Exact Matching: Algorithms and Related Problems

Nicolas El Maalouly
Department of Computer Science, ETH Zürich, Switzerland

Abstract

In 1982, Papadimitriou and Yannakakis introduced the *Exact Matching* (EM) problem where given an edge colored graph, with colors red and blue, and an integer k, the goal is to decide whether or not the graph contains a perfect matching with exactly k red edges. Although they conjectured it to be **NP**-complete, soon after it was shown to be solvable in randomized polynomial time in the seminal work of Mulmuley et al., placing it in the complexity class **RP**. Since then, all attempts at finding a deterministic algorithm for EM have failed, thus leaving it as one of the few natural combinatorial problems in **RP** but not known to be contained in **P**, and making it an interesting instance for testing the hypothesis **RP** = **P**. Progress has been lacking even on very restrictive classes of graphs despite the problem being quite well known as evidenced by the number of works citing it.

In this paper we aim to gain more insight into EM by studying a new optimization problem we call *Top-k Perfect Matching* (TkPM) which we show to be polynomially equivalent to EM. By virtue of being an optimization problem, it is more natural to approximate TkPM so we provide approximation algorithms for it. Some of the approximation algorithms rely on a relaxation of EM on bipartite graphs where the output is required to be a perfect matching with a number of red edges differing from k by at most $k/2$, which is of independent interest and generalizes to the *Exact Weight Perfect Matching* (EWPM) problem. We also consider parameterized algorithms and show that TkPM can be solved in FPT time parameterized by k and the independence number of the graph. This result again relies on new tools developed for EM which are also of independent interest.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms; Theory of computation → Approximation algorithms analysis; Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases Perfect Matching, Exact Matching, Approximation algorithms, Independence number, Parameterized complexity

Digital Object Identifier 10.4230/LIPIcs.STACS.2023.29

Related Version *Full Version*: `https://arxiv.org/abs/2203.13899`

Acknowledgements I want to thank Ola Svensson for introducing me to the Exact Matching problem, for defining the Top-k Perfect Matching problem and for helpful discussions and feedback.

1 Introduction

Deciding whether randomization adds power to sequential algorithms is an central problem in complexity theory. The main question there is whether **P** = **RP** which remains a big open problem and is tied to other important questions in the field [22]. Only few natural problems are known to be in **RP** while no deterministic algorithms are known for them. Exact Matching (EM), defined in 1982 by Papadimitriou and Yannakakis [27], is one such problem.

EXACT MATCHING (EM)
Input: A graph G, with each edges colored red or blue, and integer k.
Task: Decide whether there exists a perfect matching M in G with exactly k red edges.

At the time of its introduction it was conjectured to be **NP**-complete. Only a few years later, however, it was shown to be in **RP** by Mulmuley, Vazirani and Vazirani [26], which makes it unlikely to be **NP**-hard. In fact, in was even shown to be in **RNC** which is defined as the class of decision problems allowing an algorithm running in polylogarithmic time[1] (i.e., $O(\log n^c)$ for some constant $c > 0$) using polynomially many parallel processors, while having additional access to randomness (we refer the interested reader to [6] Chapter 12 for a formal definition). Derandomizing matching problems from this complexity class is also a big open problem [30]. This makes EM even more interesting since randomness allows it to be efficiently parallelizable while it even remains difficult to solve sequentially without such access to randomness.

The interest in EM is evidenced by the numerous works that cite it. These include works on the parallel computation complexity of the matching problem [30], planarizing gadgets for perfect matchings [20], multicriteria optimization [19], matroid intersection [7], DNA sequencing [5], binary linear equation systems with small hamming weight [2], recoverable robust assignment [15] in addition to generalizations of the problem with multiple color constraints [4, 24, 25, 29]. Despite that, deciding whether EM is in **P** has remained an open problem for almost four decades and little progress has been made even for very restricted classes of graphs, thus highlighting the surprising difficulty of the problem.

1.1 Prior Work

Restricted Graph Classes. When it comes to restricted classes of graphs, results go in two directions. The first is the sparse graphs regime where in the extreme case we have trees for which EM can be solved by a simple dynamic program (DP). This can also be generalized to bounded tree-width graphs. Such a DP has not been explicitly given in the literature but would be easy to construct by keeping track of how every edge in a bag is matched (not yet matched, matched outside the bag or matched inside the bag) as well as the total number of red edges in the matching so far. It is easy to see that the number of possible states is at most $O(3^{tw} \cdot n)$ (where n comes from the possible number of red edges in the matching) resulting in an FPT algorithm parameterized by the tree width of the graph. Continuing with sparse graph classes, EM is also known to be solvable for planar graphs [32] by relying on the existence of Pfaffian orientations to derandomize the **RNC** algorithm. The same techniques used for this derandomization also allow for computing the matching generating function (see [18] Chapter 1 for a definition) and can be generalized to other graph classes such as $K_{3,3}$-minor free graphs and graphs embeddable on a surface of bounded genus [16]. Computing the matching generating function was recently shown to be #**P**-hard already for K_8-minor free graphs [9] so these results do not generalize much further and are restricted to very sparse graphs.

The second direction is dense graph classes. Here it is known that EM is in **P** for complete and complete bipartite graphs, i.e., graphs of independence number $\alpha = 1$ and bipartite graphs of bipartite independence number[2] $\beta = 1$. In fact, these results are already non-trivial and at least four different articles have appeared on resolving them [23, 31, 17, 21]. Very recently, however, El Maalouly and Steiner [13] pushed the boundary of positive results further by showing that EM is in **P** for all graphs of bounded independence number and all bipartite graphs of bounded bipartite independence number.

[1] in the following, n denotes the number of vertices of the input graph

[2] The *bipartite independence number* of a bipartite graph G equipped with a bipartition of its vertices is defined as the largest number β such that G contains a *balanced independent set* of size 2β, i.e., an independent set using exactly β vertices from each side of the bipartition.

Generalizations. As mentioned above, prior work also considered a generalization of the problem to multiple color constraints, known as Bounded Color Matching (BCM).

Bounded Color Matching (BCM)
Input: A weighted and edge-colored (with colors $c_1, ..., c_l$) graph G and integers $k_1, ..., k_l$.
Task: Find a maximum weight matching M in G with at most k_i edges of color i for all $i \in \{1, ..., l\}$.

BCM is known to be NP-hard [28]. Mastrolilli and Stamoulis [25] provide bi-criteria approximation schemes which give an approximately maximum matching with small constraint violations. Stamoulis [29] also gives a 1/2-approximation for the objective with no constraint violations. No prior work considered bounds on the constraint violations while requiring an optimal objective, i.e., a perfect matching if the graph is unweighted. For EM, however, Yuster [32] proved that given an instance of the problem, one can decide in polynomial time that either G contains no perfect matching with exactly k red edges, or one can compute an *almost* perfect matching (i.e., of size at least $\lfloor \frac{n}{2} \rfloor - 1$) containing k red edges. This means that the techniques used in the bi-criteria approximation of BCM do not provide much further insight into solving EM since they relax the perfect matching requirement.

Another way to generalize the problem is to have a weighted instead of edge-colored graph, and require the output perfect matching to have an exact weight.

Exact Weight Perfect Matching (EWPM)
Input: A weighted graph G and integer W.
Task: Find a perfect matching M in G with $w(M) = W$.

EWPM can be reduced to EM if the edge weights are polynomial in the input size but is known to be NP-hard for exponential weights [20]. This makes approximation algorithms that aim to minimize the constraint violation even more desirable for EWPM.

1.2 Our contribution

Exact Matching. We provide an algorithm for a relaxed version of EM on bipartite graphs where we require the output to be a perfect matching and allow for a constraint violation that is a constant fraction of k, 0.5 in this case.

▶ **Theorem 1.** *There exists a deterministic polynomial time algorithm that, given a "Yes" instance of EM on a bipartite graph, outputs a perfect matching M with $0.5k \leq |R(M)| \leq 1.5k$, where $|R(M)|$ is the number of red edges in M.*

This can also be seen as an attempt to approximate the EM problem without relaxing the perfect matching constraint. This type of approximation is the first of its kind for EM. Note that in light of the above mentioned result by Yuster [32] (i.e., an algorithm that outputs an *almost* perfect matching containing k red edges) one would think that it should not be too difficult to find an algorithm for the relaxed version of EM with a constraint violation of only one red edge. However, the perfect matching requirement seems to be intrinsic to the difficulty of the problem (given the simplicity of Yuster's algorithm) and many attempts at improving the constraint violation of Theorem 1 have failed so far. We also show that the approximation algorithm works for the more general problem of EWPM, only loosing $1/poly(n)$ in the approximation factor for exponential weights.

▶ **Corollary 2.** ($\star$) *There exists a deterministic polynomial time algorithm that, given a "Yes" instance of EWPM on a bipartite graph with input weights bounded by a polynomial (resp. exponential) function of the input size, outputs a perfect matching M with $0.5W \leq w(M) \leq 1.5W$ (resp. $(0.5 - 1/poly(n))W \leq w(M) \leq (1.5 + 1/poly(n))W$).*

We also introduce a new way of finding alternating cycles with certain color and weight properties in FPT time parameterized by the number of edges in the cycles.

▶ **Proposition 3.** *Let $G = (V, E, w)$ be an edge colored and weighted graph with edge colors red and blue, and let M and M' be two perfect matchings in G and $\mathcal{C} = M \Delta M'$ s.t. $|E(\mathcal{C})| \leq L$ for some integer L. Then there exists an algorithm running in time $f(L)poly(n)$ (for $f(L) = L^{O(L)}$) that, given G and M as input, outputs a perfect matching M'' in G with $w(M'') \geq w(M')$ and $|R(M'')| = |R(M')|$.*

This allows us to get an FPT algorithm parameterized by the circumference of the graph.

▶ **Theorem 4.** *There exists a deterministic FPT algorithm, parameterized by the circumference[3] of the graph, for the Exact Matching problem in general graphs.*

Top-k Perfect Matching. The above studied problems suffer from the fact that they are not optimization problems (due to the exactness constraint which requires the optimization of more than one objective) and are thus less natural to approximate. For this reason, we study a new matching problem called Top-k Perfect Matching.

Top-k Perfect Matching (TkPM)
Input: A weighted graph G and integer k.
Task: Find a perfect matching in G maximizing the top-k weight function.

Here the top-k weight function is defined as the sum of the weights of the k highest weight edges in the matching. To our knowledge, this problem has not yet been considered in the literature, but similar types of optimization objectives have been used for other problems such as k-clustering and load balancing [8]. We show that this problem can also be reduced to EM (in deterministic polynomial time) when the edge weights are polynomially bounded in the input size.

▶ **Theorem 5.** *TkPM $\leq_p$ EM for polynomially bounded weights.*[4]

This puts TkPM with polynomial weights in the class **RP** and it remains open whether or not it is in **P**, thus making it another natural problem in this category. Interestingly, a recent result shows that EM can in turn be reduced to TkPM, making the two problems polynomially equivalent.

▶ **Lemma 6** (from [14]). *EM $\leq_p$ TkPM for polynomially bounded weights.*

This means that progress on TkPM not only provides further insight into EM, but could also help solve it directly. As previously mentioned, the main advantage of TkPM over the other studied variants of EM is that it is an optimization problem, i.e., we are maximizing a single objective function. This makes it more suitable for approximation and we provide approximation algorithms for it.

▶ **Theorem 7.** ($\star$) *There exists a deterministic polynomial time 0.5-approximation algorithm for TkPM.*

▶ **Theorem 8.** *There exists a deterministic polynomial time $(0.8 - 1/poly(n))$-approximation algorithm for TkPM on bipartite graphs.*

[3] The circumference of a graph is the length of any longest cycle in the graph.
[4] For two problems A and B, $A \leq_p B$ means that A is reducible to B in deterministic polynomial time and $A \equiv_p B$ implies both $A \leq_p B$ and $B \leq_p A$.

It is interesting to note that the main tool used for the proof of Theorem 1 (i.e., Proposition 11) was originally developed to prove Theorem 8. This shows how the study of TkPM can indeed provide insight into the EM problem.

Finally, the techniques we developed for FPT algorithms for EM so far only resulted in an FPT algorithm parameterized by the circumference of the graph. The circumference, however, is usually quite large and not very good as a parameter, so to better illustrate the use of these techniques, we combine them with techniques from [13] to show the existence of an FPT algorithm for TkPM parameterized by k and α (the independence number of the input graph), and an FPT algorithm for TkPM on bipartite graphs parameterized by k and β (the bipartite independence number of the input graph).

▶ **Theorem 9.** *There exists a deterministic algorithm for TkPM running in time $f(k,\alpha)poly(n)$ where $f(k,\alpha) = (k4^{\alpha})^{O(k4^{\alpha})}$ and α is the independence number of the input graph.*

▶ **Theorem 10.** ($\star$) *There exists a deterministic algorithm for TkPM on bipartite graphs running in time $f(k,\beta)poly(n)$ where $f(k,\beta) = (k\beta)^{O(k\beta)}$ and β is the bipartite independence number of the input graph.*

1.3 Organization of the paper

The remainder of this paper is organized as follows: In Section 2 we present the basic definitions and conventions we use throughout the paper. In Section 3 and Section 4 we study EM and TkPM respectively, both from the perspectives of approximation and parameterized algorithms. Finally in Section 5 we conclude the paper and provide some open problems.

2 Preliminaries

Due to space restrictions, proofs of statements marked ($\star$) have been deferred to the full version of this paper [12]. All graphs considered are simple. For a red/blue edge colored graph G and G' a subgraph of G, we define $R(G')$ (resp. $B(G')$) to be the set of red (resp. blue) edges in G' and $w(G')$ to be the sum of the weights of edges in G'. Undirected cycles are considered to have an arbitrary orientation. For a cycle C and $u, v \in C$, $C[u, v]$ is defined as the path from u to v along C (in the fixed but arbitrarily chosen orientation if C is undirected). Given a matching M, C is called M-alternating if for any two adjacent edges in C, one of them is in M and the other is not. An e edge is called a matching edge if $e \in M$ and a non-matching edge if $e \notin M$.

We always assume that for problems on weighted graphs, the input weights are given as positive integers and their encoding size is part of the input (i.e., they can be at most exponential in the input size if they are encoded in binary). We use w to refer to the set of weights in a weighted graph $G = (V, E, w)$. We always consider a strict ordering on the edges in which the edges are ordered by decreasing weight with ties broken arbitrarily (but the ordering is fixed for a given graph and weight function). The top-k weight function $w^k(E)$ for a set of edges E is defined as the sum of the first k edges from E in the edge ordering of the graph, i.e., $w^k(E) = \sum_{i\in\{1...k\}} w(E(i))$ where $E(i)$ is the i-th edge from E in the edge ordering of the graph.

3 Exact Matching

3.1 Approximation Algorithms

In this section, we aim to prove Theorem 1 by developing a deterministic polynomial time algorithm for EM where we require the output to be a perfect matching (abbreviated PM) and allow for a constraint violation that is a constant fraction of k. More precisely we require the output PM to have between $0.5k$ and $1.5k$ red edges. The main tool we use is the following proposition which allows us to increase the number of red edges of a PM without adding too many such edges.

▶ **Proposition 11.** *Let $G := (V, E)$ be an edge weighted directed graph containing a directed cycle C with $w(C) > 0$ and C contains at most k edges having strictly positive weight. There exists a deterministic polynomial time algorithm that, given G, finds a directed cycle in G with the same properties as C.*

Proof of Proposition 11. For simplicity, we will flip the sign of all weights so that we are looking for a negative cycle which can be found by a shortest path algorithm. In the following we will use the Bellman-Ford algorithm which relies on a dynamic program (DP) to compute the distance between any two nodes in the graph [3]. By adding an extra constraint variable to the DP, we are also able to compute the shortest path weights for paths that fulfill some bound on the constraint. More formally we start with the normal update rule for the Bellman-Ford algorithm:

$$d(s, v) = \min_{u \in V} \{d(s, u) + \bar{w}(u, v) | (u, v) \in E\}$$

where $\bar{w}(u, v) = -w(e)$ (i.e., we flip the sign of the weights) for $e = (u, v)$ and $d(s, v)$ is the distance from s to v where the length of an edge is given by its weight $\bar{w}$ (note that every vertex is considered to have a self loop of weight 0). We modify it to include the constraint variable (with an extra dimension in the table entries of the DP to account for it):

$$d(s, v, c) = \min_{u \in V} \{d(s, u, c - \mathbb{1}_{\bar{w}(u,v)<0}) + \bar{w}(u, v) | (u, v) \in E\}$$

where $\mathbb{1}$ is the indicator variable which takes value 1 if the condition is true and 0 otherwise, so the constraint variable is decreased every time the path uses a red edge. The entries $d(s, v, c)$ are initialized to ∞ for all $s, v \in V(G)$ and $c \in \{-1, 0, 1, 2, ..., k\}$, except for the entries of the form $d(s, s, c)$, for all $s \in V(G)$ and $c \in \{0, 1, 2, ..., k\}$, which are initialized to 0. This way, after running the update rule on the DP until convergence or until some entry of the form $d(s, s, c)$ becomes negative (i.e., a strictly negative cycle is detected), the value of $d(s, v, c)$ corresponds to the weight of the shortest path from s to v, containing at most c red edges, if such a path exists and is ∞ otherwise, unless there is a negative cycle. Note that the table entries can be computed iteratively, starting with entries of the form $d(s, v, 0)$ (the computation is the same as the regular Bellman-ford algorithm but with the new update rule) then increasing c by 1 every time. Finally observe that if a strictly negative cycle C containing at most k edges of strictly negative weight (i.e., positive in the original edge weight before the sign flip) exists, at least one of the entries of the form $d(s, s, c)$ for $s \in C$ and $0 \leq c \leq k$ will become negative (since the shortest path from s to itself should have negative length). Such a cycle is guaranteed by the conditions of the proposition and computing it can be done by a standard modification of the DP that keeps track of the last used edge for each updated entry. The output cycle is guaranteed to be strictly positive and have at most k strictly positive weight edges (in the original graph before the sign flip). Note that the running time of the DP is polynomial in the number of table entries, which in turn is polynomial in the size of the input graph. ◀

By repeatedly applying Proposition 11 we are able to find a PM fulfilling the requirements of Theorem 1.

Proof of Theorem 1. Let M be a PM containing a minimum number of red edges (should be at most k since we have a "Yes" instance). Note that M can be computed in polynomial time by simply using a maximum weight perfect matching algorithm [11], with -1 weights assigned to red edges and 0 weights assigned to blue edges. If $|R(M)| \geq 0.5k$ we are done, so suppose $|R(M)| < 0.5k$. We define the directed graph G' in the following way. We start with the bipartite input graph $G = (A \cup B, E)$ and orient the edges as follows: edges in M are oriented from A to B and edges not in M are oriented from B to A. This way we are guaranteed that any directed cycle in the resulting graph is an M-alternating cycle. We also define edge weights as follows: blue edges get weight 0, red edges in M get weight -1 and red edges not in M get weight $+1$. This way we have that for any M-alternating cycle C, $M' := M\Delta C$ is a perfect matching with $|R(M')| = |R(M)| + w(C)$. Note that $M\Delta M'$ is a set of disjoint cycles that are both M-alternating and M'-alternating.

Let M^* be a solution to the EM instance, i.e., $|R(M^*)| = k$ (which must exist since we are given a "Yes" instance). Observe that $w(M\Delta M^*) = |R(M^*)| - |R(M)| > 0$ so there must be a cycle $C \in M\Delta M^*$ s.t. $w(C) > 0$. Also note that M^* contains exactly k red edges so $M\Delta M^*$ contains at most k red edges not in M (i.e., edges of strictly positive weight). Finally note that the cycle C is a directed cycle (since it is alternating). So we can use Proposition 11 on the resulting graph to find a cycle C' with $w(C') > 0$ containing at most k edges of strictly positive weight. Note that $w(C') \leq k$ since edges have weight at most $+1$. Now we let $M' := M\Delta C'$ (this is possible since C' being a directed cycle implies that it must be an M-alternating cycle). Note that $|R(M)| < |R(M)| + w(C') \leq |R(M)| + k < 1.5k$ and $|R(M')| = |R(M\Delta C')| = |R(M)| + w(C')$. So if $|R(M')| \geq 0.5k$ the algorithm stops and outputs $M\Delta C'$, otherwise we repeat the above procedure, with M' replacing M, until $|R(M')| \geq 0.5k$. The running time is polynomial since the above procedure runs in polynomial time (by Proposition 11) and it is repeated at most k times. ◀

3.2 FPT Algorithms

In this section we start by proving Proposition 3 which provides a new tool for finding alternating cycles with color and weight constraints in FPT time parameterized by the size of the cycles.

Proof of Proposition 3. Our goal is to find a set of M-alternating disjoint cycles $\mathcal{C}'$ in G with the same number of matching (i.e., edges in M) and non-matching (i.e., edges not in M) red edges as $\mathcal{C}$ and weights that are at least as big, i.e., for every $C \in \mathcal{C}$ there must be a $C' \in \mathcal{C}'$ such that C' has the same number of matching and non-matching red edges as C and $w(C') \geq w(C)$ (and vice versa, i.e., there is a one to one correspondence between the cycles in $\mathcal{C}$ and the cycles in $\mathcal{C}'$). This way we construct $M'' := M\Delta\mathcal{C}'$ s.t. $w(M'') = w(M\Delta\mathcal{C}') \geq w(M\Delta\mathcal{C}) = w(M')$ and $|R(M'')| = |R(M\Delta\mathcal{C}')| = |R(M\Delta\mathcal{C})| = |R(M')|$.

Color Coding. The main tool for finding such a set of cycles is color coding [1]. The idea is to color all vertices at random with L colors. The probability that all vertices of $\mathcal{C}$ get different colors is only a function of L. This can also be achieved deterministically using a perfect hash family of size bounded by $L^{O(L)}poly(n)$, which can be guaranteed to contain at least one coloring for which all vertices of $\mathcal{C}$ have different colors (see [10] Chapter 5 for more details on derandomizing color coding).

Separating the Cycles. Observe that for every cycle in $C \in \mathcal{C}$, the following can also be achieved in $L^{O(L)}poly(n)$ time.

- Guess the set of colors $colors(C)$ of its vertices and their exact order.
- Guess its number of matching (i.e., in M) and non-matching (i.e., not in M) red edges.

Let G_C be the graph induced on the vertices of G with a color from the set $colors(C)$. Observe that C is contained in G_C and that the subgraphs G_C for $C \in \mathcal{C}$ are all disjoint. So we can look for each cycle separately.

Orienting the Cycles. Since we know the colors of the vertices of $\mathcal{C}$, we can define a bipartition (A, B) of G by splitting the set of colors into two equal parts and letting A (resp. B) be the vertices having a color from the first (resp. second), s.t. the cycles in $\mathcal{C}$ are alternating with respect to the bipartition (note that this is indeed possible since the cycles in $\mathcal{C}$ are M-alternating so they have even length). By deleting all edges with endpoints in the same part, we get a bipartite graph which contains $\mathcal{C}$. Now we can define the following orientation for the edges: edges in M are oriented from A to B and edges not in M are oriented from B to A. This way we are guaranteed that any directed cycle in the resulting graph is an alternating cycle.

From Cycles to Colorful Paths. For this part and the next, we look into one cycle $C \in \mathcal{C}$ and its corresponding subgraph G_C. Let $(c_1, c_2, ...) := colors(C)$. We first guess the edge of C with start vertex from color class $c_{|colors(C)|}$ and end vertex from color class c_1 (this can be done in polynomial time by trying all possibilities). Then we delete all edges from G_C except for the edges going from a vertex of color c_i to a vertex of color c_{i+1} for $i \in \{1, 2, ..., |colors(C)| - 1\}$. Observe that G_C is now acyclic and the remaining edges of C form a directed path from s to t in G_C.

Finding the Paths. For simplicity, we will flip the sign of all weights so that we are looking for paths of minimum weight which can be found by a shortest path algorithm. Similarly to the proof of Theorem 1 we use a modified Bellman-Ford algorithm, so we will focus on the main difference, i.e., the update rule. By adding extra constraint variables to the DP, we are also able to compute the shortest path weights for paths that fulfill an exact constraint. Note that this is only possible since the graph is acyclic (otherwise the algorithm can output non-simple paths). More formally the update rule for the Bellman-Ford algorithm is the following:

$$d(s, v, rm, rn) = \min_{u \in V} \{d(s, u, rm - \mathbb{1}_{(u,v) \in R_m}, rn - \mathbb{1}_{(u,v) \in R_n}) + \bar{w}(u, v) | (u, v) \in E\}$$

where $G_C = (V, E)$, $\bar{w}(u, v) = -w(e)$ for $e = (u, v)$, R_m is the set of matching red edges in G_C and R_n is the set of non-matching red edges in G_C. We are interested in the value $d(s, t, |(C \backslash (t, s)) \cap R_m|, |(C \backslash (t, s)) \cap R_n|)$ where $|(C \backslash (t, s)) \cap R_m|$ is the number of matching red edges in $C \backslash (t, s)$ and $|(C \backslash (t, s)) \cap R_n|$ is the number of non-matching red edges in $C \backslash (t, s)$. The DP runs in polynomial time since the number of table entries is polynomial and allows us to find a simple path $path(C)$ (since the graph is acyclic) from s to t of minimum weight, i.e., $\bar{w}(path(C)) \leq \bar{w}(C \backslash (t, s))$ which implies $w(path(C)) \geq w(C \backslash (t, s))$, and with the same number of matching and non-matching red edges as $(C \backslash (t, s))$.

Constructing the set $\mathcal{C}'$. Finally for $C \in \mathcal{C}$, let $cycle(C) := path(C) \cup (t, s)$ with $path(C)$ computed using the above DP on the processed graph G_C. Let $\mathcal{C}' := \{cycle(C) | C \in \mathcal{C}\}$. Observe that $cycle(C)$ is a cycle with the same number of matching and non-matching red edges as C, $w(cycle(C)) \geq w(C)$ and all cycles in $\mathcal{C}'$ are M-alternating and disjoint. So $\mathcal{C}'$ fulfills all the required properties.

Running Time. Observe that all the above steps can be run in $L^{O(L)}poly(n)$ time, so the total running time is of the same order. ◀

The above proposition is the key to proving Theorem 4 which gives an FPT algorithm for EM parameterized by the circumference of the graph. But first we need the following lemma which ensures that we can always make progress using a set of alternating cycles of size bounded by a function of their individual lengths.

▶ **Lemma 12.** *Given a "Yes" instance of EM and a PM M, if $|R(M)| < k$ then there exists a set of disjoint M-alternating cycles $\mathcal{C}$ s.t. $|E(\mathcal{C})| \leq 2c^4$, where c is circumference of the graph, and $|R(M)| < |R(M\Delta\mathcal{C})| \leq k$.*

Proof of Theorem 4. Let M be a PM containing a minimum number of red edges (should be at most k). Note that M can be computed in polynomial time by simply using a maximum weight perfect matching algorithm with -1 weights assigned to red edges and 0 weights assigned to blue edges. From Lemma 12 we know that there exists a set of disjoint M-alternating cycles $\mathcal{C}$ s.t. $|E(\mathcal{C})| \leq 2c^3$ and $|R(M)| < |R(M\Delta\mathcal{C})| \leq k$. Let $M' := M\Delta\mathcal{C}$. Now by using Proposition 3 we can find a PM M'' with $|R(M'')| = |R(M')|$ (here we do not need to assign any weights to edges so the weight function used to apply the proposition can simply be uniform) so $|R(M)| < |R(M'')| \leq k$. We can repeat the procedure (applying Lemma 12 on M'') until we get a PM with exactly k red edges. We need at most k repetitions, each running in time $f(L)poly(n) = L^{O(L)}poly(n)$ for $L = O(c^3)$, i.e., we get an FPT algorithm parameterized by c. ◀

Theorem 4 illustrates the use the Proposition 3 to develop FPT algorithms for Exact Matching. However, the circumference of the graph can in general be quite large. We believe that Proposition 3 can be applied to get other more interesting FPT algorithms for EM and related problems. In Section 4.4 we show one such application.

4 Top-k Perfect Matching

In this section we study TkPM which, as we show later, is polynomial time equivalent to EM, making it another problem that can be used to test the hypothesis **P** = **RP**, but with the advantage of being an optimization problem.

4.1 Minimum Weight Variant

First, we start by introducing a variant of TkPM in which we are looking for a PM minimizing (instead of maximizing) the top-k weight. This objective function has been studied in the context of other problems such as k-clustering and load balancing [8] but to our knowledge, no prior work considered it in the context of matching problems.

MINIMUM WEIGHT TOP-k PERFECT MATCHING (MINTKPM)
Input: A weighted graph G and integer k.
Task: Find a perfect matching in G minimizing the top-k weight function.

We show however, that by simply applying a threshold to the weights of the edges, we are able to reduce this problem to minimum weight perfect matching (minWPM), i.e., it is in **P**. The proof crucially relies on the idea of thresholding the weights which will also be useful for the approximation algorithms in the next sections.

▶ **Definition 13.** *Given a weighted graph G with weights w, the thresholded weights w_t for a threshold t are defined as follows: for an edge e, $w_t(e) = \max(w(e) - t, 0)$.*

▶ **Theorem 14.** *$minTkPM \equiv_p minWPM$.*

Proof. $minWPM \leq_p minTkPM$ is trivial by setting $k = n/2$ so we need to prove $minTkPM \leq_p minWPM$. Given an instance of minTkPM, let M^* be an optimal PM. Let e_k be the kth edge from M^* in the edge ordering. The algorithm starts by guessing e_k (i.e., running for all possibilities of e_k and outputting the matching of smallest top-k value among all solutions) and setting $t := w(e_k)$. We have $w_t(M^*) = w_t^k(M^*) = w^k(M^*) - kt$ since the k values above t are reduced by t, and the rest is set to 0. Now let M be a minimum weight perfect matching in the thresholded graph. Then we have $w_t^k(M) \leq w_t(M) \leq w_t(M^*) \leq w(M^*) - kt$. After removing the threshold, each of the top-k values can only increase by at most t, so we get $w^k(M) \leq w(M^*)$, i.e., we get an optimal solution. ◀

This creates an interesting division between the minimization and maximization of the top-k values, in the context of a perfect matching problem. On the one hand we have a problem that is polynomially equivalent to the general weighted matching problem (known to be in **P**), and on the other hand we get a problem that is polynomially equivalent to EM (as we show next) whose complexity remains unknown.

4.2 Reducing Top-k Perfect Matching to Exact Matching

To help reduce TkPM to EM, we introduce an intermediary problem called maximum weight EM in which we are given an instance of EM as well as edge weights (of polynomial size) and the goal is to find a PM with exactly k red edges having maximum weight among all such PMs.

MAXIMUM WEIGHT EXACT MATCHING (MWEM)
Input: An edge-weighted and edge-colored (with red/blue colors) graph G and integer k.
Task: Find a perfect matching M in G with exactly k red edges and having maximum weight among all such matchings.

We show that this new variant can be reduced to EWPM (with polynomial weights) which in turn can be reduced to EM. This shows that MWEM is in **RP**.

▶ **Lemma 15.** ($\star$) *$MWEM \leq_p EWPM \leq_p EM$ for polynomially bounded weights. The reductions also work for bipartite input graphs and for minor closed graph classes.*

Note that even though MWEM is an optimization problem, any approximation for it requires solving EM. So our focus will instead be on TkPM which we reduce to EM when the input weights are polynomially bounded in the input size.

Proof of Theorem 5. We have $MWEM \leq_p EM$ from Lemma 15, so we need to show that $TkPM \leq_p MWEM$. Given an instance of TkPM, let M^* be an optimal solution. Let e_k be the kth edge from M^* in the edge ordering. The algorithm starts by guessing e_k (i.e., running for all possibilities of e_k and outputting the matching of highest top-k value among all solutions) and setting the weights of all edges after e_k in the ordering to 0 and coloring them blue, while the rest of the edges are colored red. Note that only red edges can have non-zero weights and that M^* has exactly k red edges. Let M be the output of an algorithm for MWEM on the resulting graph. By optimality of M, we have that $w(M) \geq w(M^*)$, and since they both contain at most k non-zero weight edges we get $w^k(M) \geq w^k(M^*)$ so M is an optimal solution for TkPM. Since we only modify the weights of the edges, the reduction preserves the graph class. ◀

The above lemma, in combination with the result of [14], implies the following theorem.

▶ **Theorem 16.** *TkPM $\equiv_p$ EM for polynomially bounded weights. The equivalence also holds for bipartite input graphs and for minor closed graph classes.*

Note that it is still open whether MWEM and TkPM with exponential weights are reducible to EM or if they are **NP**-hard.

4.3 Approximation Algorithms for Top-k Perfect Matching

Note that the reduction to EM in Lemma 15 does not preserve any approximation factor since it changes the weights of the edges. So we cannot use it in combination with Theorem 1 to get an approximation algorithm for TkPM. We will, however, use Proposition 11 to get a better approximation for TkPM as we will see later. First we show that by simply applying a specific threshold to the weights of the graph, any maximum weight perfect matching (maxWPM) algorithm can output a 0.5-approximation for TkPM.

▶ **Lemma 17.** *Given an instance of TkPM, let M^* be an optimal solution. There exists a threshold t such that for any maximum weight perfect matching M in the thresholded graph, we have $w^k(M) \geq 0.5 \cdot w^k(M^*)$ (in the original graph).*

Proof. Let $t = \frac{w^k(M^*)}{2k}$. Then we have $w_t(M^*) \geq w^k(M^*) - k \cdot \frac{w^k(M^*)}{2k} = 0.5w^k(M^*)$. And since M is a maximum weight perfect matching, we have $w_t(M) \geq w_t(M^*) \geq 0.5w^k(M^*)$. Let k' be the number of edges $e \in M$ with $w_t(e) > 0$. Now we have two cases. First, if $k' \leq k$ then we have $w^k(M) \geq w_t^k(M) \geq 0.5w^k(M^*)$. Otherwise the output matching contains at least k edge of weight more than $\frac{w^k(M^*)}{2k}$, so the total weight is $w^k(M) \geq k \cdot \frac{w^k(M^*)}{2k} = 0.5w^k(M^*)$. ◀

The above lemma guarantees the existence of a threshold that will lead to a 0.5-approximation using any maximum weight PM algorithm. We may not know the exact threshold, but if the weights are polynomial we can simply try all possibilities. Otherwise we can find a good threshold using binary search (see Algorithm 1).

In order to get a better approximation factor, we rely on Proposition 11 which allows us to limit the change in the number of edges with weight above threshold.

Proof Sketch of Theorem 8. We start with a high level intuition on how the algorithm works and why it gives a better approximation.

The core idea of the algorithm is the following: instead of recomputing the maximum weight perfect matching every time we change the threshold (as is done in the previous algorithm), we keep track of one perfect matching M which we incrementally improve using alternating cycles that increase its weight. We also make sure that the cycles do not add too many edges of weight above the threshold. This way the top-k weight of M stays closer to its total weight. To find such cycles, we rely on the algorithm of Proposition 11, which allows us to find positive alternating cycles that do not add too many positive weight edges (at most k). But first we set the edge weights of edges in M to negative (i.e., multiply them by -1). This way the weight of an alternating cycle indicates the total weight change we get when taking its symmetric difference with M to get a new perfect matching. This means that, whenever possible, we can increase the total weight of M while keeping its top-k weight close to its total weight (considering the thresholded weights) since the number of positive weight edges above the threshold is limited.

Algorithm 1 TkPM 0.5-approximation Algorithm.

```
Input: An instance of TkPM.
Output: PM $M$ with $w^k(M) \geq 0.5w(M^*)$ where $M^*$ is an optimal solution.
$t_1 \leftarrow 0$, $t_2 \leftarrow w_{max}$ ;  /* where $w_{max}$ is the maximum weight in the graph */
$M \leftarrow$ MaximumWeightPerfectMatching$(G, w_{t_1})$;
if $M$ contains at most $k$ edges $e$ with $w(e) > 0$ then
    return $M$;
else
    $M_1 \leftarrow$ MaximumWeightPerfectMatching$(G, w_{t_1})$;
    $M_2 \leftarrow$ MaximumWeightPerfectMatching$(G, w_{t_2})$;
    while $t_2 - t_1 \geq 1/(k^2)$ do
        $t \leftarrow (t_1 + t_2)/2$ ;
        $M \leftarrow$ MaximumWeightPerfectMatching$(G, w_t)$;
        if $M$ contains more than $k$ edges $e$ with $w_t(e) > 0$ then
            $t_1 \leftarrow t$; $M_1 \leftarrow M$;
        else
            $t_2 \leftarrow t$; $M_2 \leftarrow M$;
    $M \leftarrow$ BestOf$(M_1, M_2)$;
    return $M$;

Procedure BestOf$(M_1, M_2)$:
    if $w^k(M_1) \geq w^k(M_2)$ then
        return $M_1$;
    else
        return $M_2$;
```

To see why this is helpful, consider the two cases in the proof of Lemma 17: $k' \leq k$ and $k' \geq k$ (remember that k' is the number of edges in M with weight strictly above the threshold).

In the case $k' \leq k$ (let $k_1 := k'$), we know that the top-k weight is the same as the total weight (for the thresholded weights), which means that we do not lose anything when considering only the top-k weight (i.e., $w_t^k(M) = w_t(M) \geq w_t(M^*) = w_t^k(M^*)$). However, we might lose some value because of the threshold. This is because when we go back to the original weights, M^* regains up to $k \cdot t$ in value (k times the threshold, since all its top-k edges might have value above the threshold) whereas M might only regain $k_1 \cdot t$ (since all other edges could have original weight close to zero). So in the case $k_1 << k$, we lose almost all the value from the threshold.

On the other hand, in case $k' \geq k$ (let $k_2 := k'$), M will also regain $k \cdot t$ in top-k weight when we add back the threshold. However, the top-k weight of M can be far from its total weight since many edges can be contributing to the total weight. This is mainly a problem when $k_2 >> k$. If k' is close to k, however, this loss is not so big (at most a fraction $(k_2 - k)/k$ of the total since the k highest weights still count).

To get a worst case approximation factor of 0.5, it must be the case that both $k_1 << k$ and $k_2 >> k$. Note, however, that the procedure detailed above (relying on Proposition 11) allows us to bound the difference between k_1 and k_2 by k (i.e,. $k_2 - k_1 \leq k$). This way, the algorithm manages to guarantee a better approximation factor.

We are now ready to describe the full algorithm. We will first show how to transform exponential weights into polynomially bounded ones while loosing at most a factor of $1 - 1/poly(n)$ in the approximation. We then provide a $0.8 - 1/poly(n)$-approximation algorithm for TkPM with polynomially bounded weights, which proves the theorem.

Dealing with Exponential Weights. For exponential size weights, we first scale and round them to make them bounded by a polynomial function of the input size. We start by deleting all edges that cannot be part of any perfect matching (this can simply be done by checking for every edge whether we could remove it along with its endpoints from the graph and still be able to get a perfect matching on the rest of the graph). Let W be the highest edge weight in the remaining graph. Observe that for $k \geq 1$ an optimal solution M^* to the top-k perfect matching problem must have $w^k(M^*) \geq W$ (since the perfect matching containing the edge of weight W is a valid solution). Now all weight encodings have at most $\log_2(W)$ non-zero bits. Let $f(n) = n \cdot poly(n)$ for any desired polynomial. If $W \leq f(n)$ then all weights are polynomial. Otherwise we re-encode the weights of all edges by only considering their $(\log_2(W) - \log_2(f(n)))$-th to $(\log_2(W))$-th bits (counting from the least significant bit) and dropping all others. We call these weights w'. So all weights are now encoded with at most $\log_2(f(n)) + 1$ bits, i.e., are bounded by a polynomial function of the input size. Now let M' be a $0.8 - 1/poly(n)$-approximation for TkPM on the graph with weights w'. Observe that for any edge e, $|w(e) - w'(e) \cdot \frac{W}{f(n)}| \leq \frac{W}{f(n)}$ (the rounding error). Since a perfect matching contains $n/2$ edges we get

$$w^k(M') \geq w'^k(M') \cdot \frac{W}{f(n)} - \frac{nW}{2f(n)} \geq w'^k(M') \cdot \frac{W}{f(n)}(1 - \frac{n}{f(n)}).$$

The last inequality resulting from the fact that the optimal solution has weight at least $W \geq f(n)$ so $w'^k(M') \geq f(n)/2$. Now since M' is a $0.8 - 1/poly(n)$-approximation we get

$$w^k(M') \geq (0.8 - 1/poly(n)) \cdot w'^k(M^*) \cdot \frac{W}{f(n)}(1 - \frac{n}{f(n)}) \geq (0.8 - 2/poly(n)) \cdot w'^k(M^*) \cdot \frac{W}{f(n)}.$$

Going back to the original weights, we get

$$w^k(M') \geq (0.8 - 2/poly(n)) \cdot (w^k(M^*) - \frac{nW}{2f(n)}).$$

Finally, using $w^k(M^*) \geq W \geq f(n)$ we get

$$w^k(M') \geq (0.8 - 3/poly(n))w^k(M^*).$$

Approximation algorithm for polynomial weights. We start with a preprocessing of the edge weights. Given an instance of TkPM, let M^* be an optimal solution. Let e_k be the kth edge from M^* in the edge ordering. The algorithm starts by guessing e_k (i.e., running for all possibilities of e_k and outputting the matching of highest top-k value among all solutions) and setting the weights of all edges after e_k in the ordering to 0. This means that some solution M^* will contain at most k edges of strictly positive weight. Note that this does not modify the top-k weight of M^* and can only decrease the top-k weight of any matching, i.e., if we find a $0.8 - 1/poly(n)$-approximation on the new weights, it is also a $0.8 - 1/poly(n)$-approximation on the original weights. Also note that now we have $w^k(M^*) = w(M^*)$. For ease of notation, let $\epsilon = 1/poly(n)$ for the desired polynomial function in the approximation factor. The algorithm will have two phases. In the first phase, we use a slight modification of Algorithm 1 (we call it ModifiedTkpmApprox in Algorithm 2) where instead of returning M, it returns M_1, M_2 and t_2. We label them M_0', M_0 and t_0 respectively. This way we are guaranteed that M_0 has at most k edges with $w_{t_0}(e) > 0$ and M_0' has at least k edges with $w_{t_0-\epsilon/k}(e) > 0$, i.e., we have one matching with less then k edges of weight strictly more than t_0 and one with more than k edges of weight strictly more than $t_0 - \epsilon/k$. Algorithm 1 also guarantees that both matchings have maximum total weight with respect to their thresholds.

The second phase only works for bipartite graphs since it will rely on the algorithm of Proposition 11 (see ImproveUsingBoundedCycles in Algorithm 2) to increase the weight of the matching instead of recomputing the maximum weight perfect matching from scratch whenever we decrease the threshold (as is done in Algorithm 1). This way we again search for a threshold t_1 such that ImproveUsingBoundedCycles fails to get a PM containing more than k edges e with $w_{t_1}(e) > 0$ and instead stops at a perfect matching M_{t_1} containing at most k edges e with $w_{t_1}(e) > 0$, but for threshold $t_2 = t_1 - \epsilon/k$ ImproveUsingBoundedCycles outputs two perfect matchings M_1 and M_2 such that M_2 has at most k edges e with $w_{t_2}(e) > 0$ and M_1 has more than k edges e with $w_{t_2}(e) > 0$.

The full algorithm is given in Algorithm 2. Note that for a graph $G := (A \cup B, E, w)$ and a perfect matching M, we define the directed graph $G_M := (A \cup B, E, w')$ where every edges in M is oriented from A to B and has weight $w'(e) = -w(e)$ and every edge not in M is oriented from B to A and has weight $w'(e) = w(e)$. This means that for $M' := M \Delta C$ where C is a directed cycle (which implies that it is M-alternating), M' is a PM with $w(M') = w(M) + w'(C)$. The proof of correctness and running time of the algorithm can be found in the extended version of this paper [12]. ◀

4.4 Parametrized Complexity of TkPM

In this section we show that TkPM can be solved in Fixed Parameter Tractable (FPT) time parameterized by k and α, the independence number of the graph (i.e., the size of the largest independent set). The algorithm mainly uses Proposition 3. To prove its correctness we will rely on what [13] defines as skip, which allows us to shorten alternating cycles using the bound on the independence number. This way we manage to bound the total number of edges, in a symmetric difference with some optimal solution, by a function of k and α. We start by adapting the following definition and lemma from [13].

▶ **Definition 18** (adapted from [13]). *Let C be a an M-alternating cycle. A skip S is a set of 2 non-matching edges $e_1 := (v_1, v_2)$ and $e_2 := (v_1', v_2')$ with $e_1, e_2 \notin C$ and $v_1, v_1', v_2, v_2' \in C$ s.t. $C' = e_1 \cup e_2 \cup C \setminus (C[v_1, v_1'] \cup C[v_2, v_2'])$ is an M-alternating cycle and $|C| - |C'| > 0$.*

▶ **Lemma 19** (adapted from [13]). *Let P be an M-alternating path containing a set $\mathcal{P}$ of disjoint paths, each of length at least 3 starting and ending at non-matching edges, and $|\mathcal{P}| \geq 4^{\alpha}$. Then P contains a skip.*

This allows us to prove the following.

▶ **Proposition 20.** ($\star$) *Let M be a PM in a weighted graph G of independence number α, with edge colors red and blue. Let $\mathcal{C}$ be a set of disjoint M-alternating cycles in G. Let $R \subseteq E(\mathcal{C})$ be the set of red edges in $\mathcal{C}$ and $k = |R|$. Then G must contain a set of disjoint M-alternating cycles $\mathcal{C}'$ s.t. $R \subseteq \mathcal{C}'$, $|E(\mathcal{C})| \leq kf(\alpha)$ for $f(\alpha) = 4^{\alpha+1}$ and all edges in $E(\mathcal{C}')\backslash E(\mathcal{C})$ are non-matching edges.*

▶ **Lemma 21.** ($\star$) *Given an instance of TkPM and a PM M, there exists an optimal solution M' s.t. $|E(M\Delta M')| \leq kf(\alpha)$ for $f(\alpha) = 4^{\alpha+1}$, where α is the independence number of the input graph.*

Algorithm 2 TkPM 0.8-approximation Algorithm.

```
Input: An instance of TkPM and $\epsilon := 1/poly(n)$ for some polynomial function.
Output: PM $M$ with $w^k(M) \geq (0.8 - \epsilon)w(M^*)$ where $M^*$ is an optimal solution.
for $e_k \in E(G)$ do
    for $e \in E(G)$ do
        if $e > e_k$ then
            $w(e) \leftarrow 0$;
    $M_0', M_0, t_0 \leftarrow$ ModifiedTkpmApprox; $M_1, M_2 \leftarrow M_0$; $Success \leftarrow False$;
    while $Success == False$ and $t - \epsilon/k > 0$ do
        $t \leftarrow t - \epsilon/k$;
        $M_1, M_2, Success \leftarrow$ ImproveUsingBoundedCycles$(M_1, t)$;
        if $Success$ then
            $t_2 \leftarrow t$; $t_1 \leftarrow t + \epsilon/k$;
        else
            $M_0 \leftarrow M_1$;
    $M \leftarrow$ BestOf$(M, M_0, M_0', M_1, M_2)$;
return $M$;

Procedure ImproveUsingBoundedCycles$(M, t)$:
    if $M_1$ contains more than $k$ edges $e$ with $w_t > 0$ then
        return $M_1, M_1, True$;
    else
        $M_1, M_2 \leftarrow M$;
        while $M_1$ contains at most $k$ edges $e$ with $w_t(e) > 0$ and there exists a
          positive cycle in $G_{M_1}$, for threshold $t$, containing at most $k$ edges $e$ with
          $w_t(e) > 0$ and $e \notin M_1$ do
            Use Proposition 11 to find a cycle $C$ with the above properties;
            $M_2 \leftarrow M_1$; $M_1 \leftarrow M_2 \Delta C$;
        if $M_1$ contains less than $k$ edges $e$ with $w_t(e) > 0$ then
            return $M_1, M_1, False$;
        else
            return $M_1, M_2, True$;
```

Proof of Theorem 9. Given an instance of TkPM, we start by computing any perfect matching M. By Lemma 21, we know that there exists an optimal solution M' s.t. $|E(M \Delta M')| \leq kf(\alpha)$ for $f(\alpha) = 4^{\alpha+1}$. Let e_k be the kth edge from M' in the edge ordering. The algorithm starts by guessing e_k (i.e., running for all possibilities of e_k and outputting the matching of highest top-k value among all solutions) and setting the weights of all edges after e_k in the ordering to 0 and coloring them blue, while the rest of the edges are colored red. Note that this does not modify the top-k weight of M' and can only decrease the top-k weight of any other matching. Now we can use the algorithm of Proposition 3 to find a PM M'' with $w(M'') \geq w(M')$ and $|R(M'')| = |R(M')| = k$. Since only red edges have non-zero weight we get $w^k(M'') \geq w^k(M')$ so M'' is an optimal solution (since M' is an optimal solution). The running time is dominated by the algorithm of Proposition 3 which runs in time $L^{O(L)}$ where $L = |E(M \Delta M')| \leq k4^{\alpha+1}$. ◀

5 Conclusion and Open Problems

In the paper we study the Top-k perfect matching problem which is shown to be polynomially equivalent to the Exact Matching problem. In the course of developing approximation algorithms for this problem we also initiate a new direction of study for the EM problem where the goal is to minimize the constraint violation while requiring the output to be a perfect matching. We also continue the study of the parameterized complexity of EM that was initiated by [13]. To show the utility of these developments, we provide FPT algorithms for TkPM which rely on them.

Our work leaves open many questions, we list a few. Starting with questions related to EM, can we reduce the constraint violation in Theorem 1 to less than $0.5k$? The aim would be to get $o(k)$, but so far no constant improvement is known. Also, can we design an FPT algorithm for EM parameterized by k and α? This would be an intermediate step towards an FPT algorithm parameterized only by α and resolving the open problem in [13]. For TkPM, the first open question is whether the problem is **NP**-hard. We know it is unlikely for the case of polynomial sized input weights since we can reduce it to EM, but the exponential size weights case is still open. Another interesting problem is to get an FPT (or even XP) algorithm for TkPM parameterized only by α. Note that this is known for EM, but the reduction of TkPM to EM does not preserve the independence number of the graph. An FPT algorithm parameterized only by k, for either EM or TkPM would also be highly desirable. Finally an important step forward would be to improve the approximation algorithms for TkPM, with the goal of getting a polynomial time approximation scheme.

References

1 Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *Journal of the ACM (JACM)*, 42(4):844–856, 1995.

2 Vikraman Arvind, Johannes Köbler, Sebastian Kuhnert, and Jacobo Torán. Solving linear equations parameterized by hamming weight. *Algorithmica*, 75(2):322–338, 2016.

3 Richard Bellman. On a routing problem. *Quarterly of applied mathematics*, 16(1):87–90, 1958.

4 André Berger, Vincenzo Bonifaci, Fabrizio Grandoni, and Guido Schäfer. Budgeted matching and budgeted matroid intersection via the gasoline puzzle. *Mathematical Programming*, 128(1):355–372, 2011.

5 Jacek Błażewicz, Piotr Formanowicz, Marta Kasprzak, Petra Schuurman, and Gerhard J Woeginger. A polynomial time equivalence between DNA sequencing and the exact perfect matching problem. *Discrete Optimization*, 4(2):154–162, 2007.

6 Daniel P Bovet and Pierluigi Crescenzi. *Introduction to the Theory of Complexity*. Prentice Hall International (UK) Ltd., GBR, 1994.

7 Paolo M. Camerini, Giulia Galbiati, and Francesco Maffioli. Random pseudo-polynomial algorithms for exact matroid problems. *Journal of Algorithms*, 13(2):258–273, 1992.

8 Deeparnab Chakrabarty and Chaitanya Swamy. Approximation algorithms for minimum norm and ordered optimization problems. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, pages 126–137, 2019.

9 Radu Curticapean and Mingji Xia. Parameterizing the permanent: Hardness for K_8-minor-free graphs. *arXiv preprint*, 2021. `arXiv:2108.12879`.

10 Marek Cygan, Fedor V Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized algorithms*, volume 5. Springer, 2015.

11 Jack Edmonds. Maximum matching and a polyhedron with 0, 1-vertices. *Journal of research of the National Bureau of Standards B*, 69(125-130):55–56, 1965.

12 Nicolas El Maalouly. Exact matching: Algorithms and related problems. *arXiv preprint*, 2022. `arXiv:2203.13899`.

13 Nicolas El Maalouly and Raphael Steiner. Exact Matching in Graphs of Bounded Independence Number. In *47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022)*, volume 241 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 46:1–46:14, 2022.

14 Nicolas El Maalouly and Lasse Wulf. Exact matching and the top-k perfect matching problem. *arXiv preprint*, 2022. `arXiv:2209.09661`.

15 Dennis Fischer, Tim A Hartmann, Stefan Lendl, and Gerhard J Woeginger. An investigation of the recoverable robust assignment problem. *arXiv preprint*, 2020. `arXiv:2010.11456`.

16 Anna Galluccio and Martin Loebl. On the theory of Pfaffian orientations. I. Perfect matchings and permanents. *Electronic Journal of Combinatorics*, 6:R6, 1999.

17 Hans-Florian Geerdes and Jácint Szabó. A unified proof for karzanov's exact matching theorem. Technical Report QP-2011-02, Egerváry Research Group, Budapest, 2011. `egres.elte.hu`.

18 Christopher David Godsil. *Algebraic combinatorics*. Routledge, 2017.

19 Fabrizio Grandoni and Rico Zenklusen. Optimization with more than one budget. *arXiv preprint*, 2010. `arXiv:1002.2147`.

20 Rohit Gurjar, Arpita Korwar, Jochen Messner, Simon Straub, and Thomas Thierauf. Planarizing gadgets for perfect matching do not exist. In *International Symposium on Mathematical Foundations of Computer Science*, pages 478–490. Springer, 2012.

21 Rohit Gurjar, Arpita Korwar, Jochen Messner, and Thomas Thierauf. Exact perfect matching in complete graphs. *ACM Transactions on Computation Theory (TOCT)*, 9(2):1–20, 2017.

22 Valentine Kabanets and Russell Impagliazzo. Derandomizing polynomial identity tests means proving circuit lower bounds. *computational complexity*, 13(1-2):1–46, 2004.

23 AV Karzanov. Maximum matching of given weight in complete and complete bipartite graphs. *Cybernetics*, 23(1):8–13, 1987.

24 Monaldo Mastrolilli and Georgios Stamoulis. Constrained matching problems in bipartite graphs. In *International Symposium on Combinatorial Optimization*, pages 344–355. Springer, 2012.

25 Monaldo Mastrolilli and Georgios Stamoulis. Bi-criteria and approximation algorithms for restricted matchings. *Theoretical Computer Science*, 540:115–132, 2014.

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

27 Christos H Papadimitriou and Mihalis Yannakakis. The complexity of restricted spanning tree problems. *Journal of the ACM (JACM)*, 29(2):285–309, 1982.

28 Irena Rusu. Maximum weight edge-constrained matchings. *Discrete applied mathematics*, 156(5):662–672, 2008.

29 Georgios Stamoulis. Approximation algorithms for bounded color matchings via convex decompositions. In *International Symposium on Mathematical Foundations of Computer Science*, pages 625–636. Springer, 2014.

30 Ola Svensson and Jakub Tarnawski. The matching problem in general graphs is in quasi-nc. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 696–707. Ieee, 2017.

31 Tongnyoul Yi, Katta G Murty, and Cosimo Spera. Matchings in colored bipartite networks. *Discrete Applied Mathematics*, 121(1-3):261–277, 2002.

32 Raphael Yuster. Almost exact matchings. *Algorithmica*, 63(1):39–50, 2012.