreover, we developed data reduction rules for edge-weighted graphs. We showed that Maximum-Weight Matching admits a quasi-linear-time computable linear-size kernel with respect to the parameter *feedback edge number*.

Our work leaves several challenges for future work. We gave an algorithm that finds a crown in $O(m\sqrt{n})$ time. Can this algorithm be executed faster? Is there a restricted version of crown reductions that can be executed faster? Are there some other data reduction rules for Matching, e. g. for vertices of degree three or more? It would also be interesting to experimentally validate how our data reduction rules for the edge-weighted case work in practice.

References

[Abu+07] F. N. Abu-Khzam, M. R. Fellows, M. A. Langston, and W. H. Suters. "Crown Structures for Vertex Cover Kernelization". In: *Theory of Computing Systems* 41.3 (2007), pp. 411–430 (cit. on p. 18).

[BK09] M. Bartha and M. Kresz. "A Depth-first Algorithm to Reduce Graphs in Linear Time". In: *Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*. IEEE Computer Society, 2009, pp. 273–281 (cit. on pp. 17, 18).

[Blu90] N. Blum. "A New Approach to Maximum Matching in General Graphs". In: *International Colloquium on Automata, Languages, and Programming (ICALP)*. Vol. 443. Lecture Notes in Computer Science. Springer, 1990, pp. 586–597 (cit. on pp. 10, 20).

[BM76] J. A. Bondy and U. S. R Murty. *Graph Theory With Applications*. Macmillan London, 1976 (cit. on p. 19).

[Cyg+15] M. Cygan, F. V. Fomin, L. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized Algorithms*. Springer, 2015 (cit. on p. 15).

[DF13] R. G. Downey and M. R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013 (cit. on p. 15).

[DH03] D. E. Drake and S. Hougardy. "A simple approximation algorithm for the weighted matching problem". In: *Information Processing Letters* 85.4 (2003), pp. 211–213 (cit. on pp. 10, 56).

[Edm65] J. Edmonds. "Paths, Trees, and Flowers". In: *Canadian Journal of Mathematics* (1965), pp. 449–467 (cit. on pp. 10, 41).

[FG06] J. Flum and M. Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2006 (cit. on p. 15).

[Gab90] H. N. Gabow. "Data Structures for Weighted Matching and Nearest Common Ancestors with Linking". In: *Symposium on Discrete Algorithms (SODA)*. SIAM, 1990, pp. 434–443 (cit. on p. 10).

[GT91] H. N. Gabow and R. E. Tarjan. "Faster Scaling Algorithms for General Graph-Matching Problems". In: *Journal of the ACM* 38.4 (1991), pp. 815–853 (cit. on pp. 10, 20).

[HK73] J. E. Hopcroft and R. M. Karp. "A $n^{5/2}$ Algorithm for Maximum Matchings in Bipartite Graphs." In: *SIAM Journal on Computing* 2.4 (1973), pp. 225–231 (cit. on pp. 19, 20).

[Kol08] V. Kolmogorov. *BLOSSOM V - implementation of Edmonds' algorithm for computing a minimum cost perfect matching in a graph*. 2008-2009. URL: http://pub.ist.ac.at/~vnk/software/blossom5-v2.05.src.tar.gz (cit. on pp. 10, 33, 50).

[Kol09] V. Kolmogorov. "Blossom V: a new implementation of a minimum cost perfect matching algorithm". In: *Mathematical Programming Computation* 1.1 (2009), pp. 43–67 (cit. on pp. 10, 50).

[Les] J. Leskovec. *Stanford Network Analysis Package (SNAP)*. URL: https://snap.stanford.edu/data/ (cit. on pp. 10, 33).

[LP86] L. Lovász and M. D. Plummer. *Matching Theory*. Vol. 29. North-Holland Mathematics Studies. Elsevier Science, 1986 (cit. on p. 50).

[MNN17] G. B. Mertzios, A. Nichterlein, and R. Niedermeier. "The Power of Linear-Time Data Reduction for Maximum Matching". In: *Mathematical Foundations of Computer Science (MFCS)*. Vol. 83. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017, 46:1–46:14 (cit. on pp. 10, 11, 17, 18, 29).

[MS04] M. Mucha and P. Sankowski. "Maximum Matchings via Gaussian Elimination". In: *Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE Computer Society, 2004, pp. 248–255 (cit. on p. 10).

[MS07] J. Maue and P. Sanders. "Engineering Algorithms for Approximate Weighted Matching". In: *Workshop on Experimental Algorithms (WEA)*. Vol. 4525. Lecture Notes in Computer Science. Springer, 2007, pp. 242–255 (cit. on p. 10).

[MV80] S. Micali and V. V. Vazirani. "An $O(\sqrt{|V|}|E|)$ Algorithm for Finding Maximum Matching in General Graphs". In: *Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE, 1980, pp. 17–27 (cit. on pp. 10, 20).

[Nav79] B. Naveh. *JGraphT*. 1979. URL: http://jgrapht.org/ (cit. on p. 33).

[Nie06] R. Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford University Press, 2006 (cit. on p. 15).

[Pre99] R. Preis. "Linear Time 1/2-Approximation Algorithm for Maximum Weighted Matching in General Graphs". In: *Symposium on Theoretical Aspects of Computer Science (STACS)*. Vol. 1563. Lecture Notes in Computer Science. Springer, 1999, pp. 259–269 (cit. on pp. 10, 56).

[Wil12] V. V. Williams. "Multiplying matrices faster than Coppersmith-Winograd". In: *Symposium on the Theory of Computing (STOC)*. ACM, 2012, pp. 887–898 (cit. on p. 10).

A. Tables

Table 8: Time for file conversion and graph creation

graph	vertices	edges	file size [KB]	conversion [s]	graph creation [s]
G_1	6,474	12,572	112	0.004	0.142
G_2	6,301	20,777	210	0.036	0.314
G_3	8,114	26,013	266	0.037	0.312
G_4	8,717	31,525	325	0.028	0.302
G_5	8,846	31,839	329	0.040	0.434
G_6	5,241	14,484	343	0.103	0.329
G_7	10,876	39,994	421	0.033	0.396
G_8	22,687	54,705	631	0.072	0.642
G_9	9,875	25,973	643	0.085	0.561
G_{10}	26,518	65,369	761	0.055	0.702
G_{11}	36,682	88,328	1,052	0.137	1.012
G_{12}	7,115	100,762	1,069	0.512	1.053
G_{13}	62,586	147,892	1,809	0.130	1.643
G_{14}	23,133	93,439	2,363	0.165	2.092
G_{15}	12,006	118,489	2,949	0.235	2.500
G_{16}	36,692	183,831	3,954	0.541	3.719
G_{17}	265,009	364,481	4,884	0.377	4.484
G_{18}	18,771	198,050	5,160	0.358	4.559
G_{19}	27,769	352,285	5,446	0.355	4.473
G_{20}	75,879	405,740	5,530	0.421	5.198
G_{21}	34,546	420,877	6,547	0.485	6.517
G_{22}	82,168	504,230	11,010	0.722	8.993
G_{23}	334,863	925,872	12,290	1.048	11.259
G_{24}	317,080	1,049,866	13,604	0.839	12.811
G_{25}	262,111	899,792	16,839	1.299	15.030
G_{26}	325,729	1,090,108	21,049	1.568	17.753
G_{27}	281,903	1,992,636	32,117	1.987	32.571
G_{28}	1,134,890	2,987,624	37,813	2.502	41.554
G_{29}	1,088,092	1,541,898	45,020	2.740	44.258
G_{30}	403,394	2,443,408	46,731	2.830	50.362
G_{31}	1,379,917	1,921,660	57,846	3.806	54.122
G_{32}	2,394,385	4,659,565	64,910	4.360	67.348
G_{33}	875,713	4,322,051	73,613	5.357	70.936
G_{34}	1,965,206	2,766,607	85,729	5.030	82.916
G_{35}	685,230	6,649,470	107,554	7.270	111.137
G_{36}	1,696,415	11,095,298	145,611	11.212	161.926
G_{37}	3,774,768	16,518,947	273,963	22.588	436.311

Table 9: Number of vertices and edges before and after kernelization methods

	before kernelization		after Kernelization Method 1		after Kernelization Methods 2 or 3	
graph	vertices	edges	remaining vertices	remaining edges	remaining vertices	remaining edges
G_1	6,474	12,572	0	0	0	0
G_2	6,301	20,777	0	0	0	0
G_3	8,114	26,013	0	0	376	1,831
G_4	8,717	31,525	0	0	8	12
G_5	8,846	31,839	0	0	0	0
G_6	5,241	14,484	932	4,705	1,821	6,777
G_7	10,876	39,994	7	12	7	12
G_8	22,687	54,705	0	0	32	50
G_9	9,875	25,973	1,261	3,918	3,265	7,812
G_{10}	26,518	65,369	0	0	9	14
G_{11}	36,682	88,328	0	0	35	51
G_{12}	7,115	100,762	0	0	0	0
G_{13}	62,586	147,892	0	0	34	50
G_{14}	23,133	93,439	9,794	45,047	14,572	58,133
G_{15}	12,006	118,489	4,018	52,317	6,803	67,859
G_{16}	36,692	183,831	8,146	24,386	13,040	37,378
G_{17}	265,009	364,481	0	0	206	377
G_{18}	18,771	198,050	12,173	147,157	14,828	159,346
G_{19}	27,769	352,285	21,186	301,724	23,903	315,630
G_{20}	75,879	405,740	383	844	1,289	2,571
G_{21}	34,546	420,877	28,137	353,776	31,002	366,267
G_{22}	82,168	504,230	169	349	2,587	4,651
G_{23}	334,863	925,872	60,418	162,125	162,945	383,115
G_{24}	317,080	1,049,866	74,362	242,996	130,679	365,303
G_{25}	262,111	899,792	236,289	839,893	248,472	861,548
G_{26}	325,729	1,090,108	64,299	538,502	98,491	630,690
G_{27}	281,903	1,992,636	157,881	1,344,733	223,715	1,607,308
G_{28}	1,134,890	2,987,624	1,110	2,182	5,416	8,377
G_{29}	1,088,092	1,541,898	256,190	453,914	535,248	811,606
G_{30}	403,394	2,443,408	341,989	2,196,491	374,257	2,278,969
G_{31}	1,379,917	1,921,660	295,902	520,928	644,913	966,772
G_{32}	2,394,385	4,659,565	9	15	159	251
G_{33}	875,713	4,322,051	309,886	1,899,593	445,336	2,422,882
G_{34}	1,965,206	2,766,607	470,400	832,096	1,012,251	1,518,651
G_{35}	685,230	6,649,470	427,115	4,963,176	546,996	5,668,384
G_{36}	1,696,415	11,095,298	109,905	502,953	448,326	1,566,274
G_{37}	3,774,768	16,518,947	1,002,264	5,992,430	1,569,851	7,615,801

Table 10: Number of vertices and edges before and after kernelization methods in percent, where 100 percent is the number of vertices or the number of edges in a graph

	before kernelization		after Kernelization Method 1		after Kernelization Methods 2 or 3	
graph	vertices [%]	edges [%]	remaining vertices [%]	remaining edges [%]	remaining vertices [%]	remaining edges [%]
G_1	100	100	0	0	0	0
G_2	100	100	0	0	0	0
G_3	100	100	0	0	4.63	7.04
G_4	100	100	0	0	0.09	0.04
G_5	100	100	0	0	0	0
G_6	100	100	17.78	32.48	34.75	46.79
G_7	100	100	0.06	0.03	0.06	0.03
G_8	100	100	0	0	0.14	0.09
G_9	100	100	12.77	15.08	33.06	30.08
G_{10}	100	100	0	0	0.03	0.02
G_{11}	100	100	0	0	0.10	0.06
G_{12}	100	100	0	0	0	0
G_{13}	100	100	0	0	0.05	0.03
G_{14}	100	100	42.34	48.21	62.99	62.21
G_{15}	100	100	33.47	44.15	56.66	57.27
G_{16}	100	100	22.20	13.27	35.54	20.33
G_{17}	100	100	0	0	0.08	0.10
G_{18}	100	100	64.85	74.30	78.99	80.46
G_{19}	100	100	76.29	85.65	86.08	89.60
G_{20}	100	100	0.50	0.21	1.70	0.63
G_{21}	100	100	81.45	84.06	89.74	87.02
G_{22}	100	100	0.21	0.07	3.15	0.92
G_{23}	100	100	18.04	17.51	48.66	41.38
G_{24}	100	100	23.45	23.15	41.21	34.80
G_{25}	100	100	90.15	93.34	94.80	95.75
G_{26}	100	100	19.74	49.40	30.24	57.86
G_{27}	100	100	56.01	67.49	79.36	80.66
G_{28}	100	100	0.10	0.07	0.48	0.28
G_{29}	100	100	23.54	29.44	49.19	52.64
G_{30}	100	100	84.78	89.89	92.78	93.27
G_{31}	100	100	21.44	27.11	46.74	50.31
G_{32}	100	100	0	0	0.01	0.01
G_{33}	100	100	35.39	43.95	50.85	56.06
G_{34}	100	100	23.94	30.08	51.51	54.89
G_{35}	100	100	62.33	74.64	79.83	85.25
G_{36}	100	100	6.48	4.53	26.43	14.12
G_{37}	100	100	26.55	36.28	41.59	46.10

Table 11: Computation time of kernelization

graph	Kernelization Method 1	Kernelization Method 2	Kernelization Method 3
G_1	0.041	0.026	0.029
G_2	0.032	0.021	0.023
G_3	0.035	0.025	0.021
G_4	0.027	0.020	0.020
G_5	0.024	0.019	0.018
G_6	0.007	0.004	0.004
G_7	0.034	0.022	0.022
G_8	0.040	0.035	0.036
G_9	0.015	0.012	0.012
G_{10}	0.052	0.046	0.046
G_{11}	0.070	0.061	0.063
G_{12}	0.086	0.076	0.077
G_{13}	0.126	0.108	0.108
G_{14}	0.045	0.027	0.027
G_{15}	0.051	0.032	0.032
G_{16}	0.116	0.098	0.096
G_{17}	0.349	0.369	0.352
G_{18}	0.051	0.028	0.028
G_{19}	0.061	0.034	0.033
G_{20}	0.323	0.308	0.305
G_{21}	0.081	0.047	0.047
G_{22}	0.385	0.369	0.368
G_{23}	0.847	0.566	0.560
G_{24}	0.719	0.583	0.575
G_{25}	0.111	0.074	0.072
G_{26}	0.399	0.355	0.345
G_{27}	1.322	0.849	0.840
G_{28}	2.735	2.743	2.709
G_{29}	1.497	0.976	1.022
G_{30}	0.389	0.212	0.210
G_{31}	1.826	1.299	1.293
G_{32}	7.489	7.505	7.467
G_{33}	2.422	1.915	1.906
G_{34}	2.750	1.868	1.808
G_{35}	5.027	3.465	3.353
G_{36}	13.028	11.266	11.350
G_{37}	13.070	8.924	9.343

Table 12: Percent of edges of a maximum matching found during kernelizations

graph	maximum matching [%]	edges of a maximum matching found during Kernelization Method 1 [%]	edges of a maximum matching found during Kernelization Methods 2 or 3 [%]
G_1	100	100.00	100.00
G_2	100	100.00	100.00
G_3	100	100.00	98.02
G_4	100	100.00	99.94
G_5	100	100.00	100.00
G_6	100	80.55	62.17
G_7	100	99.93	99.93
G_8	100	100.00	99.85
G_9	100	86.11	64.26
G_{10}	100	100.00	99.97
G_{11}	100	100.00	99.86
G_{12}	100	100.00	100.00
G_{13}	100	100.00	99.92
G_{14}	100	55.73	34.45
G_{15}	100	64.70	40.43
G_{16}	100	67.52	47.94
G_{17}	100	100.00	99.90
G_{18}	100	33.57	19.16
G_{19}	100	22.94	13.09
G_{20}	100	99.18	97.29
G_{21}	100	17.98	9.63
G_{22}	100	99.70	96.60
G_{23}	100	80.46	48.26
G_{24}	100	73.79	54.13
G_{25}	100	9.70	5.04
G_{26}	100	57.95	45.98
G_{27}	100	41.24	16.69
G_{28}	100	99.81	99.08
G_{29}	100	75.79	49.49
G_{30}	100	15.04	7.03
G_{31}	100	77.93	51.99
G_{32}	100	99.99	99.91
G_{33}	100	56.29	40.20
G_{34}	100	75.39	47.12
G_{35}	100	33.98	14.76
G_{36}	100	92.51	69.31
G_{37}	100	68.71	52.01

Table 13: The number of vertices and edges in kernel: the theoretical bound (see Theorem 3.1 on page 18) and the empirically measured values with Kernelization Method 1. For comparison the number of vertices and edges in the original graphs are also given.

	number of vertices			number of edges		
graph	theoretical bound for kernel	in kernel after Kernelization Method 1	in original graph	theoretical bound for kernel	in kernel after Kernelization Method 1	in original graph
G_1	24,396	0	6,474	30,495	0	12,572
G_2	57,912	0	6,301	72,390	0	20,777
G_3	71,620	0	8,114	89,525	0	26,013
G_4	91,236	0	8,717	114,045	0	31,525
G_5	91,984	0	8,846	114,980	0	31,839
G_6	38,388	932	5,241	47,985	4,705	14,484
G_7	116,476	7	10,876	145,595	12	39,994
G_8	128,124	0	22,687	160,155	0	54,705
G_9	66,100	1,261	9,875	82,625	3,918	25,973
G_{10}	155,448	0	26,518	194,310	0	65,369
G_{11}	206,632	0	36,682	258,290	0	88,328
G_{12}	374,684	0	7,115	468,355	0	100,762
G_{13}	341,272	0	62,586	426,590	0	147,892
G_{14}	283,492	9,794	23,133	354,365	45,047	93,439
G_{15}	427,036	4,018	12,006	533,795	52,317	118,489
G_{16}	592,816	8,146	36,692	741,020	24,386	183,831
G_{17}	460,412	0	265,009	575,515	0	364,481
G_{18}	718,272	12,173	18,771	897,840	147,157	198,050
G_{19}	1,298,632	21,186	27,769	1,623,290	301,724	352,285
G_{20}	1,319,452	383	75,879	1,649,315	844	405,740
G_{21}	1,545,568	28,137	34,546	1,931,960	353,776	420,877
G_{22}	1,688,252	169	82,168	2,110,315	349	504,230
G_{23}	2,364,040	60,418	334,863	2,955,050	162,125	925,872
G_{24}	2,931,148	74,362	317,080	3,663,935	242,996	1,049,866
G_{25}	2,550,728	236,289	262,111	3,188,410	839,893	899,792
G_{26}	3,057,520	64,299	325,729	3,821,900	538,502	1,090,108
G_{27}	6,844,392	157,881	281,903	8,555,490	1,344,733	1,992,636
G_{28}	7,410,940	1,110	1,134,890	9,263,675	2,182	2,987,624
G_{29}	1,816,048	256,190	1,088,092	2,270,060	453,914	1,541,898
G_{30}	8,160,084	341,989	403,394	10,200,105	2,196,491	2,443,408
G_{31}	2,168,668	295,902	1,379,917	2,710,835	520,928	1,921,660
G_{32}	9,070,940	9	2,394,385	11,338,675	15	4,659,565
G_{33}	13,796,336	309,886	875,713	17,245,420	1,899,593	4,322,051
G_{34}	3,216,156	470,400	1,965,206	4,020,195	832,096	2,766,607
G_{35}	23,859,664	427,115	685,230	29,824,580	4,963,176	6,649,470

Table 14: Ratios of the computation times of finding a maximum matching with specified method and Edmonds' Blossom Shrinking to computation time of finding a maximum matching with Edmonds' Blossom Shrinking without kernelization

graph	without kernelization	with Kernelization Method 1	with Kernelization Method 2	with Kernelization Method 3
G_1	1	0.01077	0.00957	0.00957
G_2	1	0.01663	0.01330	0.01330
G_3	1	0.01764	0.03234	0.02499
G_4	1	0.01138	0.00828	0.00828
G_5	1	0.00967	0.00767	0.00767
G_6	1	0.08273	0.23022	0.23112
G_7	1	0.00790	0.00642	0.00617
G_8	1	0.00421	0.00375	0.00524
G_9	1	0.02545	0.14642	0.14466
G_{10}	1	0.00442	0.00394	0.00386
G_{11}	1	0.00212	0.00184	0.00178
G_{12}	1	0.01764	0.00868	0.00853
G_{13}	1	0.00070	0.00064	0.00060
G_{14}	1	0.13090	0.31320	0.32022
G_{15}	1	0.08623	0.27445	0.29564
G_{16}	1	0.05578	0.13587	0.15208
G_{17}	1	0.00014	0.00014	0.00013
G_{18}	1	0.29864	0.48308	0.51096
G_{19}	1	0.30206	0.56124	0.62441
G_{20}	1	0.00113	0.00118	0.00116
G_{21}	1	0.36795	0.69524	0.84536
G_{22}	1	0.00117	0.00155	0.00154
G_{23}	1	0.02295	0.16576	0.17432
G_{24}	1	0.06873	0.21546	0.22583
G_{25}	1	0.74456	0.71185	0.72642
G_{26}	1	0.08963	0.20657	0.21158
G_{27}	1	0.28741	0.54455	0.53910
G_{28}	1	0.00003	0.00004	0.00004

Table 15: Ratios of the computation times of finding a maximum matching with specified method and Edmonds' Blossom Shrinking to computation time of finding a maximum matching with Kernelization Method 1 and Edmonds' Blossom Shrinking

graph	with Kernelization Method 1	with Kernelization Method 2	with Kernelization Method 3
G_1	1	0.89	0.89
G_2	1	0.80	0.80
G_3	1	1.83	1.42
G_4	1	0.73	0.73
G_5	1	0.79	0.79
G_6	1	2.78	2.79
G_7	1	0.81	0.78
G_8	1	0.89	1.24
G_9	1	5.75	5.68
G_{10}	1	0.89	0.87
G_{11}	1	0.87	0.84
G_{12}	1	0.49	0.48
G_{13}	1	0.92	0.87
G_{14}	1	2.39	2.45
G_{15}	1	3.18	3.43
G_{16}	1	2.44	2.73
G_{17}	1	0.98	0.95
G_{18}	1	1.62	1.71
G_{19}	1	1.86	2.07
G_{20}	1	1.05	1.03
G_{21}	1	1.89	2.30
G_{22}	1	1.32	1.31
G_{23}	1	7.22	7.59
G_{24}	1	3.13	3.29
G_{25}	1	0.96	0.98
G_{26}	1	2.30	2.36
G_{27}	1	1.89	1.88
G_{28}	1	1.25	1.20

Table 16: Size of graphs and speed-up of computation of finding a maximum matching with Kernelization Method 1 and Edmonds' Blossom Shrinking compared to Edmonds' Blossom Shrinking without kernelization. The graphs are sorted on the number of edges.

graph	vertices	edges	speed up
G_1	6,474	12,572	92.9
G_6	5,241	14,484	12.1
G_2	6,301	20,777	60.2
G_9	9,875	25,973	39.3
G_3	8,114	26,013	56.7
G_4	8,717	31,525	87.9
G_5	8,846	31,839	103.4
G_7	10,876	39,994	126.5
G_8	22,687	54,705	237.3
G_{10}	26,518	65,369	226.3
G_{11}	36,682	88,328	470.9
G_{14}	23,133	93,439	7.6
G_{12}	7,115	100,762	56.7
G_{15}	12,006	118,489	11.6
G_{13}	62,586	147,892	1,435.3
G_{16}	36,692	183,831	17.9
G_{18}	18,771	198,050	3.3
G_{19}	27,769	352,285	3.3
G_{17}	265,009	364,481	7,184.6
G_{20}	75,879	405,740	887.4
G_{21}	34,546	420,877	2.7
G_{22}	82,168	504,230	852.2
G_{25}	262,111	899,792	1.3
G_{23}	334,863	925,872	43.6
G_{24}	317,080	1,049,866	14.5
G_{26}	325,729	1,090,108	11.2
G_{27}	281,903	1,992,636	3.5
G_{28}	1,134,890	2,987,624	29,367.0

Table 17: Reduction of graphs with Kernelization Method 1 and reduction of computation times of finding a maximum matching with Kernelization Method 1 and Edmonds' Blossom Shrinking in comparison to Edmonds' Blossom Shrinking without kernelization

graph	vertices [%]	edges [%]	computation time [%]
G_1	100.00	100.00	98.92
G_2	100.00	100.00	98.34
G_3	100.00	100.00	98.24
G_4	100.00	100.00	98.86
G_5	100.00	100.00	99.03
G_6	82.22	67.52	91.73
G_7	99.94	99.97	99.21
G_8	100.00	100.00	99.58
G_9	87.23	84.92	97.45
G_{10}	100.00	100.00	99.56
G_{11}	100.00	100.00	99.79
G_{12}	100.00	100.00	98.24
G_{13}	100.00	100.00	99.93
G_{14}	57.66	51.79	86.91
G_{15}	66.53	55.85	91.38
G_{16}	77.80	86.73	94.42
G_{17}	100.00	100.00	99.99
G_{18}	35.15	25.70	70.14
G_{19}	23.71	14.35	69.79
G_{20}	99.50	99.79	99.89
G_{21}	18.55	15.94	63.20
G_{22}	99.79	99.93	99.88
G_{23}	81.96	82.49	97.70
G_{24}	76.55	76.85	93.13
G_{25}	9.85	6.66	25.54
G_{26}	80.26	50.60	91.04
G_{27}	43.99	32.51	71.26
G_{28}	99.90	99.93	100.00

Table 18: Computation time of finding a maximum matching with Kernelization Method 2 and Edmonds' Blossom shrinking algorithm in detail

graph	kernel [s]	matching [s]	postprocessing [s]	total [s]
G_1	0.007	0	0.001	0.008
G_2	0.015	0	0.001	0.016
G_3	0.026	0.039	0.001	0.066
G_4	0.023	0	0.001	0.024
G_5	0.022	0	0.001	0.023
G_6	0.010	0.245	0.001	0.256
G_7	0.024	0	0.002	0.026
G_8	0.038	0.001	0.001	0.040
G_9	0.012	0.567	0.002	0.581
G_{10}	0.048	0	0.001	0.049
G_{11}	0.063	0	0.002	0.065
G_{12}	0.060	0	0	0.060
G_{13}	0.115	0	0.003	0.118
G_{14}	0.027	23.480	0.003	23.510
G_{15}	0.032	12.813	0.001	12.846
G_{16}	0.099	11.795	0.003	11.897
G_{17}	0.372	0.001	0.004	0.377
G_{18}	0.029	77.044	0.002	77.075
G_{19}	0.035	162.106	0.003	162.144
G_{20}	0.314	0.048	0.005	0.367
G_{21}	0.049	321.139	0.004	321.192
G_{22}	0.377	0.152	0.007	0.536
G_{23}	0.594	4,339.966	0.040	4,340.600
G_{24}	0.586	5,276.740	0.035	5,277.361
G_{25}	0.071	12,461.396	0.057	12,461.524
G_{26}	0.339	1,678.027	0.016	1,678.382
G_{27}	0.866	16,042.361	0.039	16,043.266
G_{28}	2.899	0.821	0.087	3.807

Table 19: Computation time of finding a maximum matching with Kernelization Method 3 and Edmonds' Blossom shrinking algorithm in detail

graph	kernel [s]	matching [s]	postprocessing [s]	total [s]
G_1	0.007	0	0.001	0.008
G_2	0.015	0	0.001	0.016
G_3	0.026	0.024	0.001	0.051
G_4	0.022	0	0.002	0.024
G_5	0.022	0	0.001	0.023
G_6	0.008	0.248	0.001	0.257
G_7	0.024	0	0.001	0.025
G_8	0.055	0	0.001	0.056
G_9	0.010	0.562	0.002	0.574
G_{10}	0.046	0	0.002	0.048
G_{11}	0.062	0	0.001	0.063
G_{12}	0.058	0	0.001	0.059
G_{13}	0.108	0	0.003	0.111
G_{14}	0.025	24.008	0.004	24.037
G_{15}	0.034	13.802	0.002	13.838
G_{16}	0.095	13.219	0.003	13.317
G_{17}	0.359	0	0.005	0.364
G_{18}	0.028	81.492	0.002	81.522
G_{19}	0.032	180.359	0.003	180.394
G_{20}	0.304	0.051	0.005	0.360
G_{21}	0.045	390.497	0.004	390.546
G_{22}	0.366	0.158	0.006	0.530
G_{23}	0.549	4,564.309	0.042	4,564.900
G_{24}	0.592	5,530.710	0.034	5,531.336
G_{25}	0.063	12,716.516	0.058	12,716.637
G_{26}	0.339	1,718.689	0.015	1,719.043
G_{27}	0.836	15,881.826	0.039	15,882.701
G_{28}	2.738	0.861	0.084	3.683

Table 20: Computation time of finding a maximum matching with Kernelization Method 1 and Edmonds' Blossom shrinking algorithm. There are no computation time of Edmonds' Blossom shrinking algorithm without kernelization for $G_{29}, \ldots, G_{36}$ because these graphs could not be solved within the time limit.

graph	kernel + postprocessing [s]	matching in kernel [s]	matching in graph [s]
G_1	0.009	0	0.836
G_2	0.020	0	1.203
G_3	0.036	0	2.041
G_4	0.033	0	2.900
G_5	0.029	0	2.998
G_6	0.009	0.083	1.112
G_7	0.032	0	4.049
G_8	0.044	0.001	10.680
G_9	0.017	0.084	3.968
G_{10}	0.055	0	12.447
G_{11}	0.075	0	35.318
G_{12}	0.122	0	6.915
G_{13}	0.128	0	183.713
G_{14}	0.046	9.780	75.063
G_{15}	0.052	3.984	46.807
G_{16}	0.120	4.764	87.563
G_{17}	0.384	0	2,758.893
G_{18}	0.055	47.592	159.548
G_{19}	0.065	87.200	288.904
G_{20}	0.346	0.004	310.607
G_{21}	0.088	169.902	461.986
G_{22}	0.404	0.001	345.124
G_{23}	0.860	600.255	26,186.697
G_{24}	0.788	1,682.641	24,493.126
G_{25}	0.169	13,034.091	17,505.910
G_{26}	0.423	727.801	8,124.839
G_{27}	1.398	8,466.103	29,461.539
G_{28}	3.019	0.038	89,774.879
G_{29}	1.881	9,982.338	
G_{30}	0.687	54,760.604	
G_{31}	2.485	11,363.179	
G_{32}	7.584	0	
G_{33}	2.691	43,363.358	
G_{34}	3.107	33,996.467	
G_{36}	13.409	1,595.557	

Table 21: Computation time of finding a maximum matching without kernelization

graph	vertices	edges	matching size	Edmonds' Blossom Shrinking [s]	Blossom V [s]	speed up
G_1	6,474	12,572	1,048	0.836	0.015	55.7
G_2	6,301	20,777	2,054	1.203	0.048	25.1
G_3	8,114	26,013	2,574	2.041	0.069	29.6
G_4	8,717	31,525	3,405	2.900	0.112	25.9
G_5	8,846	31,839	3,428	2.998	0.122	24.6
G_6	5,241	14,484	2,329	1.112	0.018	61.8
G_7	10,876	39,994	4,348	4.049	0.200	20.2
G_8	22,687	54,705	6,017	10.680	0.247	43.2
G_9	9,875	25,973	4,457	3.968	0.054	73.5
G_{10}	26,518	65,369	7,208	12.447	0.365	34.1
G_{11}	36,682	88,328	9,268	35.318	0.642	55.0
G_{12}	7,115	100,762	2,249	6.915	0.303	22.8
G_{13}	62,586	147,892	15,693	183.713	1.562	117.6
G_{14}	23,133	93,439	10,970	75.063	0.855	87.8
G_{15}	12,006	118,489	5,649	46.807	0.345	135.7
G_{16}	36,692	183,831	12,198	87.563	1.324	66.1
G_{17}	265,009	364,481	18,315	2,758.893	1.180	2,338.0
G_{18}	18,771	198,050	9,150	159.548	0.571	279.4
G_{19}	27,769	352,285	13,746	288.904	1.950	148.2
G_{20}	75,879	405,740	21,960	310.607	3.827	81.2
G_{21}	34,546	420,877	17,150	461.986	1.900	243.2
G_{22}	82,168	504,230	25,565	345.124	10.750	32.1
G_{24}	317,080	1,049,866	138,074	24,493.126	159.601	153.5
G_{25}	262,111	899,792	130,822	17,505.910	3.690	4,744.1
G_{26}	325,729	1,090,108	66,160	8,124.839	2.472	3,286.7

Table 22: Speed-up of computation of finding a maximum matching with kernelization compared to computation of finding a maximum matching without kernelization

graph	Kernelization Method 1	Kernelization Method 2	Kernelization Method 3
G_1	1.9	1.9	1.7
G_2	1.9	4.1	4.1
G_3	2.4	2.4	2.6
G_4	6.1	6.9	6.3
G_5	5.3	6.4	6.4
G_6	0.9	1.0	0.9
G_7	6.5	8.1	8.1
G_8	5.6	6.2	6.6
G_9	1.7	1.5	1.5
G_{10}	7.0	5.7	7.8
G_{11}	8.6	9.4	9.5
G_{12}	3.5	4.6	5.3
G_{13}	12.4	13.0	13.4
G_{14}	2.8	3.6	3.6
G_{15}	2.3	1.0	1.0
G_{16}	6.8	6.1	6.2
G_{17}	3.3	2.9	3.0
G_{18}	1.7	1.4	1.4
G_{19}	1.2	1.2	1.2
G_{20}	11.1	11.8	12.1
G_{21}	1.1	1.1	1.1
G_{22}	26.6	27.5	28.3
G_{23}	48.0	19.5	19.8
G_{24}	76.5	38.0	38.2
G_{25}	1.1	1.3	1.3
G_{26}	3.1	2.9	2.9
G_{27}	2.9	2.4	2.5
G_{30}	9.3	90.6	90.9
G_{32}	24.7	27.8	27.9
G_{35}	3.8	2.6	2.6

Table 23: Computation time of finding a maximum matching with kernelization and Blossom V

graph	kernel and postprocessing of Kernelization Method 1 [s]	kernel and postprocessing of Kernelization Method 2 [s]	matching in kernel from Kernelization Method 1 [s]	matching in kernel from Kernelization Method 2 [s]
G_1	0.008	0.008	0	0
G_2	0.030	0.014	0	0
G_3	0.027	0.023	0	0.005
G_4	0.025	0.022	0	0
G_5	0.024	0.020	0	0
G_6	0.007	0.005	0.009	0.010
G_7	0.030	0.024	0.001	0.001
G_8	0.046	0.039	0	0.002
G_9	0.016	0.012	0.015	0.023
G_{10}	0.053	0.065	0	0.001
G_{11}	0.073	0.065	0	0.002
G_{12}	0.088	0.067	0	0
G_{13}	0.132	0.119	0	0.007
G_{14}	0.048	0.031	0.258	0.205
G_{15}	0.051	0.035	0.096	0.311
G_{16}	0.117	0.100	0.078	0.117
G_{17}	0.358	0.372	0	0.033
G_{18}	0.056	0.030	0.226	0.314
G_{19}	0.066	0.038	1.586	1.601
G_{20}	0.337	0.318	0.010	0.011
G_{21}	0.088	0.053	1.663	1.716
G_{22}	0.399	0.382	0.011	0.015
G_{23}	0.967	0.652	0.368	2.626
G_{24}	0.804	0.650	1.286	3.555
G_{25}	0.181	0.142	3.178	2.864
G_{26}	0.425	0.368	0.357	0.480
G_{27}	1.393	0.916	6.168	7.906
G_{30}	0.508	0.315	390.658	39.731
G_{32}	8.645	7.582	0.203	0.304
G_{35}	5.121	3.561	17.222	28.811

Table 24: Computation time of finding a maximum matching with Kernelization Method 1 and Blossom V in detail

graph	kernel [s]	matching [s]	postprocessing [s]	total [s]
G_1	0.008	0	0	0.008
G_2	0.029	0	0.001	0.030
G_3	0.026	0	0.001	0.027
G_4	0.024	0	0.001	0.025
G_5	0.022	0	0.002	0.024
G_6	0.006	0.009	0.001	0.016
G_7	0.028	0.001	0.002	0.031
G_8	0.042	0	0.004	0.046
G_9	0.015	0.015	0.001	0.031
G_{10}	0.051	0	0.002	0.053
G_{11}	0.070	0	0.003	0.073
G_{12}	0.088	0	0	0.088
G_{13}	0.128	0	0.004	0.132
G_{14}	0.044	0.258	0.004	0.306
G_{15}	0.050	0.096	0.001	0.147
G_{16}	0.113	0.078	0.004	0.195
G_{17}	0.354	0	0.004	0.358
G_{18}	0.054	0.226	0.002	0.282
G_{19}	0.062	1.586	0.004	1.652
G_{20}	0.332	0.010	0.005	0.347
G_{21}	0.082	1.663	0.006	1.751
G_{22}	0.391	0.011	0.008	0.410
G_{23}	0.913	0.368	0.054	1.335
G_{24}	0.754	1.286	0.050	2.090
G_{25}	0.113	3.178	0.068	3.359
G_{26}	0.403	0.357	0.022	0.782
G_{27}	1.346	6.168	0.047	7.561
G_{30}	0.405	390.658	0.103	391.166
G_{32}	8.632	0.203	0.013	8.848
G_{35}	5.005	17.222	0.116	22.343

Table 25: Computation time of finding a maximum matching with Kernelization Method 2 and Blossom V in detail

graph	kernel [s]	matching [s]	postprocessing [s]	total [s]
G_1	0.008	0	0	0.008
G_2	0.013	0	0.001	0.014
G_3	0.022	0.005	0.001	0.028
G_4	0.021	0	0.001	0.022
G_5	0.019	0	0.001	0.020
G_6	0.004	0.010	0.001	0.015
G_7	0.023	0.001	0.001	0.025
G_8	0.037	0.002	0.002	0.041
G_9	0.011	0.023	0.001	0.035
G_{10}	0.064	0.001	0.001	0.066
G_{11}	0.063	0.002	0.002	0.067
G_{12}	0.066	0	0.001	0.067
G_{13}	0.116	0.007	0.003	0.126
G_{14}	0.027	0.205	0.004	0.236
G_{15}	0.033	0.311	0.002	0.346
G_{16}	0.096	0.117	0.004	0.217
G_{17}	0.368	0.033	0.004	0.405
G_{18}	0.028	0.314	0.002	0.344
G_{19}	0.034	1.601	0.004	1.639
G_{20}	0.312	0.011	0.006	0.329
G_{21}	0.048	1.716	0.005	1.769
G_{22}	0.375	0.015	0.007	0.397
G_{23}	0.597	2.626	0.055	3.278
G_{24}	0.602	3.555	0.048	4.205
G_{25}	0.072	2.864	0.070	3.006
G_{26}	0.347	0.480	0.021	0.848
G_{27}	0.868	7.906	0.048	8.822
G_{30}	0.213	39.731	0.102	40.046
G_{32}	7.566	0.304	0.016	7.886
G_{35}	3.424	28.811	0.137	32.372

Table 26: Computation time of finding a maximum matching with Kernelization Method 3 and Blossom V in detail

graph	kernel [s]	matching [s]	postprocessing [s]	total [s]
G_1	0.009	0	0	0.009
G_2	0.014	0	0	0.014
G_3	0.022	0.002	0.001	0.025
G_4	0.023	0	0.001	0.024
G_5	0.019	0	0.001	0.020
G_6	0.006	0.009	0.001	0.016
G_7	0.023	0.001	0.001	0.025
G_8	0.036	0.001	0.002	0.039
G_9	0.012	0.021	0.002	0.035
G_{10}	0.045	0.001	0.002	0.048
G_{11}	0.062	0.002	0.002	0.066
G_{12}	0.057	0	0.001	0.058
G_{13}	0.112	0.006	0.004	0.122
G_{14}	0.028	0.205	0.004	0.237
G_{15}	0.033	0.312	0.002	0.347
G_{16}	0.095	0.116	0.003	0.214
G_{17}	0.356	0.034	0.004	0.394
G_{18}	0.029	0.312	0.003	0.344
G_{19}	0.033	1.596	0.004	1.633
G_{20}	0.302	0.011	0.006	0.319
G_{21}	0.047	1.710	0.005	1.762
G_{22}	0.365	0.014	0.007	0.386
G_{23}	0.566	2.613	0.055	3.234
G_{24}	0.581	3.549	0.049	4.179
G_{25}	0.074	2.848	0.069	2.991
G_{26}	0.346	0.469	0.021	0.836
G_{27}	0.861	7.877	0.047	8.785
G_{30}	0.195	39.610	0.102	39.907
G_{32}	7.517	0.305	0.015	7.837
G_{35}	3.286	28.547	0.135	31.968

Table 27: Computation time of finding a maximum matching with Kernelization Method 1

graph	Edmonds' Blossom Shrinking [s]	Blossom V [s]
G_1	0.009	0.008
G_2	0.020	0.030
G_3	0.036	0.027
G_4	0.033	0.025
G_5	0.029	0.024
G_6	0.092	0.016
G_7	0.032	0.031
G_8	0.045	0.046
G_9	0.101	0.031
G_{10}	0.055	0.053
G_{11}	0.075	0.073
G_{12}	0.122	0.088
G_{13}	0.128	0.132
G_{14}	9.826	0.306
G_{15}	4.036	0.147
G_{16}	4.884	0.195
G_{17}	0.384	0.358
G_{18}	47.647	0.282
G_{19}	87.265	1.652
G_{20}	0.350	0.347
G_{21}	169.990	1.751
G_{22}	0.405	0.410
G_{24}	1,683.429	2.090
G_{25}	13,034.260	3.359
G_{26}	728.225	0.782
G_{27}	8,467.501	7.561
G_{30}	54,761.294	391.166
G_{32}	7.584	8.848

Table 28: Size of found matching in percent, where 100 percent is the size of a maximum matching

	Path Growing Weighted Matching			Greedy Weighted Matching		
graph	without kernelization [%]	with Kernelization Method 1 [%]	with Kernelization Methods 2 or 3 [%]	without kernelization [%]	with Kernelization Method 1 [%]	with Kernelization Methods 2 or 3 [%]
G_1	88.36	100.00	100.00	87.21	100.00	100.00
G_2	78.38	100.00	100.00	87.44	100.00	100.00
G_3	78.79	100.00	99.84	88.19	100.00	99.92
G_4	75.74	100.00	100.00	82.94	100.00	100.00
G_5	76.23	100.00	100.00	84.13	100.00	100.00
G_6	82.78	99.06	95.15	84.41	99.01	95.41
G_7	75.09	100.00	100.00	81.92	100.00	100.00
G_8	77.10	100.00	100.00	87.15	100.00	100.00
G_9	79.29	98.61	94.14	81.18	98.54	94.62
G_{10}	76.25	100.00	100.00	87.00	100.00	100.00
G_{11}	76.31	100.00	100.00	87.93	100.00	99.99
G_{12}	77.59	100.00	100.00	80.61	100.00	100.00
G_{13}	75.55	100.00	100.00	88.94	100.00	100.00
G_{14}	80.99	94.68	89.49	82.91	94.97	90.58
G_{15}	80.01	97.04	91.26	82.12	97.26	91.64
G_{16}	82.46	96.61	92.43	82.70	96.69	92.54
G_{17}	98.68	100.00	100.00	98.83	100.00	99.99
G_{18}	81.28	90.71	86.67	83.06	91.50	87.97
G_{19}	84.78	90.79	88.45	85.77	91.07	88.79
G_{20}	77.83	99.97	99.63	76.43	99.96	99.59
G_{21}	81.50	87.17	84.94	81.61	86.97	84.63
G_{22}	76.68	99.98	99.71	76.68	99.98	99.70
G_{23}	84.28	97.97	92.95	86.04	98.01	93.28
G_{24}	81.15	97.89	94.12	82.28	97.88	94.38
G_{25}	87.36	89.76	88.70	87.81	90.01	89.09
G_{26}	88.18	97.67	96.08	88.88	97.89	96.33
G_{27}	89.75	93.60	91.42	89.76	93.42	91.47
G_{28}	83.42	100.00	99.92	83.19	100.00	99.90
G_{29}	86.16	98.39	95.91	86.20	98.33	96.20
G_{30}	85.41	89.47	87.38	85.41	89.51	87.49
G_{31}	85.83	98.59	96.20	85.93	98.53	96.53
G_{32}	85.08	100.00	99.99	83.32	100.00	99.99
G_{33}	83.51	93.64	90.81	83.97	93.27	90.49
G_{34}	86.72	98.33	95.76	86.90	98.26	95.92
G_{35}	88.34	92.16	90.06	88.66	92.24	90.38
G_{36}	86.19	99.22	96.03	86.82	99.21	96.09
G_{37}	87.11	95.75	93.18	89.38	95.64	93.35

Table 29: Computation time of finding a matching with Path Growing Weighted Matching with and without kernelization

graph	without kernelization	with Kernelization Method 1 [s]	with Kernelization Method 2 [s]	with Kernelization Method 3 [s]
G_1	0.066	0.010	0.010	0.011
G_2	0.060	0.019	0.017	0.016
G_3	0.120	0.033	0.027	0.027
G_4	0.056	0.028	0.022	0.022
G_5	0.052	0.021	0.018	0.019
G_6	0.014	0.007	0.008	0.009
G_7	0.065	0.028	0.024	0.025
G_8	0.327	0.044	0.038	0.039
G_9	0.046	0.018	0.019	0.020
G_{10}	0.726	0.052	0.046	0.047
G_{11}	1.022	0.074	0.066	0.067
G_{12}	0.071	0.092	0.061	0.059
G_{13}	3.634	0.128	0.117	0.118
G_{14}	0.176	0.082	0.118	0.119
G_{15}	0.082	0.070	0.072	0.073
G_{16}	0.898	0.151	0.203	0.193
G_{17}	84.711	0.370	0.383	0.381
G_{18}	0.175	0.134	0.157	0.158
G_{19}	0.360	0.252	0.247	0.245
G_{20}	4.331	0.351	0.324	0.319
G_{21}	0.499	0.463	0.473	0.469
G_{22}	4.474	0.407	0.396	0.391
G_{23}	32.109	1.793	6.435	6.388
G_{24}	38.514	1.826	5.390	5.369
G_{25}	13.592	11.238	12.308	12.293
G_{26}	75.500	1.751	7.262	7.265
G_{27}	32.111	13.261	23.473	23.266
G_{28}	1,053.707	2.922	2.916	2.895
G_{29}	339.874	11.546	53.780	52.999
G_{30}	46.917	18.283	21.534	21.433
G_{31}	449.632	13.132	63.019	62.947
G_{32}	6,913.250	7.404	7.779	7.615
G_{33}	565.209	33.603	104.633	104.447
G_{34}	1,251.901	39.070	200.372	199.967
G_{35}	211.761	126.354	154.355	153.855
G_{36}	2,653.465	20.887	137.142	137.066
G_{37}	6,075.878	238.300	411.213	409.511

Table 30: Computation time of finding a matching with Greedy Weighted Matching with and without kernelization

graph	without kernelization	with Kernelization Method 1 [s]	with Kernelization Method 2 [s]	with Kernelization Method 3 [s]
1	0.011	0.022	0.018	0.018
2	0.020	0.047	0.035	0.024
3	0.016	0.029	0.020	0.016
4	0.008	0.038	0.029	0.030
5	0.008	0.023	0.018	0.019
6	0.003	0.013	0.010	0.009
7	0.005	0.029	0.024	0.025
8	0.008	0.044	0.039	0.037
9	0.003	0.020	0.014	0.014
10	0.010	0.053	0.047	0.045
11	0.014	0.081	0.066	0.064
12	0.012	0.083	0.061	0.059
13	0.026	0.131	0.117	0.112
14	0.014	0.052	0.039	0.039
15	0.015	0.058	0.042	0.043
16	0.025	0.126	0.113	0.106
17	0.057	0.368	0.373	0.358
18	0.026	0.074	0.052	0.052
19	0.049	0.109	0.083	0.082
20	0.058	0.344	0.323	0.311
21	0.060	0.140	0.107	0.105
22	0.071	0.409	0.384	0.375
23	0.509	0.992	0.729	0.660
24	0.199	0.853	0.726	0.697
25	0.197	0.362	0.326	0.330
26	0.167	0.501	0.463	0.468
27	0.318	1.613	1.176	1.170
28	0.528	2.977	2.930	2.822
29	0.414	1.903	1.436	1.331
30	0.464	0.923	0.747	0.740
31	0.489	2.343	2.439	1.690
32	0.676	17.462	7.650	7.636
33	0.826	2.871	2.477	2.237
34	0.617	3.448	2.725	2.501
35	0.411	5.805	4.110	3.830
36	1.412	33.846	11.694	11.485
37	3.256	15.159	10.850	10.916

Table 31: Computation time of finding a matching with Path Growing Weighted Matching with Kernelization Method 1 in detail

graph	kernel [%]	matching [%]	postprocessing [%]
G_1	90.00	10.00	0
G_2	94.74	0	5.26
G_3	93.94	0	6.06
G_4	92.86	3.57	3.57
G_5	95.24	0	4.76
G_6	71.43	14.29	14.29
G_7	92.86	3.57	3.57
G_8	93.18	0	6.82
G_9	83.33	5.56	11.11
G_{10}	96.15	0	3.85
G_{11}	95.95	0	4.05
G_{12}	100.00	0	0
G_{13}	96.88	0	3.13
G_{14}	53.66	42.68	3.66
G_{15}	71.43	25.71	2.86
G_{16}	78.15	19.21	2.65
G_{17}	98.92	0	1.08
G_{18}	39.55	58.96	1.49
G_{19}	25.00	73.41	1.59
G_{20}	98.29	0.28	1.42
G_{21}	18.36	80.56	1.08
G_{22}	98.28	0	1.72
G_{23}	50.03	47.02	2.96
G_{24}	41.73	55.48	2.79
G_{25}	1.01	98.43	0.55
G_{26}	23.30	75.44	1.26
G_{27}	10.36	89.35	0.29
G_{28}	96.89	0.07	3.05
G_{29}	13.78	84.20	2.02
G_{30}	2.28	97.22	0.50
G_{31}	14.96	82.94	2.10
G_{32}	99.78	0	0.22
G_{33}	7.88	91.76	0.36
G_{34}	7.04	91.87	1.09
G_{35}	3.92	96.00	0.08
G_{36}	63.64	35.38	0.98
G_{37}	5.77	93.91	0.31

Table 32: Computation time of finding a matching with Greedy Weighted Matching with Kernelization Method 1 in detail

graph	kernel [%]	matching [%]	postprocessing [%]
G_1	100.00	0	0
G_2	95.74	0	4.26
G_3	93.10	0	6.90
G_4	94.74	0	5.26
G_5	91.30	0	8.70
G_6	76.92	7.69	15.38
G_7	93.10	0	6.90
G_8	95.45	0	4.55
G_9	90.00	0	10.00
G_{10}	98.11	0	1.89
G_{11}	96.30	0	3.70
G_{12}	100.00	0	0
G_{13}	96.95	0	3.05
G_{14}	82.69	11.54	5.77
G_{15}	87.93	8.62	3.45
G_{16}	93.65	3.17	3.17
G_{17}	98.64	0	1.36
G_{18}	70.27	25.68	4.05
G_{19}	57.80	38.53	3.67
G_{20}	98.26	0	1.74
G_{21}	60.71	35.71	3.57
G_{22}	98.29	0	1.71
G_{23}	91.43	3.02	5.54
G_{24}	88.51	5.16	6.33
G_{25}	30.94	50.83	18.23
G_{26}	80.64	14.77	4.59
G_{27}	84.00	13.02	2.98
G_{28}	96.94	0	3.06
G_{29}	81.82	6.04	12.14
G_{30}	43.99	45.72	10.29
G_{31}	81.82	5.68	12.51
G_{32}	99.91	0	0.09
G_{33}	85.55	10.17	4.28
G_{34}	81.18	6.18	12.65
G_{35}	89.13	8.92	1.95
G_{36}	99.22	0.17	0.61
G_{37}	86.92	8.32	4.76

Table 33: Computation time of finding a matching with heuristics with Kernelization Method 1

graph	Path Growing Weighted Matching	Greedy Weighted Matching
G_1	0.010	0.022
G_2	0.019	0.047
G_3	0.033	0.029
G_4	0.028	0.038
G_5	0.021	0.023
G_6	0.007	0.013
G_7	0.028	0.029
G_8	0.044	0.044
G_9	0.018	0.020
G_{10}	0.052	0.053
G_{11}	0.074	0.081
G_{12}	0.092	0.083
G_{13}	0.128	0.131
G_{14}	0.082	0.052
G_{15}	0.070	0.058
G_{16}	0.151	0.126
G_{17}	0.370	0.368
G_{18}	0.134	0.074
G_{19}	0.252	0.109
G_{20}	0.351	0.344
G_{21}	0.463	0.140
G_{22}	0.407	0.409
G_{23}	1.793	0.992
G_{24}	1.826	0.853
G_{25}	11.238	0.362
G_{26}	1.751	0.501
G_{27}	13.261	1.613
G_{28}	2.922	2.977
G_{29}	11.546	1.903
G_{30}	18.283	0.923
G_{31}	13.132	2.343
G_{32}	7.404	17.462
G_{33}	33.603	2.871
G_{34}	39.070	3.448
G_{35}	126.354	5.805
G_{36}	20.887	33.846
G_{37}	238.300	15.159