The parameterized complexity of a new matching problem the Paired Matching problem

Verhaegh, Ruben F.A.

Award date:
2022

EINDHOVEN UNIVERSITY OF TECHNOLOGY

Department of Mathematics and Computer Science
Algorithms, Geometry & Applications

The parameterized complexity of a new matching problem: the Paired Matching problem

Master's Thesis

R.F.A. Verhaegh

5 July 2022

Supervision:	*Assessment committee:*
B.M.P. Jansen	B.M.P. Jansen
L. Sanità	L. Sanità
	A. Ravagnani
	N. Zannone

Credits: 45

This is a public Master's thesis.

This Master's thesis has been carried out in accordance with the rules of the TU/e Code of Scientific Conduct.

Abstract

We introduce a new matching problem originating from industry called the Paired Matching problem. The objective in the problem is to find a maximum matching of minimum cost in a bipartite graph. This is complicated by a non-trivial definition of cost, which is expressed based on a pairing of the vertices in one partite set. We prove that the problem is NP-complete, but also give some special cases of the problem which can be solved in polynomial time. We furthermore study the parameterized complexity of various parameterizations of the problem. These include parameterizations by the solution size and by different measures for the density or complexity of the graph. For most of these, we provide parameterized algorithms to show that these parameterizations are either in FPT or in XP. For a parameterization of particular practical interest, we give a randomized XP algorithm by generalizing a result known for the Exact Matching problem.

Contents

1 Introduction **1**

2 Preliminaries **4**
2.1 Matchings and matrices 4
2.2 Complexity theory 6

3 Classical complexity of the Paired Matching problem **9**
3.1 NP-completeness of the Paired Matching problem 9
3.2 The approximability of MPM 15
3.3 Restriction to balanced bipartite graphs 17

4 Parameterized complexity of PM: input parameters **19**
4.1 PM parameterized by $|T|$ 20

5 Parameterized complexity of PM: dense graphs **24**
5.1 PM on complete bipartite graphs 24
5.2 PM parameterized by the number of non-edges 26
5.3 PM parameterized by the number of picky tasks 28
5.3.1 Some preliminaries 28
5.3.2 An FPT algorithm 30
5.4 PM parameterized by the number of picky people 36
5.5 PM parameterized by a vertex cover of the non-edges 37

6 Parameterized complexity of PM: equivalence classes **40**
6.1 PM parameterized by the number of equivalence classes of people pairs . . 40
6.1.1 Randomized XP algorithm 50
6.2 PM parameterized by the number of equivalence classes of tasks 56

7 Conclusion and discussion **61**

1 Introduction

Background and motivation In a graph $G = (V, E)$, a matching is a subset $M \subseteq E$ of the edges, such that every vertex in V is incident with at most one edge from M. A matching is called *perfect* if it is of size $|V|/2$, meaning that every vertex is incident with exactly one edge in the matching. The field of matching theory studies and explores matchings and their properties, which are relevant for a plethora of applications. These include, but are definitely not limited to, applications in scheduling [3], neural networks [6], protein interactions [42], electrical networks [2] and pattern recognition [7].

In many of these applications, we are interested in the existence of matchings with certain properties. The exact properties that are being required for these matchings can vary greatly between applications. Common techniques to efficiently find matchings in a graph include flow-based algorithms [15], finding augmenting paths [10, 19] and the evaluation of polynomials related to matrices derived from the graph [39]. Some of the most fundamental matching problems, such as the Maximum Matching problem and the Minimum Weight Perfect Matching problem have long been known to be solvable in polynomial time [10], while many other matching problems however have been proven to be NP-complete, including the Exact Weight Perfect Matching problem [32], the Induced Matching problem [38] and the Rainbow Matching problem [22].

The Exact Matching problem (EM) is a problem that deserves to be highlighted in particular, as it is closely related to the matching problem studied in this thesis. In this problem, a simple graph in which each edge is colored either red or blue is given, along with a positive integer r. The objective is to determine whether the graph contains a perfect matching with exactly r red edges [32]. One of the interesting aspects of the problem is that a randomized algorithm is known to solve it in polynomial time [30], yet no deterministic polynomial time algorithm for it is known. EM is one of the few natural problems for which this is currently the case [11] and it is conjectured that problems that are solvable by a randomized polynomial time algorithm are also solvable by a deterministic polynomial time algorithm [1, 20, 31, 36].

In this thesis, we introduce and study a new matching problem. Its origins and relevance lie in pick-and-place machines that are used in electronics assembly to place electrical components on printed circuit boards. Recently, a company from the Eindhoven Brainport area designed an upgrade to one of their machines, such that it could pick and place two of these components at the same time. With this upgrade also comes a new optimization problem: which pairs of components should be handled simultaneously to minimize the time spent. This optimization problem can be formulated as the following problem.

MINIMUM PAIRED MATCHING (MPM)

Given: A bipartite graph $G = (P + T, E)$ with "people vertices" P and "task vertices" T, where the $|P| = 2n$ people vertices are partitioned into ordered pairs $(a_1, b_1), (a_2, b_2), \ldots, (a_n, b_n)$. Also given are a constant $c \in \mathbb{R}_{\geq 0}$, and a function $f : T \times T \to \mathbb{R}_{\geq 0}$.

Objective: Determine the minimum cost of a matching with cardinality $|T|$ in G. Each people pair (a_i, b_i) contributes separately to the cost of a matching M and the total cost of the matching is the sum of all these contributions. The contribution of a pair (a_i, b_i) is as follows:

- If neither a_i nor b_i is matched to a task by M, their contribution to the cost is 0.
- If exactly one of a_i and b_i is matched to a task by M, their contribution to the cost is c.
- If a_i is matched to task $t_1 \in T$ and b_i is matched to task $t_2 \in T$, then their contribution to the cost is given by $f(t_1, t_2)$. Note the possible asymmetry in the definition of f, meaning that $f(t_1, t_2)$ might not equal $f(t_2, t_1)$.

We define the PAIRED MATCHING problem (PM) to be the corresponding decision problem in which an additional parameter $d \in \mathbb{R}_{\geq 0}$ is added to the input. Rather than finding the minimum cost of a matching with cardinality $|T|$ in the graph, the objective is then to determine whether such a matching exists with cost at most d.

We study the algorithmic complexity of PM and MPM and we do this with a particular focus on parameterized complexity theory. This field of research can provide insight into what exactly makes certain problems hard and can give rise to efficient algorithms for restricted input classes. By associating some parameter k with each problem instance, the complexity of algorithms solving such a parameterized problem can be expressed in terms of both the input size n and the associated parameter k. Although some of the most relevant topics from this field are introduced in Section 2, we refer the unfamiliar reader to the book Parameterized Algorithms by Cygan et al. [9] for a more complete introduction and overview of the field.

Our main goal is to classify the algorithmic complexity of PM and to give guarantees on the worst-case running times for exact (parameterized) algorithms for the problem.

Our contribution One of the first results presented in this report is a proof of the NP-completeness of PM. Next, we provide parameterized algorithms for several parameterizations of the problem. One of these solves PM parameterized by $|T|$ and makes use of a kernelization technique presented by Bodlaender, Jansen, and Kratsch [4]. We present algorithms for multiple parameterizations relating to the density of the input graph. The main idea behind these algorithms is to separate the problem into a part that can be solved by a brute-force algorithm and a part that can be solved via a reduction to a polynomial time solvable matching problem.

A practically motivated parameterization of PM concerns a parameter related to a specific notion of complexity of the input graph. When partitioning the people vertices in an instance into sets of vertices with indistinguishable properties, the number of partite sets serves as a parameter for which inputs generally attain low values in practice. We obtain a randomized parameterized algorithm by generalizing a result from Mulmuley, Vazirani and Vazirani [30], which was used to describe a randomized algorithm for the aforementioned Exact Matching problem. For a closely related parameterization, we also present a parameterized algorithm, which is obtained by writing the problem as Integer Linear Program and using an algorithm by Lenstra [25] to solve it.

Related work Many matching problems have been studied from the perspective of parameterized complexity theory before. The Exact Matching problem is one of them and parameterized algorithms for it were developed in [12] and [11] when considering the independence number of the graph to be the parameter.

An other, more common parameterization to consider for graph problems is the size of the solution whose existence needs to be determined. Since the objective in EM is to determine the existence of a specific type of *perfect* matching, this size is always polynomial in the input size, making the solution size an uninteresting parameter to consider for EM. However, there exist many other matching problems which do not require a solution to be perfect and problems like the Rainbow Matching problem [17] and 3-Dimensional Matching problem [13] admit efficient parameterized algorithms when parameterized by the solution size. Contrarily, the Induced Matching problem is an example of a matching problem for which it has been proven that such algorithms are unlikely to exist [29], even when the input is restricted to bipartite graphs [28].

Organization The remainder of this report is organized as follows. In Section 2, some of the required background knowledge is presented and basic definitions and conventions are given. In Section 3, we study PM and MPM from the perspective of classical complexity theory and prove that PM is NP-complete. In the three sections afterwards, we present parameterized algorithms for various parameterizations of PM. We start by doing so for the parameterization of the problem by $|T|$ in Section 4. Next, we consider four parameterizations related to the graph density in Section 5. In Section 6, we consider two parameterizations related to the number of so-called equivalence classes of the input graph. We conclude the report in Section 7 and provide some open questions.

2 Preliminaries

For a positive integer n, we use $[n]$ to denote the set $\{1, 2, \ldots, n\}$. For a set S and an integer k, we use $\binom{S}{k}$ to denote the set of all subsets of S with size k. All graphs we consider are finite and undirected. In most cases, we furthermore assume graphs to be simple, meaning no self-loops or parallel edges are allowed. On a few occasions, we deviate from this convention by allowing parallel edges and we indicate this by calling graphs with parallel edges multi-graphs.

We denote a graph G with vertex set V and edge set E by $G = (V, E)$. Two vertices $u, v \in V$ are said to be *adjacent* when $\{u, v\} \in E$. For a vertex $v \in V$ the set of vertices adjacent to it in G is said to be *neighborhood* of v in G and we denote this set by $N_G(V)$. For a vertex $v \in V$ and an edge $e \in E$, we say that v and e are *incident* with one another when $v \in e$. For a subset $V' \subseteq V$, the subgraph of G *induced* by V' is the graph with vertex set V' and edge set $E \cap \binom{V'}{2}$. We denote it using $G[V']$. As defined in Section 1 as well, a *matching* $M \subseteq E$ in G is a set of edges such that every vertex in V is incident with at most one edge from M. M is said to be *perfect* when $|M| = |V|/2$, in which case every vertex in V is incident with exactly one edge from M. A *vertex cover* $X \subseteq V$ in G is a set of vertices such that every edge in E is incident with at least one vertex from X.

A *bipartite graph* is a graph $G = (V, E)$ in which the vertex set can be partitioned into two disjoint parts $V = A \cup B$ such that $E \subseteq A \times B$. We use the notation $G = (A+B, E)$ to denote a simple bipartite graph G with parts A and B. A bipartite graph $G = (A+B, E)$ is said to be *balanced* when $|A| = |B|$.

The remainder of the preliminaries are divided into two parts. In Section 2.1 we present a few known results on matchings, matrices and the relation between them and in Section 2.2 we introduce the most relevant (parameterized) complexity classes for this report.

2.1 Matchings and matrices

The preliminaries in this section are mostly relevant to Section 6.1. There, we find that some properties of matchings and matrices are relevant when developing algorithms to find matchings. One of them is the so-called isolating lemma. It makes a statement about set systems, which are defined as follows.

Definition 2.1 (Set system)**.** *A set system is a pair (S, F), where S is some finite set and F is a family of subsets of S.*

Let such a set system (S, F) be given and suppose a weight $w(x)$ is assigned to each element $x \in S$. Then we define the weight of a set $S' \subseteq S$ as $\sum_{x \in S'} w(x)$. The isolating lemma now states the following.

Lemma 2.2 (Isolating lemma [30]). *Let (S, F) be a set system where each element of S is assigned a random integer weight chosen uniformly and independently from $[1, 2 \cdot |S|]$ and with $F \neq \emptyset$. Then*

$$\mathbb{P}(\textit{There is a unique minimum weight set in } F) \geq \frac{1}{2}.$$

When taking S to be the set of edges of a graph and F to be (a subset of) the set of matchings in the graph, we allow ourselves to use the isolating lemma to make statements about matchings in the graph. When Mulmuley et al. first introduced the isolating lemma, this is exactly what they did, by using it in a randomized algorithm to check for the existence of perfect matchings [30]. This was achieved by combining the lemma with a variation of Tutte's Theorem. This theorem uses the notion of the *Tutte matrix* of a graph.

Definition 2.3 (Tutte matrix). *Let $G = (V, E)$ be a simple graph with vertices $V = \{v_1, v_2, \ldots, v_n\}$. The Tutte matrix A of G is an $n \times n$ matrix, where each entry $a_{i,j}$ is given by:*

$$a_{i,j} = \begin{cases} 0 & \text{if } \{v_i, v_j\} \notin E \\ x_{i,j} & \text{if } \{v_i, v_j\} \in E \text{ and } j > i \\ -x_{j,i} & \text{if } \{v_i, v_j\} \in E \text{ and } i > j \end{cases}$$

where the $x_{i,j}$ are indeterminates. As a result of this construction, the elements above the diagonal are positive and the ones below it are negative.

The Tutte matrix is an example of a *skew-symmetric* matrix. If we denote the elements of an $n \times n$ square matrix A by $a_{i,j}$, A is said to be skew-symmetric if $a_{i,j} = -a_{j,i}$ for all $i, j \in \{1, 2, \ldots, n\}$.

Tutte's theorem now states the following claim relating the Tutte matrix of a graph to the existence of a perfect matching in the graph.

Theorem 2.4 (Tutte's theorem [39]). *A graph has a perfect matching if and only if the determinant of its Tutte matrix is not identically 0.*

Although we do not obtain new results directly from Tutte matrices and their determinant, we do investigate matrices that are similar to Tutte matrices and their Pfaffian. The Pfaffian of a matrix is a function closely related to the determinant. To define it, we first introduce some notation related to permutations, as these are used in the definition of the Pfaffian.

The definition of the Pfaffian specifically uses the *sign* of a permutation, which is a function classifying every permutation as either an even or an odd permutation. Whether a permutation σ on a set X with a total ordering on it is considered even or odd depends on the number of inversions of the permutation, where an inversion is defined as a pair of elements $x, y \in X$ with $x < y$ and $\sigma(x) > \sigma(y)$. If the number of inversions of a permutation is even, the permutation is said to be even as well and if the number of

inversions is odd, the permutation itself is also said to be odd. The sign of a permutation σ is then simply defined as:

$$\operatorname{sign}(\sigma) = \begin{cases} +1 & \text{if } \sigma \text{ is even,} \\ -1 & \text{if } \sigma \text{ is odd.} \end{cases}$$

Next, we let Π_{2n} denote the set of all partitions of $[2n]$ into unordered pairs. Every element $\alpha \in \Pi_{2n}$ can be written in lexicographical order as

$$\{\{i_1, j_1\}, \{i_2, j_2\}, \ldots, \{i_n, j_n\}\}$$

such that $i_1 < i_2 < \ldots < i_n$ and $i_\ell < j_\ell$ for $\ell \in [n]$. Using this representation of α, we can define

$$\pi_\alpha = \begin{bmatrix} 1 & 2 & 3 & 4 & \cdots & 2n-1 & 2n \\ i_1 & j_1 & i_2 & j_2 & \cdots & i_n & j_n \end{bmatrix}$$

to be a permutation associated with α. Using this notation, we can define the Pfaffian for skew-symmetric matrices.

Definition 2.5 (Pfaffian)**.** *Let A be a skew-symmetric $2n \times 2n$ matrix with entries $a_{i,j}$. Let*

$$A_\alpha := \operatorname{sign}(\pi_\alpha) \cdot a_{i_1,j_1} \cdot a_{i_2,j_2} \cdot \ldots \cdot a_{i_n,j_n}$$

for all $\alpha \in \Pi_{2n}$. The Pfaffian of A is then defined as:

$$\operatorname{pf}(A) = \sum_{\alpha \in \Pi_{2n}} A_\alpha.$$

For $n \times n$ skew-symmetric matrices with n odd, the Pfaffian is defined to be 0.

There exist many other definitions for the Pfaffian of a square matrix [24], many of which also define the Pfaffian for square matrices that are not skew-symmetric. The definition provided above for skew-symmetric matrices is however often useful in the context of matchings. It is furthermore known that for skew-symmetric matrices the Pfaffian is the square root of the determinant [16].

2.2 Complexity theory

As briefly introduced in Section 1, the high-level idea of parameterized complexity theory is to associate a positive integer k with every problem instance. We call a decision problem in which this is done a *parameterized problem* and the parameter k associated with every problem instance serves to capture the complexity of the instance in some well-defined way. The complexity of algorithms solving a parameterized problem, which we refer to as *parameterized algorithms* can then be expressed as a function of the input size n and the associated parameter k. We briefly introduce the parameterized complexity classes and techniques that are most relevant to us in this report and refer

to the book Parameterized Algorithms by Cygan et al. [9] for a more complete overview of parameterized complexity theory.

One of the parameterized complexity classes we use is the class FPT, containing the problems we call *fixed-parameter tractable*. A parameterized problem is in FPT when it can be solved in $g(k) \cdot n^{\mathcal{O}(1)}$ time for some computable function $g : \mathbb{N} \to \mathbb{N}$. This notion of fixed-parameter tractability is useful, since an FPT algorithm for a problem can be efficient in practice when we only need to deal with inputs for which this associated parameter k is small. We often abuse the notation and use "FPT" both to denote the parameterized complexity class and as an abbreviation for "fixed-parameter tractable". We often say that a problem is FPT with respect to a certain parameter k, that a parameterized problem is FPT or that it can be solved in FPT time. We call parameterized algorithms achieving this time complexity FPT as well, or say that they run in FPT time. The same holds for the names of other (parameterized) complexity classes.

A popular technique to develop FPT algorithms is that of *kernelization*. The idea of kernelization is to reduce a problem instance to an equivalent one whose size depends on the original parameter k but not the total size n of the original input. Formally, kernelization is defined as follows.

Definition 2.6 (Kernelization, kernel [9])**.** *A kernelization algorithm, or a simply a kernel, for a parameterized problem Q is an algorithm that takes an instance (I, k) of Q as input and returns in polynomial time an equivalent instance (I', k') of Q with $|I'| + k' \leq g(k)$ for some computable function $g : \mathbb{N} \to \mathbb{N}$.*

We abuse the notation by calling the output of a kernelization algorithm a kernel as well. Since the size of this output depends only on the original parameter k and not on the size n of the entire original input, we can obtain an FPT algorithm by running a kernelization algorithm followed by a possibly exponential time algorithm on its output.

Another parameterized complexity class that will be used in this report is the class XP, containing the problems we call *slice-wise polynomial time solvable*. A problem is in XP when it can be solved in $\mathcal{O}(n^{g(k)})$ time for some computable function $g : \mathbb{N} \to \mathbb{N}$. We see that for instances with a fixed parameter value k, problems in either of the classes FPT or XP can be solved in time that is polynomial in the input size n. Remark however that the class XP is defined less strictly than the class FPT, as it allows for the degree of this polynomial to depend on k. We therefore find that FPT $\subseteq$ XP.

Besides deterministic algorithms, we also study randomized ones. There exist many definitions of randomized algorithms in literature, but throughout this report, we use the following definition.

Definition 2.7 (Randomized algorithm)**.** *A randomized algorithm for a decision problem is an algorithm that:*

- *For every NO-instance, always returns NO.*
- *For every YES-instance, returns YES with probability* $\geq \frac{1}{2}$ *and NO otherwise.*

A randomized algorithm never returns false positives as every NO-instance is correctly identified. So if a randomized algorithm outputs YES, we know for sure that this output is correct. It does have the possibility of returning false negatives, as every run of the algorithm on a YES-instance could incorrectly identify the instance as a NO-instance. The probability of this happening is by definition at most $\frac{1}{2}$. Moreover, when running the algorithm multiple times on the same YES-instance, each run independently keeps a probability of $\leq \frac{1}{2}$ of falsely returning NO. So if we were to run the algorithm m times on the same YES-instance, the probability that all these runs return NO is $\leq \frac{1}{2^m}$. If at least one of these m runs returns YES however, we can already conclude that the algorithm was given a YES-instance. This observation allows us to construct an algorithm that trades running time for reliability as every additional run halves the chance of falsely identifying YES-instances as NO-instances.

The running times of randomized algorithms can be categorized using the same classifications as used for deterministic algorithms, but because randomized algorithms do behave fundamentally differently from deterministic ones, we define randomized counterparts to complexity classes whose definitions are based on deterministic algorithms. We informally define RP to be the randomized complexity class containing all problems that admit a randomized algorithm as in Definition 2.7 that always runs in polynomial time. It is conjectured that P = RP [1, 20, 31, 36]. Similarly, we can informally define RXP to be the randomized parameterized complexity class containing all parameterized problem that admit a randomized algorithm as in Definition 2.7 that always runs in XP time.

3 Classical complexity of the Paired Matching problem

In this section, we classify the complexity of PM according to classical complexity theory. We start with a proof of NP-completeness in Section 3.1, after which we show some results on the approximability in Section 3.2. Finally, we show that PM is NP-complete even when restricted to balanced bipartite graphs in Section 3.3.

3.1 NP-completeness of the Paired Matching problem

To prove that PM is NP-complete, we provide a polynomial time reduction from 3OCC-SAT, a variant of the well-known SAT problem which asks to determine the satisfiability of a Boolean formula in conjunctive normal form. The 3OCC-SAT problem poses an extra restriction on the input by requiring that every variable occurs at most three times.

3-Occurence Satisfiability (3OCC-SAT)	
Given:	A Boolean formula F in conjunctive normal form (CNF) on the variables $X = \{x_1, x_2, \ldots, x_n\}$ such that every variable occurs at most three times.
Objective:	Determine whether X admits a truth assignment that satisfies F.

Although many variants of the SAT problem limit the clauses to contain no more than three literals, remark that clauses in 3OCC-SAT instances are allowed to be arbitrarily large. Now given the NP-completeness of the SAT problem [8], it is not hard to convince ourselves of the NP-completeness of 3OCC-SAT as every CNF formula can easily be rewritten to an equivalent one in which no variables occur more than three times [40]. Suppose that we are given a CNF formula in which some variable x occurs $k > 3$ times. Then we can replace the i-th occurrence of x with a new variable x_i for every occurrence $i = 1, \ldots, k$. To ensure that these newly introduced variables all receive the same truth assignment, we additionally add the following clauses to the formula: $(\neg x_1 \vee x_2) \wedge (\neg x_2 \vee x_3) \wedge \cdots \wedge (\neg x_{k-1} \vee x_k) \wedge (\neg x_k \vee x_1)$. This means that $x_1 \Rightarrow x_2 \Rightarrow x_k \Rightarrow x_1$, so the new formula can only be satisfied by setting $x_1 = x_2 = \ldots = x_k$, making it equivalent to the original formula. Doing this for all variables that occur more than three times results in an equivalent CNF formula of polynomial size in which each variable occurs at most three times. Hence, we can use this problem to prove Theorem 3.1.

Theorem 3.1. *Let $c_1, c_2 \in \mathbb{R}_{\geq 0}$ be any two constants such that $0 \leq c_1 < c_2$. Then any 3OCC-SAT instance can be reduced to an equivalent PM instance in which $c = c_1$, f only takes values $2 \cdot c_1$ and $2 \cdot c_2$ and $d = c_1 \cdot |T|$. Hence, PM is NP-complete, even when c and f are integer valued and bounded by a constant.*

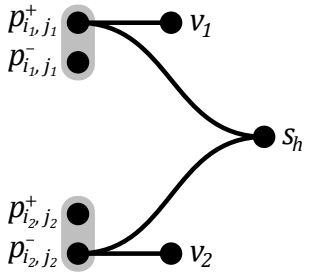
Proof. Let F be a 3OCC-SAT instance, where F is a formula with clauses $C_1, C_2, \ldots, C_m$ using the variables $X = \{x_1, x_2, \ldots, x_n\}$. Let $\ell_{i,1}, \ell_{i,2}, \ldots, \ell_{i,|C_i|}$ be the occurrences of literals of the i-th clause. We reduce F to an equivalent PM-instance (G, c, f, d) and we

start by constructing the graph $G = (P + T, E)$. See also Figure 2 for an example. The set T will be constructed as the union of two sets T_1 and T_2.

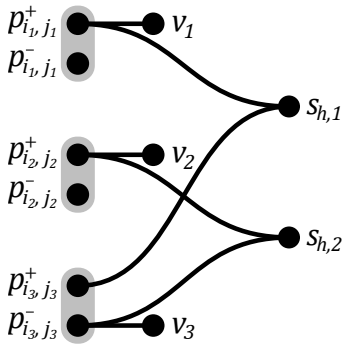
For every clause C_i we add a vertex v_i to T_1. For every occurrence $\ell_{i,j}$ of a literal we add two vertices $p^+_{i,j}$ and $p^-_{i,j}$ to P, representing a TRUE or FALSE assignment of the variable in $\ell_{i,j}$ respectively. These two vertices form a pair in P. If $\ell_{i,j}$ is a positive variable we connect $p^+_{i,j}$ and v_i. If $\ell_{i,j}$ is a negated variable we connect $p^-_{i,j}$ and v_i. Remark that this construction thus creates multiple people pairs for literals occurring multiple times.

By finding a matching in this graph which matches all vertices in T_1, we are determining an assignment of the variables in F which satisfies all its clauses. However, not every such matching in G necessarily corresponds to a valid truth assignment of the variables in X: we might try to set a variable to TRUE to satisfy one clause and at the same time set it to FALSE to satisfy another clause. Hence, we need to expand our graph to prevent situations like these. We can only encounter this problem for variables that occur both positively and negatively in the formula, so we expand our reduction based on the following two cases:

- Suppose x_h is a variable that occurs twice in F: once positively in the literal ℓ_{i_1,j_1} and once negated in the literal ℓ_{i_2,j_2}. We then add a vertex s_h to the set T_2 and connect $p^+_{i_1,j_1}$ and $p^-_{i_1,j_1}$ to s_h. See Figure 1a for an example.
- Suppose x_h is a variable that occurs three times in F and suppose that it occurs positively in the literals ℓ_{i_1,j_1} and ℓ_{i_2,j_2} and that it occurs negated in the literal ℓ_{i_3,j_3}. We then add two vertices $s_{h,1}$ and $s_{h,2}$ to T_2. We connect both $p^+_{i_1,j_1}$ and $p^+_{i_3,j_3}$ to $s_{h,1}$ and we connect both $p^+_{i_2,j_2}$ and $p^-_{i_3,j_3}$ to $s_{h,2}$. See Figure 1b for an example. For variables that occur once positively and twice negated, we can simply swap the $+$ and $-$ signs in this step.



(a) The direct neighborhood of the vertices representing the literals that x_h occurs in if it occurs once positively and once negatively.



(b) The direct neighborhood of the vertices representing the literals that x_h occurs in if it occurs twice positively and once negatively.

Figure 1: The structure created for variables that occur two or three times in F.

For variables that occur only in positive form or only in negated form, we skip this step as there is only one assignment for such a variable that allows it to satisfy clauses. The problematic situation explained above, where it is simultaneously assigned TRUE to satisfy one clause and FALSE to satisfy another, is therefore not applicable to these variables.

By taking $T = T_1 \cup T_2$ we have now finalized the construction of $G = (P + T, E)$. Now let c_1 and c_2 be any two given constants with $0 \leq c_1 < c_2$. We let c and f depend on these two constants by taking $c = c_1$ and taking f to be given by:

$$f(t_1, t_2) = \begin{cases} 2c_1 & \text{if } t_1, t_2 \in T_2 \\ 2c_2 & \text{otherwise} \end{cases}$$

Finally, we take $d = c_1 \cdot |T|$, to complete our construction of the PM instance (G, c, f, d).

Figure 2 shows the resulting graph for the example formula

$$(x_1 \vee \neg x_2 \vee x_3 \vee x_4) \wedge (\neg x_1 \vee \neg x_2) \wedge (x_1 \vee x_2 \vee \neg x_3).$$

For better interpretability of the image, vertices and edges related to occurrences of the same variable are drawn in the same color.

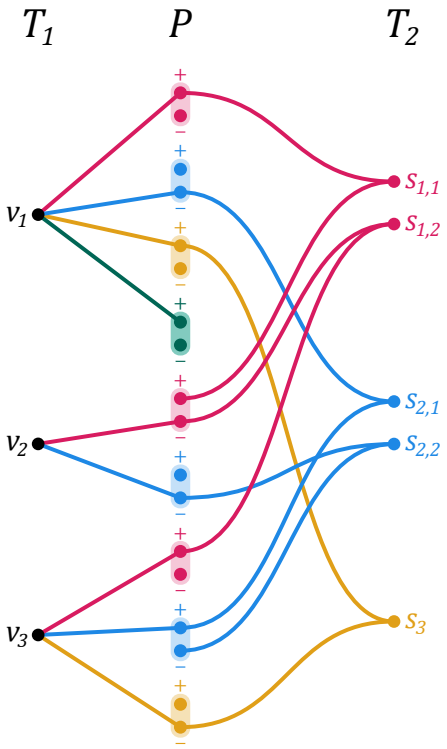


Figure 2: Example reduction from the formula $(x_1 \vee \neg x_2 \vee x_3 \vee x_4) \wedge (\neg x_1 \vee \neg x_2) \wedge (x_1 \vee x_2 \vee \neg x_3)$.

We continue by showing that the reduction is correct, followed by an explanation of how the reduction can be done in polynomial time.

Correctness. To prove that the reduction is correct, we show that F is a YES-instance if and only if (G, c, f, d) is a YES-instance.

($\Rightarrow$)
Suppose that F is a YES-instance for 3OCC-SAT. Then there is a truth assignment $\mathcal{T} : X \to \{\text{TRUE}, \text{FALSE}\}$ of the variables in X that satisfies F. In particular, it satisfies each clause individually. We will use this assignment to construct a matching M that covers every vertex in T and which has a cost of d. We construct it as the union of two matchings M_1 and M_2. M_1 is used to cover the vertices in T_1 and can be seen as actually encoding the truth assignment $\mathcal{T}$ into G. M_2 is used to cover the remaining vertices at a low enough cost. First we construct M_1.

Consider the clause C_i and consider a literal $\ell_{i,j}$ that satisfies it under assignment $\mathcal{T}$. If there are multiple literals satisfying the clause, we can pick any arbitrary one of them. If $\ell_{i,j}$ is a positive variable, we include $\{p^+_{i,j}, v_i\}$ in M_1. If $\ell_{i,j}$ is a negated variable, we include $\{p^-_{i,j}, v_i\}$ in M_1. By construction, the edge that we add exists in G. Doing this for every clause C_i ensures that every vertex in T_1 is covered by M_1. Furthermore, the vertices in T_1 do not have any shared neighbors so none of the edges of M_1 coincide, meaning it is indeed a matching.

We continue by constructing a matching M_2 which covers the vertices in T_2 and none of the vertices already covered by M_1. Consider a variable x_h that occurs more than once in F, both positively and negated. We distinguish two cases:

- Suppose x_h occurs twice in F: once positively in ℓ_{i_1,j_1} and once negatively in ℓ_{i_2,j_2}. If x_h is set to TRUE in $\mathcal{T}$, we know that ℓ_{i_2,j_2} cannot be used to satisfy C_{i_2}, which in turn means that we did not cover $p^-_{i_2,j_2}$ with M_1. Hence, we can add $\{p^-_{i_2,j_2}, s_h\}$ to M_2. By construction, this edge exists. Similarly, if x_h is instead set to FALSE in $\mathcal{T}$, we add $\{p^+_{i_1,j_1}, s_h\}$ to M_2. Again, this edge exists and does not coincide with an edge from M_1.

- Suppose x_h occurs three times in F. Assume for now that it occurs twice positively in ℓ_{i_1,j_1} and ℓ_{i_2,j_2} and once negated in ℓ_{i_3,j_3}. If x_h is set to TRUE in $\mathcal{T}$, we know that ℓ_{i_3,j_3} cannot be used to satisfy C_{i_3}, which in turn means that we did not cover $p^-_{i_3,j_3}$ with M_1. Because $p^+_{i_3,j_3}$ does not have any neighbors in T_1, this vertex was also not covered by M_1. Hence we can add the edges $\{p^+_{i_3,j_3}, s_{h,1}\}$ and $\{p^-_{i_3,j_3}, s_{h,2}\}$ to M_2. By construction, these edges exist.

  If x_h is instead set to FALSE in $\mathcal{T}$, we know that ℓ_{i_1,j_1} cannot be used to satisfy C_{i_1} and ℓ_{i_2,j_2} cannot be used to satisfy C_{i_2}. This in turn means that neither $p^+_{i_1,j_1}$ nor $p^+_{i_2,j_2}$ was covered by M_1. Hence, we can add the edges $\{p^+_{i_1,j_1}, s_{h,1}\}$ and $\{p^+_{i_2,j_2}, s_{h,2}\}$ to M_2. By construction these edges exist.

If x_h were to occur once positively and twice negated in F, we can simply swap the $+$ and $-$ signs in this step.

If we do this for every variable that occurs both positively and negated, we ensure that every vertex in T_2 is covered by M_2. Moreover, we have done so without covering vertices already covered by M_1. Combining this with the fact that M_1 and M_2 are both matchings, we get that $M := M_1 \cup M_2$ is also a matching. M then covers all vertices in $T = T_1 \cup T_2$.

Finally, we determine the cost of M. Note that most pairs in P have only one edge matched by M, therefore each contributing $c = c_1$ to the total cost of M. In our construction, the only pairs in P of which both vertices are matched by M correspond to variables that occur three times. In fact, every such variable has only one literal $\ell_{i,j}$ whose corresponding vertex pair even has edges connected to both vertices at all. There are only two cases in which both these vertices are matched by M: if $\ell_{i,j}$ is a positive variable which is set to FALSE in the satisfying assignment $\mathcal{T}$ or if $\ell_{i,j}$ is a negated variable which is set to TRUE in $\mathcal{T}$. In these cases, the corresponding vertices $p^+_{i,j}$ and $p^-_{i,j}$ are both matched to a vertex in T_2, meaning that together they contribute $2c_1$ to the cost of M. So for every edge in M it can be said that it contributes c_1 to the cost of M, meaning that its cost is $c_1 \cdot |T| = d$. Hence, (G, c, f, d) is a YES-instance.

($\Leftarrow$)
Suppose that (G, c, f, d) is a YES-instance. Then there exists a matching M in $G = (P + T, E)$ which has cost at most $d = c_1 \cdot |T|$ and covers all vertices in T. We partition the matching into $M = M_1 \cup M_2$ such that M_1 contains all the edges covering T_1 and M_2 contains all the edges covering T_2. We show that there exists an assignment $\mathcal{T}$ of the variables $x_1, \ldots, x_n$ such that F is satisfied.

For every edge $\{p^+_{i,j}, v_i\} \in M$ we assign the variable in the literal $\ell_{i,j}$ to TRUE and for every edge $\{p^-_{i,j}, v_i\} \in M$ we assign the variable in the literal $\ell_{i,j}$ to FALSE. Any possible remaining variables may receive an arbitrary assignment. Because M matches every vertex in T and in particular in T_1, every clause of F is satisfied by this assignment: if $\{p^+_{i,j}, v_i\} \in M$ (and therefore in G), then by construction C_i contains the positive literal $\ell_{i,j}$ meaning that it can be satisfied by setting the corresponding variable to TRUE. Likewise, if $\{p^-_{i,j}, v_i\} \in M$ (and therefore in G), then by construction C_i contains the negated literal $\ell_{i,j}$ meaning that it can be satisfied by setting the corresponding variable to FALSE. So this assignment indeed satisfies F. It remains to show that this is a valid assignment, i.e.: there is no variable which is set to TRUE and FALSE to satisfy multiple clauses.

This problem could of course only happen for a variable x_h which occurs at least once positively and at least once negated. We distinguish two cases:

- Suppose x_h occurs twice in F: once positively in ℓ_{i_1,j_1} and once negated in ℓ_{i_2,j_2}. This implies the presence of a vertex $s_h \in T$ which only has the vertices $p^+_{i_1,j_1}$ and $p^-_{i_2,j_2}$ as neighbors. Since s_h is by definition covered by M, we know that at least one of these two neighbors must be connected to x_h in M and because M is a matching, this particular vertex is not also used to satisfy its corresponding clause. Hence, at most one of the literals ℓ_{i_1,j_1} and ℓ_{i_2,j_2} is used to satisfy its corresponding clause meaning that x_h does not get assigned both TRUE and FALSE.

- Suppose x_h occurs three times in F. Assume for now that it occurs twice positively in ℓ_{i_1,j_1} and ℓ_{i_2,j_2} and once negated in ℓ_{i_3,j_3}. We will prove that if $\{p^-_{i_3,j_3}, v_{i_3}\} \in M$ (in which case we assign x_h to FALSE to satisfy clause C_{i_3}) it holds that $\{p^+_{i_1,j_1}, v_{i_1}\}, \{p^+_{i_2,j_2}, v_{i_2}\} \notin M$. This implies that x_h is not set to TRUE and FALSE simultaneously to satisfy multiple clauses.

  So suppose that $\{p^-_{i_3,j_3}, v_{i_3}\} \in M$. Because $s_{h,2}$ is by definition matched in M, but apparently not to $p^-_{i_3,j_3}$, it must instead be matched to its only other neighbor: $p^+_{i_2,j_2}$. Because $p^+_{i_2,j_2}$ is then already matched to a vertex other than v_{i_2} by M and M is a matching, we know that $\{p^+_{i_2,j_2}, v_{i_2}\} \notin M$.

  To show that also $\{p^+_{i_1,j_1}, v_{i_1}\} \notin M$, we first argue that $\{p^+_{i_3,j_3}, s_{h,1}\} \notin M$. If this edge were to be in M, then the pair $(p^+_{i_3,j_3}, p^-_{i_3,j_3})$ would contribute $2c_2$ to the cost of M by the definition of f. Since M has size $|T|$ and a cost of at most $c_1 \cdot |T|$, pairs in P cannot contribute more than c_1 to the cost per matched vertex in them without exceeding the cost of d. Hence, $\{p^+_{i_3,j_3}, s_{h,1}\}$ cannot be in M as it would make the pair $(p^+_{i_3,j_3}, p^-_{i_3,j_3})$ contribute $c_2 > c_1$ to the cost of M per matched vertex. So we establish that $\{p^+_{i_3,j_3}, s_{h,1}\} \notin M$. This implies that $\{p^+_{i_1,j_1}, s_{h,1}\} \in M$, because $s_{h,1}$ must be matched by M and $p^+_{i_1,j_1}$ is its only other neighbor, which in turn implies that $\{p^+_{i_1,j_1}, v_{i_1}\} \notin M$ because M is a matching.

  This shows that x_h is not set to TRUE and FALSE simultaneously in our assignment. An analogous argument can be made in case x_h occurs once positively and twice negated in F.

This proves that our assignment is indeed valid and satisfies F, meaning that F is a YES-instance for 3OCC-SAT.

Reduction is polynomial. For every occurrence of a literal we create two vertices in P and for every clause we create one vertex in T. This can be done in polynomial time. For each literal occurrence, we then connect one of the two corresponding vertices to the vertex corresponding to the clause it occurs in, which can again be done in polynomial time. In the second step of the reduction, we add for every variable that occurs twice another vertex to T together with two edges. Also, two vertices are added to T together with four more edges for every variable that occurs three times. Again, this can be done in polynomial time. Since c and f only take constant values, they can of course also be defined in polynomial time. Once the construction of the graph G is complete, the value

$d = c_1 \cdot |T|$ can also be computed in polynomial time. This concludes the reduction, so it can be computed in polynomial time and has polynomial size.

We have now shown that the reduction is both correct and polynomial in size and time, which in turn proves the first part of the theorem and shows that PM is NP-hard. To show that that PM is also NP-complete, we observe that the problem is in NP. For every matching in a PM instance, its cost can be computed in polynomial time. Since by definition every YES-instance has a matching whose cost is at most the given budget, this matching then serves as a polynomial time verifiable certificate that an instance is indeed a YES-instance. □

3.2 The approximability of MPM

An even stronger result follows almost directly from Theorem 3.1.

Corollary 3.2. *Unless P = NP, MPM cannot be approximated in polynomial time within a multiplicative factor of the optimal solution.*

Proof. By taking $c_1 = 0$ in Theorem 3.1, we obtain a restriction to the input of PM in which we always have $d = 0$. An algorithm that approximates MPM within some multiplicative factor of the optimal solution would return 0 if and only if the optimal solution is 0, so this approximation algorithm could also be used to solve any PM instance under this restriction. Since Theorem 3.1 states that this restricted version of PM is still NP-complete, such an approximation algorithm cannot run in polynomial time unless P = NP. □

Of course, any optimization problem in which the optimal solution is 0 can be solved exactly using an approximation algorithm. Approximating such problems is therefore just as hard as solving them exactly. However, even if we require c and f to be positive in MPM, thereby avoiding the trivial case from above where $d = 0$, we find the problem to be hard to approximate.

Proposition 3.3. *Unless P = NP, there is no polynomially bounded function $\alpha : \mathbb{N} \to \mathbb{N} \setminus \{0\}$ for which MPM has a polynomial time $\alpha(n)$-approximation scheme, not even when c and f are required to be both strictly positive and either integer valued or bounded by a constant.*

Proof. Recall that the NP-hardness of PM was proven in Theorem 3.1 by showing that, for any fixed c_1 and c_2 such that $0 \leq c_1 < c_2$, every 3OCC-SAT instance can be reduced to an equivalent PM instance in which $c = c_1$, f only takes values $2c_1$ and $2c_2$ and $d = c_1 \cdot |T|$. We show how for any $\alpha(n) > 1$, a choice can be made for the values of c_1 and c_2 such that they are strictly positive and such that this reduction always leads to an instance which can be solved exactly using an $\alpha(n)$-approximation for MPM. Moreover, this choice is such that c_1 and c_2 are polynomial in the number of vertices in the PM instance, and thereby polynomial in the input size of the 3OCC-SAT instance

it was obtained from, if $\alpha(n)$ is as well. It follows then that every 3OCC-SAT instance can be reduced to a PM instance of polynomial size that can be solved exactly using an $\alpha(n)$-approximation. Since 3OCC-SAT is NP-hard, no algorithm achieving such an $\alpha(n)$-approximation can run in polynomial time unless P = NP.

Before fixing a choice for c_1 and c_2, consider any PM instance $(G = (P + T), c, f)$ of the form described in Theorem 3.1 for some $0 \leq c_1 < c_2$. The minimum possible cost of a matching with cardinality $|T|$ is then $c_1 \cdot |T|$. Furthermore, if a matching of size $|T|$ has a greater cost than this, its cost must be at least $c_1 \cdot (|T| - 2) + 2 \cdot c_2$.

From this observation, we can see that an $\alpha(n)$-approximation algorithm for MPM can be used to determine whether or not G contains a matching of size $|T|$ with cost $c_1 \cdot |T|$ if $\alpha(n) < \frac{c_1 \cdot (|T|-2)+2 \cdot c_2}{c_1 \cdot |T|}$. We continue by showing how to choose c_1 and c_2 for any given $\alpha(n)$ such that this inequality holds. By rewriting the inequality, we find that c_1 and c_2 are needed such that:

$$\begin{aligned}
\alpha(n) &< \frac{c_1 \cdot |T| + 2 \cdot (c_2 - c_1)}{c_1 \cdot |T|} \\
\alpha(n) &< 1 + \frac{2 \cdot (c_2 - c_1)}{c_1 \cdot |T|} \\
\alpha(n) &< 1 + \frac{2}{|T|} \cdot \frac{c_2 - c_1}{c_1} \\
\frac{|T|}{2} \cdot (\alpha(n) - 1) &< \frac{c_2 - c_1}{c_1} \\
\frac{|T|}{2} \cdot (\alpha(n) - 1) + 1 &< \frac{c_2}{c_1}
\end{aligned}$$

The statement we want to prove now poses two different sets of restrictions on c_1 and c_2 under each of which this inequality must hold. The first one being that they are both positive and integer valued. Then the inequality can be satisfied by choosing $c_1 = 1$ and c_2 to be some integer larger than $\frac{|T|}{2} \cdot (\alpha(n) - 1) + 1$. Note that such c_1 and c_2 can be chosen to be polynomial in the number of vertices when $\alpha(n)$ is as well.

The second set of restrictions states that c_1 and c_2 must be both positive and bounded from above by some constant. Let B be such a constant. Then the inequality can be satisfied by choosing $c_2 = B/2$ and c_1 to be smaller than $c_2 / \left(\frac{|T|}{2} \cdot (\alpha(n) - 1) + 1\right)$. Again, when $\alpha(n)$ is polynomial in the number of vertices, such c_1 and c_2 can be chosen such that they can be stored in space polynomial in the number of vertices. □

We can also see that Proposition 3.3 does not hold when c and f are required to be strictly positive, integer valued *and* bounded by a constant B. In this case, the first two constraints require c and f to be at least 1, while the third constraint requires them to be at most B, restricting c and f to take integer values in the interval $[1, B]$. To see

that MPM can be approximated in this case, we turn to the more general case where c and f are required to be at least $L > 0$ and at most U, restricting c and f to take (not necessarily integer) values in the interval $[L, U]$. Then every matching of size $|T|$ must have a cost of at least $L \cdot |T|/2$ and at most $U \cdot |T|$. The former is attained by matchings in which each people pair that is covered has both its vertices matched to a task vertex at a cost of L. The latter is attained by matchings in which each people pair that is covered has only one vertex matched to a task vertex at a cost of U. It follows then that every matching of size $|T|$ serves as a $2\frac{U}{L}$-approximation.

3.3 Restriction to balanced bipartite graphs

Throughout the report, it will often be practical to work with PM instances in which the graph $G = (P + T, E)$ is a balanced bipartite graph. That is, instances with $|P| = |T|$. Under this restriction, any matching in G covering $|T|$ must be a perfect matching and hence the value of c no longer plays a role in determining the cost of such a matching. Since the restriction to balanced bipartite graphs changes the input requirements, we define the Balanced Paired Matching problem (BPM) as a variant of PM and we continue by showing that BPM is also NP-complete.

Balanced Paired Matching (BPM)	
Given:	A balanced bipartite graph $G = (P + T, E)$, where the $2n$ vertices in P are partitioned into ordered pairs $(a_1, b_1), (a_2, b_2), \ldots, (a_n, b_n)$. Also given are a cost function $f : T \times T \to \mathbb{R}_{\geq 0}$ and a budget $d \in \mathbb{R}_{\geq 0}$.
Objective:	Determine whether a perfect matching M exists in G with cost at most d. If we let $T_M(p)$ denote the vertex in T that is matched to $p \in P$ by M, then the cost of M is given by $\sum_{i=1}^{n} f(T_M(a_i), T_M(b_i))$.

Lemma 3.4. *Every PM instance (G, c, f, d) can be reduced to an equivalent BPM instance (G', f, d) in polynomial time, where G' is obtained from G by only adding task vertices which are connected to all people vertices. Hence, BPM is NP-complete.*

Proof. Let (G, c, f, d) be a PM instance with $G = (P + T, E)$. If $|P| < |T|$, then G is trivially a NO instance, so we assume w.l.o.g. that $|P| \geq |T|$. We continue by showing how to reduce the instance to an equivalent BPM instance (G', f', d), where $G' = (P + T', E')$ is a balanced bipartite graph. See also Figure 3 for

This is done by first introducing a new set of vertices C with $|C| = |P| - |T|$. We use $T' = T \cup C$ and connect every vertex in C to every vertex in P. Figure 3 shows an example. Finally, we use

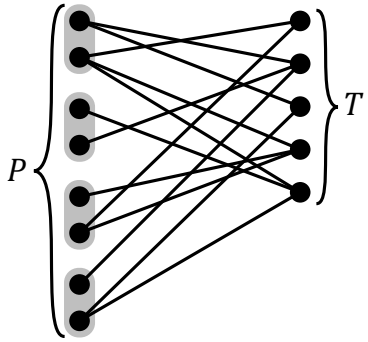
$$f'(t_1, t_2) = \begin{cases} f(t_1, t_2) & \text{if } t_1, t_2 \in T \\ c & \text{if } |\{t_1, t_2\} \cap C| = 1 \\ 0 & \text{if } t_1, t_2 \in C. \end{cases}$$

This concludes the construction of (G', f', d). Since $|T'| = |T| + (|P| - |T|) = |P|$, it holds that G' is a balanced bipartite graph. Moreover, the only modification to the graph G

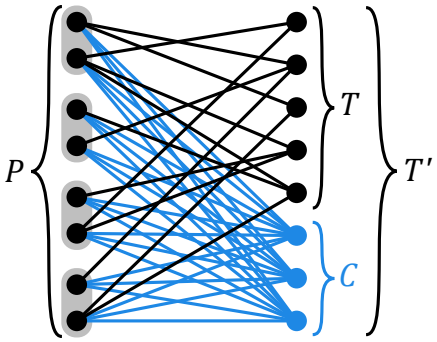
is the addition of the set of task vertices C, all of which are connected to all vertices in P.

We can also see that the obtained BPM instance is equivalent to the original PM instance. If we let M denote a solution to (G, c, f, d), then we can augment it without affecting its cost by adding edges to arbitrarily match each unmatched people vertex to a task vertex from C and obtain a solution M' to (G', f', d). Likewise, given a solution M' to (G', f', d), we obtain a solution M to (G, c, f, d) by removing all edges from M' that cover a vertex from C.

Since the reduction can be performed in polynomial time, it follows that BPM is NP-complete.



(a) An unbalanced example graph G.



(b) The balanced bipartite graph G' obtained from G using the reduction from Lemma 3.4. The newly added vertices and edges are marked blue, while all other elements were already in G.

Figure 3: Example of reducing a PM instance to a BPM instance.

□

4 Parameterized complexity of PM: input parameters

The remainder of the report is dedicated to the classification of the parameterized complexity of PM for various parameterizations of the problem. What a suitable parameterization of a problem is, varies by problem and application. If applicable, a natural parameter to consider is often the size of a solution that needs to be found in an instance. An example where such a parameterization is useful is the well known Vertex Cover problem. The problem of determining whether a vertex cover of size k exists in a graph, is FPT with respect to k [9].

In the Paired Matching problem, given an instance $(G = (P + T, E), c, f, d)$, we need to determine whether a matching of the size $|T|$ exists of cost at most d. A valid solution is therefore always one of size $|T|$, so this could be an interesting parameter to consider for PM. Indeed, we find the problem to be FPT when parameterized by $|T|$, which is shown in Section 4.1.

Other parameterizations that seem natural to consider at first are the sizes or values of the other parameters that are required for a PM instance: P, c, f and d. First, we remark that any non-trivial instance has $|P| \geq |T|$. After all, if we were to have an instance with $|P| < |T|$, a matching of size $|T|$ cannot exist, making every such instance trivially a NO-instance. $|P|$ can therefore not be small with respect to the size of the input graph, meaning that it is not a suitable parameter to consider for the parameterized complexity of PM. Since Theorem 3.1 states that PM is NP-complete, even when c and f are bounded by a constant, bounds on these two parameters also do not yield useful parameterizations of the problem.

The last parameter remaining from the list of default input parameters to PM is d. Using the value of d to parameterize PM by, would be an unusual choice for a parameter, as problems are usually parameterized by integer valued parameters, which d does not have to be. Regardless, we can make the following observation.

Observation 4.1. *PM cannot be FPT with respect to d unless $P = NP$, even when d is restricted to be integer.*

Proof. Suppose for the sake of contradiction that there exists an algorithm $\mathcal{A}$ that solves any PM instance (G, c, f, d) in $g(d) \cdot |(G, c, f, d)|^{\mathcal{O}(1)}$ time. Then we could also reduce any instance (G, c, f, d) to the equivalent instance $(G, \frac{c}{d}, \frac{f}{d}, 1)$ and use $\mathcal{A}$ to solve it in $g(1) \cdot |(G, \frac{c}{d}, \frac{f}{d}, 1)|$ time, which is polynomial in the original input size. Since we have already shown PM to be NP-complete, this would imply that P = NP. □

Note that the argument above does not hold when c and f are also required to be integer. The argument works via a reduction that divides c and f by d, resulting in values that are not necessarily integer. In fact, when c and f are not only required to be integer, but also to be strictly positive, any matching of size $|T|$ must have a cost of at least $|T|/2$, so any non-trivial instance under these restrictions must therefore have $d \geq |T|/2$. Since

we show PM to be FPT w.r.t. $|T|$ in Section 4.1, it is also FPT w.r.t. d when c and f are required to be positive integers.

4.1 PM parameterized by $|T|$

As mentioned above, PM is FPT when parameterized by $|T|$. To prove this, we show that PM admits a kernel of cubic size. The kernelization is based on the observation that it does not matter to which people pair a task is assigned, it only matters which other task vertex gets connected to the same pair, if any. This means that we could remove all pairs for which it holds that for any solution covering that pair, there also exists another solution of the same cost which does not cover that pair. A key ingredient to prove the correctness of the kernelization will be a formalization of this idea as given by the following result from Bodlaender, Jansen, and Kratsch [4] (Theorem 2).

Lemma 4.2. *Let $B = (X + Y, E)$ be a bipartite graph and let $M \subseteq E$ be a maximum matching in B. Let $X_M \subseteq X$ be the set of vertices in X that are covered by M. Then for each $Y' \subseteq Y$, if there exists a matching M' in B that covers Y', then there exists a matching M'' in $G[X_M \cup Y]$ that covers Y'.*

We continue by showing the following result:

Proposition 4.3. *PM admits a kernel with $\mathcal{O}(|T|^2)$ vertices and $\mathcal{O}(|T|^3)$ edges.*

Proof. Let $(G = (P + T, E), c, f, d)$ be a PM instance. We will construct a graph $G' = (P' + T, E')$ with $\mathcal{O}(|T|^2)$ vertices and $\mathcal{O}(|T|^3)$ edges such that (G, c, f, d) is a YES-instance if and only if (G', c, f, d) is a YES-instance. Figure 4 shows the kernelization for an example input.

Before constructing G', we first construct an auxiliary bipartite graph $B = (X + Y, E'')$ from G. For every pair (a_i, b_i) in P we add a vertex x_i to X. We construct Y as the union of two sets $Y = Y_1 \cup Y_2$. For every vertex $t_j \in T$ we add a vertex y_j to Y_1 and for every pair of distinct vertices $t_{j_1}, t_{j_2} \in T$ we add the vertices y_{j_1,j_2} and y_{j_2,j_1} to Y_2. We add an edge between every $x_i \in X$ and $y_j \in Y_1$ for which either $\{a_i, t_j\} \in E$ or $\{b_i, t_j\} \in E$. We also add an edge between every $x_i \in X$ and $y_{j_1,j_2} \in Y_2$ for which $\{a_i, t_{j_1}\}, \{b_i, t_{j_2}\} \in E$. This concludes the construction of B.

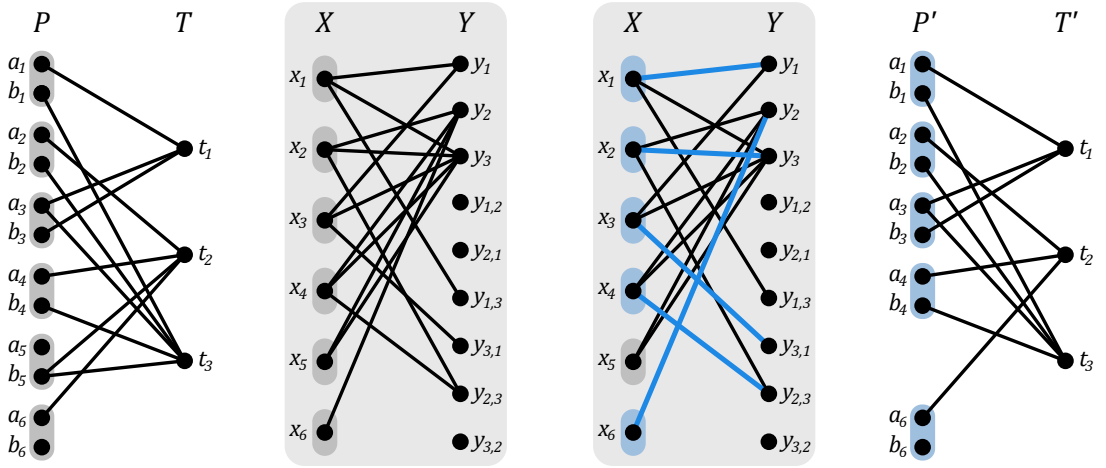
An interesting observation to make is that every matching M_G in G can be encoded in B as a matching M_B. Given any M_G, we could construct a corresponding M_B in which:

- The presence of an edge between some $x_i \in X$ and some $y_j \in Y_1$ would indicate that M_G leaves one vertex of (a_i, b_i) unmatched, while matching the other to $t_j \in T$.
- The presence of an edge between some $x_i \in X$ and some $y_{j_1,j_2} \in Y_2$ in M_B would indicate that a_i and b_i get matched to t_{j_1} and t_{j_2} respectively by M_G.

Now that we have constructed our auxiliary graph B, we continue by finding a maximum matching M in it and we use this matching to construct the graph $G' = (P' + T', E')$.

To this end, let $X_M \subseteq X$ denote the set of vertices in X that are covered by M. Then G' is obtained from G by removing all people pairs (a_i, b_i) for which $x_i \notin X_M$.

Because $|Y| = \mathcal{O}(|T|^2)$, we also have that $|X_M| = |M| = \mathcal{O}(|T|^2)$, which in turn means that $|P'| = \mathcal{O}(|T|^2)$. Since $T' = T$, this gives us that G' has $\mathcal{O}(|T|^2)$ vertices and therefore $\mathcal{O}(|T|^3)$ edges.



(a) *Input graph* G. Example of an unbalanced bipartite graph G.

(b) *Auxiliary graph* B. Step 1: construct the auxiliary graph B. Consider e.g. the vertices of (a_1, b_1). They are connected to t_1 and t_3 respectively in G, so x_1 is connected to y_1, y_3 and $y_{1,3}$ in B.

(c) *Auxiliary graph* B. Step 2: find a maximum matching M in B (the blue edges). The vertices in X_M are highlighted blue.

(d) *Output graph* G'. Step 3: remove people pairs from P corresponding to unmatched vertices in B to obtain G'.

Figure 4: Kernelization applied to an example graph. Figure 4a shows an example graph G. Figures 4b and 4c depict the auxiliary graph B and respectively show its construction and a maximum matching in it. They are drawn on a gray background to avoid confusion between the three different graphs depicted (G, B and G'). Figure 4d shows the resulting graph G'.

This concludes the kernelization. To show that this procedure gives a valid kernel for PM, we first show that it provides a correct reduction. Finally, we show that it can be performed in polynomial time.

Correctness of the kernelization. To see that the kernelization provides a correct reduction, we show that (G, c, f, d) is a YES-instance if and only if (G', c, f, d) is a YES-instance. Clearly, if (G', c, f, d) is a YES-instance, then so is (G, c, f, d), since G' is obtained from G by only removing people vertices. Then every matching in G' also exists in G, since G' is a subgraph of G. Moreover, since no task vertices were removed from G to obtain G', a matching that covers all task vertices in G' also does so in G.

Suppose now that (G, c, f, d) is a YES-instance, meaning that there is some matching M_G in G of cost $d^* \leq d$. If M_G only contains edges that are also present in G', then this matching also exists in G', making (G', c, f, d) trivially a YES-instance. So suppose M_G does not exist in G'. Then we will construct a matching $M_{G'}$ in G' with the same cost d^*.

To this end, we take a look at the graph B. Although B was just an auxiliary graph to construct G' (and is therefore not part of our final kernelization), it will be useful in constructing the matching $M_{G'}$. As explained above, the matching M_G in G could be encoded in B as some matching M_B. Let $Y' \subseteq Y$ denote the set of vertices in Y that are covered by this M_B. Each vertex in Y' represents an ordered combination of either one or two vertices that are matched to the same pair of people vertices by M_G.

Consider now again the maximum matching M in X from before and the set X_M of vertices in X covered by M. By Lemma 4.2, there exists a matching M'' in $B[X_M \cup Y]$ covering all the vertices in Y'. We use this matching M'' to construct $M_{G'}$. We do the following for every edge $e \in M''$:

- If e has endpoints $x_i \in X$ and $y_j \in Y_1$, then we add either $\{a_i, t_j\}$ or $\{b_i, t_j\}$ to $M_{G'}$, depending on which of these edges is present in G'. We know that at least one of these edges is present in G', because e exists in B and $x_i \in X_M$.
- If e has endpoints $x_i \in X$ and $y_{j_1,j_2} \in Y_2$, then we add the edges $\{a_i, t_{j_1}\}$ and $\{b_i, t_{j_2}\}$ to $M_{G'}$. Because e exists in B, both these edges are present in G and because $x_i \in X_M$, this edge also exists in G'.

Because M'' covers the same vertices in Y as M_B does, this construction of $M_{G'}$ leads to a matching in G', which matches the same combinations of vertices to the same pair of people vertices as M_G did in G. Not only does this make $M_{G'}$ a valid matching in G', but it also has the same cost d^* as M_G, making (G', c, f, d) a YES-instance.

The kernelization takes polynomial time. The kernelization can be split into three steps, each of which takes polynomial time. The first step is to construct the auxiliary graph $B = (X + Y, E'')$. A single vertex is added to X for every people pair in P. For every individual task vertex and for every combination of two task vertices, a single vertex is added to Y, amounting to a quadratic number of vertices in total. Then for every pair of a vertex from X and a vertex from Y, the existence of the edge between them depends on the existence of one or two edges in G, which can be checked in polynomial time.

The second step is then to find a matching in B, which can be done in polynomial time, given that B is polynomial in size [26].

Given a matching in B, the third step is to determine which vertices from X are covered by the matching. For every vertex for which this is not the case, the corresponding people pair in G and its incident edges are removed to obtain G'. All these steps can again be done in polynomial time. □

Of course, performing just the kernelization on a PM instance does not yet yield an answer to the problem. Combining it with a brute-force algorithm however, yields a procedure solving PM in FPT time with respect to $|T|$.

Corollary 4.4. *PM is FPT with respect to* $|T|$.

Proof. Let (G, c, f, d) be a PM-instance with $G = (P + T, E)$. To solve it in FPT time with respect to $|T|$, the first step would be to perform the kernelization from Proposition 4.3. This yields an equivalent instance (G', c, f, d) in which G has $\mathcal{O}(|T|^2)$ vertices and $\mathcal{O}(|T|^3)$ edges.

The second step is to solve the new instance using a brute-force algorithm. A very naive way in which this can be done is to iterate over all $2^{\mathcal{O}(|T|^3)}$ subsets of edges in G' and checking whether they are a matching such that all task vertices are covered at a cost of at most d.

Since $|G'| = |T|^{\mathcal{O}(1)}$, it can be checked in $|T|^{\mathcal{O}(1)}$ time, whether an edge set is a matching. If c and f are very large compared to $|T|$, it may not be possible to compute the cost of a given matching within time depending asymptotically only on $|T|$. If we let U denote the maximum value attained by c and f however, we can express the time needed to compute the cost of a given matching as $(\log U \cdot |T|)^{\mathcal{O}(1)}$, when c and f are stored in binary. This yields an algorithm to solve (G, c, f, d) in $2^{\mathcal{O}(|T|^3)}(\log U \cdot |T|)^{\mathcal{O}(1)}$ time. □

5 Parameterized complexity of PM: dense graphs

In this section and in Section 6, we consider parameterizations of PM using structural parameters. With parameters like these, we aim to measure the complexity of the input and hope to find algorithms that can solve our problem efficiently on large inputs as long as their complexity is not too high. What constitutes as "complex" may vary between problems and input types. Complexity measures on graphs typically regard many classes of sparse graphs as less complex than many dense graphs. In this section however, we focus instead on parameters that generally attain low values for very dense graphs, while they attain higher values for more sparse graphs. This is because of the observation that PM can be solved in polynomial time on complete bipartite graphs, as shown in Section 5.1.

In Sections 5.2–5.5, we consider four different parameterizations of PM that can each be interpreted as a distance measure of "how far a graph is from being complete bipartite". Furthermore, we find each of these parameters to be unaffected by the reduction from PM to BPM presented in Section 3.3. Hence, we can work with balanced graphs, which is often more practical than to work with unbalanced graphs, without affecting the validity of the results applied to PM in a general setting.

5.1 PM on complete bipartite graphs

As mentioned, PM can be solved in polynomial time on complete bipartite graphs. Under this restriction, the problem can be reduced to the Weighted Perfect Matching problem:

Weighted Perfect Matching (WPM)	
Given:	A graph $G = (V, E)$, an edge weight function $w : E \to \mathbb{R}_{\geq 0}$ and a budget $d \in \mathbb{R}_{\geq 0}$.
Objective:	Determine whether a perfect matching exists in G with weight at most d.

The optimization version of the problem is known as the Minimum Weight Perfect Matching problem (MWPM), in which the goal is to find a perfect matching of minimum weight. MWPM problem has long been known to be solvable in polynomial time and the current best algorithm to solve it in general graphs runs in $\mathcal{O}(n(m + n \log n))$ time, where n and m are the number of vertices and edges respectively [14]. By extension, WPM can also be solved in the same time.

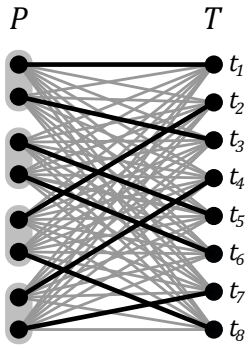
Proposition 5.1. *PM is solvable in polynomial time on complete bipartite graphs.*

Proof. We show that every BPM instance can be reduced to an equivalent WPM instance in polynomial time. Since WPM is solvable in polynomial time [14], it follows that BPM can also be solved in polynomial time. Note that this suffices to prove the statement as every complete PM instance can be reduced to an equivalent BPM instance. This follows from the observation that the reduction in Lemma 3.4 only adds task vertices that are connected to all people vertices.

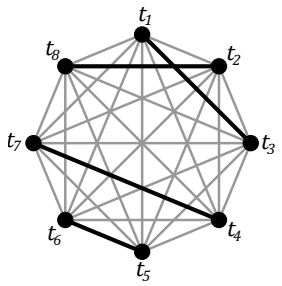
Now let $(G = (P + T, E), f, d)$ be a BPM instance. Then let $G' = (V, E')$ be the complete graph with vertex set $V = T$ and let $w : E \to \mathbb{R}$ be given by $w(\{t_1, t_2\}) = \min\{f(t_1, t_2), f(t_2, t_1)\}$ to construct WPM instance $(G' = (V, E'), w, d)$. Figure 5 shows an example of a complete bipartite BPM instance reduced to a (complete) WPM instance. It also shows how – as explained below – a matching in one instance can be transformed into a matching in the other. We use this to show that (G, f, d) and (G', w, d) are equivalent.

For one direction, we assume that M is a matching in G of cost $d^* \leq d$ and use it to construct a perfect matching M' in G' with weight at most d^*. Let (a_i, b_i) be some pair of people vertices in P. Since G is balanced and M is a perfect matching, both vertices of this people pair are covered by M. Let $t_1, t_2 \in T$ be the two task vertices matched to a_i and b_i respectively. Then we add $\{t_1, t_2\} \in E'$ to M' which has weight $\min\{f(t_1, t_2), f(t_2, t_1)\} \leq f(t_1, t_2)$. Doing this for every people pair in P thus results in an edge set $M' \subseteq E'$ with weight at most $d^* \leq d$. Moreover, M' is a perfect matching, since every task vertex in T is incident with exactly one edge from M.

For the other direction, we assume that M' is a matching in G' with weight $d^* \leq d$ and use it to construct a perfect matching M in G with cost d^*. Suppose that the edges of M' have an arbitrary ordering $e_1, e_2, \ldots, e_{|T|}$ and let $e_i = \{t_1, t_2\} \in M'$. If $f(t_1, t_2) \leq f(t_2, t_1)$, then we add the edges $\{a_i, t_1\}$ and $\{b_i, t_2\}$ to M. Otherwise, we add the edges $\{a_i, t_2\}$ and $\{b_i, t_1\}$ instead, meaning that the pair (a_i, b_i) contributes $\min\{f(t_1, t_2), f(t_2, t_1)\} = w(e_i)$ to the total cost of M. Doing this for every edge in M', results in an edge set $M \subseteq E$ with cost d^*. And because M' is a perfect matching in G', M is a perfect matching in G. □



(a) A complete bipartite example graph G. The black edges form a perfect matching M.



(b) The resulting graph G' obtained from G using the reduction from Proposition 5.1. The black edges are the complete matching M' in G' corresponding to the matching M in G.

Figure 5: A complete BPM instance reduced to a WPM instance. The matching M in G can be obtained from M' in G' by looking at the matched vertices of each people pair in P: t_1 and t_3 are matched to the same people pair in G, so they are connected by M' in G'. t_5 and t_6 are matched to the same people pair in G, so they are connected by M' in G', etc. Matching M' can equivalently be transformed into M: t_1 and t_3 are connected by M', so they are matched to the same people pair in G by M, etc.

Knowing that the problem is easy when restricted to complete bipartite graphs, we investigate the complexity of PM on graphs that are close to being complete bipartite (for various definitions of "close").

5.2 PM parameterized by the number of non-edges

To measure "how far a graph is from being complete bipartite", it seems logical to look at the non-edges of the graph, i.e.: the pairs of vertices that could be connected, but are not. By defining a metric on bipartite graphs that takes value 0 on complete bipartite graphs and (in general) higher values the more edges are missing from the graph, we have a measure that assigns each bipartite graph a distance to being complete bipartite. A very basic measure that would achieve this, would simply be the total number of non-edges of the bipartite graph. Since we only consider bipartite graphs, pairs from the same partite set are by definition not connected. Rather than counting the absence of such edges as non-edges as well, we use the following definition.

Definition 5.2 (Non-edge)**.** *Let $G = (P + T, E)$ be a bipartite graph. We define the set of non-edges of G to be $\{\{p, t\} \mid p \in P, t \in T, \{p, t\} \notin E\}$.*

We show that PM is FPT when parameterized by the number of non-edges in the graph. This is achieved by showing that, even for a linear number of non-edges, the reduction in the proof of Proposition 5.1 can be used to solve PM. More specifically, we show

that this reduction to WPM still works when the number of non-edges is at most $n/2$. This then gives an FPT algorithm where the problem is solved in polynomial time when the number of non-edges is at most $n/2$ using a subroutine for solving WPM and the problem is bruteforced when the number of non-edges is more than $n/2$. Recall that by definition of n, we have $|P| = 2n$.

Lemma 5.3. *Let $(G = (P+T, E), f, d)$ be a BPM instance and let (G', w, d) be the WPM instance obtained from it using the reduction presented in the proof of Proposition 5.1. If G has at least $(2n)^2 - n/2$ edges, then the minimum cost paired matching in G has cost d if and only if the minimum weight perfect matching in G' has weight d.*

Proof. Equivalently to the proof of Proposition 5.1, it holds that if G has a paired matching of cost d, then G' has a perfect matching with weight at most d. It remains to show that G has a paired matching M of cost d when G' has a perfect matching M' with weight d. Like in Proposition 5.1, we could construct M from M', by matching two task vertices to the same pair in P if the corresponding vertices in G' are matched to each other by M'. Because G may not be a complete bipartite graph, we need to be careful however that we only match task vertices to pairs they are actually connected to.

This can be done due to the limited number of non-edges of G. Since $|P| = |T| = 2n$ and G has at least $(2n)^2 - n/2$ edges, it has at most $n/2$ non-edges. This means that at most $n/2$ task vertices have missing edges and that at most $n/2$ pairs in P have missing edges. Since there are n pairs in P, this also means that there are at least n such pairs in which both vertices are connected to *all* vertices in T. Now when constructing our paired matching M from M' as in Proposition 5.1, we do not use arbitrary people pairs to match the tasks. Instead, when we are considering some edge $\{t_1, t_2\} \in M'$ which includes at least one task with incident non-edges, we use one of the people pairs that are connected to all vertices in T to match these tasks. Because there are at least n such pairs in P, this suffices to cover all these task vertices. For the remaining edges in M' we can use any people pair, as these remaining edges only match task vertices together which are connected to all people vertices. □

As discussed above, this lemma indicates a strategy for an FPT algorithm. When the graph in a PM instance has at most $n/2$ non-edges, Lemma 5.3 shows that the problem can be solved in polynomial time using a reduction to WPM. Otherwise, the size of the graph is polynomial in the number of non-edges, meaning that a brute-force algorithm (like e.g. presented in Corollary 4.4) runs in FPT time w.r.t. the number of non-edges.

Corollary 5.4. *PM if FPT with respect to the number of non-edges in the graph.*

5.3 PM parameterized by the number of picky tasks

Instead of counting the non-edges in the graph themselves, we can look at the vertices incident with non-edges. The parameterizations in this subsection and the next two consider different ways of counting these. This results in lower parameter values for the same graph than would be attained by simply counting the non-edges. In Section 5.5, we consider the minimum size of a vertex cover of the non-edges as a parameter. Before we do that however, it is both useful and insightful to develop algorithms for solving the problem when parameterizing by either the number of people or task vertices that are incident with non-edges. We call such vertices *picky*.

Definition 5.5 (Picky vertex)**.** *Let $G = (P + T, E)$ be a bipartite graph. A vertex $v \in P \cup T$ is called picky if it is contained in at least one of the non-edges of G.*

In this section, we consider PM parameterized by the number of picky task vertices and in Section 5.4 we consider the parameterization by the number of picky people vertices. Finally, in Section 5.5, we combine these results to give an XP algorithm for PM parameterized by the size of a minimum vertex cover of the non-edges of G.

5.3.1 Some preliminaries

Before giving an FPT algorithm in Section 5.3.2, we introduce some additional concepts and notation.

First, we introduce the notion of the *task assignment* of a people pair $p = (a_i, b_i)$ for some matching M. We denote it using $\mathrm{TA}_M(p)$. This task assignment is defined as a sequence of length 2 indicating which task vertices are matched to a_i and b_i respectively. In case a vertex is left uncovered by M, we use the special 0-element ϵ to denote this. The first element of $\mathrm{TA}_M(p)$ is then either the unique task vertex t_1 such that $\{a_i, t_1\} \in M$ or ϵ if no such vertex exists. Likewise, the second element of $\mathrm{TA}_M(p)$ is the unique task vertex t_2 such that $\{b_i, t_2\} \in M$ or ϵ if no such vertex exists.

Next, we define the *task distribution* of a matching M in G, which we denote by TD_M. It is defined as the union of $\mathrm{TA}_M(p)$ for every people pair p with at least one vertex covered by M. This means that TD_M is a set of sequences of length 2, where each sequence contains either two task vertices or a task vertex together with ϵ. Because the cost of M is only determined by the combinations of tasks matched to the same people pair, the cost of M can be computed directly from TD_M. This makes it a useful definition to work with.

Now we denote by $\mathcal{D}_S$ the entire collection of possible task distributions on some set S. By this, we mean that for a given set S, $\mathcal{D}_S$ contains all unique sets of length-2 sequences such that:

- each sequence consists either of two distinct elements from S or the 0-element ϵ together with one element from S
- and each element from S occurs in exactly one of these sequences.

Similarly, for a set S of even cardinality, we denote by $\mathcal{S}_S$ the collection of unique sets of length-2 sequences using every element of S exactly once. This means that $\mathcal{S}_S$ is the subset of all task distributions $\mathcal{D}_S$ on S containing only the ones corresponding to a matching that covers either both or none of the vertices of each people pair.

In Sections 5.3.2 and 5.4, we present algorithms that iterate over the sets $\mathcal{D}_S$ and $\mathcal{S}_S$ respectively for a given set S. Before presenting the algorithms, we therefore analyze how fast these two sets can be iterated over.

Lemma 5.6. *For a given set S of even size, the set $\mathcal{S}_S$ can be iterated over in $2^{\mathcal{O}(|S| \log |S|)}$ time.*

Proof. Since $\mathcal{S}_S$ is defined as the collection of unique partitions of S into ordered pairs, every element in this collection can be uniquely defined based on the following two choices:

(1) Which half of the elements of S appears as the first element in one of the ordered pairs. The other half of the elements then appears as the second element in a pair.

(2) Given a choice for (1), which combinations of elements are partitioned into the same ordered pair.

To iterate over all options for this first choice, we can iterate over all subsets $S_a \subseteq S$ of size $|S|/2$, which can be done in $2^{\mathcal{O}(k)}$ time [33], and define $S_b = S \setminus S_b$. Then S_a and S_b represent a choice for which elements of S appear as the first and second element of an ordered pair respectively.

Given such a choice where the elements in S_a are chosen to appear as the first element of a pair and the elements in S_b as the second element of a pair, we still need to pair each element in S_a with an element from S_b. To iterate over all the possible options here, we fix an arbitrary ordering on the elements in S_a and iterate over all permutations of S_b, which can be done in $2^{\mathcal{O}(|S| \log |S|)}$ time [21]. Then pairing the i-th element of S_a with the i-th element of a permutation of S_b, yields one of the possible partitions of S into ordered pairs.

By doing this for every permutation of S_b, for every choice of S_a and S_b, we go through every element in $\mathcal{S}_S$ in $2^{\mathcal{O}(|S| \log |S|)}$ time. □

Given a running time bound on the iteration over $\mathcal{S}_S$, we can now also give one for the iteration over the superset $\mathcal{D}_S$.

Lemma 5.7. *For a given set S, the set $\mathcal{D}_S$ can be iterated over in $2^{\mathcal{O}(|S| \log |S|)}$ time.*

Proof. As noted earlier, $\mathcal{D}_S$ is a superset of $\mathcal{S}_S$. The difference between the sets is that the elements in $\mathcal{D}_S$ are allowed to contain pairs in which a single ϵ occurs. This observation allows us to characterize each of the elements in $\mathcal{D}_S$ by the following three choices:

(1) Which subset $S_1 \subseteq S$ is contained in pairs that also contain an ϵ. The remaining elements $S_2 = S \setminus S_1$ are then each contained in a pair with another element from S_2.

(2) Given a choice for (1), which of the elements of S_1 appear as the first in their pair and which of them appear second.

(3) Given a choice for (1) and (2), how the elements of S_2 are partitioned into ordered pairs.

To iterate over all options for the first choice, we can simply iterate over all subsets $S_1 \subseteq S$ and compute S_2 as $S \setminus S_1$, which can be done in $2^{\mathcal{O}(|S|)}$ time. The only limitation we need to be careful of is that S_2 needs to be of even size, since this set indicates the elements that are grouped into pairs with one another. Therefore, we simply skip all iterations in which $|S_2|$ is odd.

For the second choice, we assume we are given a set S_1 and S_2 already. To iterate over all options for the second choice, we can iterate over all subsets $S_1' \subseteq S_1$ such that the elements in S_1' are chosen to appear first in their ordered pair, while the elements in $S_1 \setminus S_1'$ are chosen to appear second in their ordered pair. This can be done in $2^{\mathcal{O}(|S_1|)}$ time.

Finally, for the third choice, we assume to have already made a decision for choices (1) and (2), meaning we are in particular given a set S_2 indicating the elements from S which are contained in pairs with one another. We need to iterate over all possible partitions of S_2 into ordered pairs. By definition, the collection $\mathcal{S}_{S_2}$ contains all of these partitions and from Lemma 5.6 we know that we can iterate over this set in $2^{\mathcal{O}(|S_2| \log |S_2|)}$ time.

It follows then that iterating over all options for choice (3), for all options for choice (2), for all options for choice (1) can be done in $2^{\mathcal{O}(|S| \log |S|)}$ time. □

5.3.2 An FPT algorithm

Using the concepts from the section above, we can describe an algorithm that solves PM in FPT time when parameterized by the number of picky task vertices. Pseudocode for this algorithm can be found in Algorithm 3. We let k denote the number of picky task vertices.

The idea is to split the problem into a hard and an easy part. From Section 5.1, we already know how to deal with instances without picky vertices in polynomial time. This suggests that it could be harder to deal with picky vertices than to deal with non-picky vertices. The high-level strategy of the algorithm is therefore to find matchings for all different possible task distributions of the picky tasks and computing for each of them the cheapest way to extend them into a matching that covers all task vertices. To achieve this, the algorithm uses two subroutines. The first is outlined in Algorithm 1 and is used to find a matching in the input graph given a desired task distribution. The other is outlined in Algorithm 2 and is used to determine what the cheapest way is to extend a

smaller matching into one that covers all task vertices. Before proving the correctness of Algorithm 3, we prove the correctness of each of these two subroutines.

Algorithm 1 Subroutine for FPT algorithm

1: **function** MATCHING-FROM-TASK-DISTRIBUTION($G = (P + T, E), D$)
2: $X \leftarrow \{(a_1, b_1), (a_2, b_2), \ldots, (a_n, b_n)\}$
3: $Y \leftarrow D$
4: $E' \leftarrow \emptyset$
5: **for** $d \in D$ **do**
6: **for** every people pair $p \in X$ which could have task assignment d in G **do**
7: $E' \leftarrow E' \cup \{\{p, d\}\}$
8: **end for**
9: **end for**
10: $G' \leftarrow (X + Y, E')$
11:
12: Let M' be a maximum matching in G'.
13: **if** $|M'| < |Y|$ **then return** NO
14: $M \leftarrow \emptyset$
15: **for** $\{p, d\} \in M'$ **do**
16: Let $p = (a_i, b_i)$ and let $d = (t_1, t_2)$.
17: **if** $t_1 \neq \epsilon$ **then** $M \leftarrow M \cup \{\{a_i, t_1\}\}$
18: **if** $t_2 \neq \epsilon$ **then** $M \leftarrow M \cup \{\{b_i, t_2\}\}$
19: **end for**
20: **return** M
21: **end function**

Lemma 5.8. *Let $G = (P + T, E)$ be as in a PM instance: with the $|P| = 2n$ people vertices partitioned into ordered pairs $(a_1, b_1), (a_2, b_2), \ldots, (a_n, b_n)$. When Algorithm 1 is given G and for some $T' \subseteq T$ a task distribution $D \in \mathcal{D}_{T'}$, it returns in $n^{\mathcal{O}(1)}$ time either a matching with task distribution D or* NO *if no such matching exists in G.*

Proof. The algorithm works by finding a maximum matching in an auxiliary bipartite graph $G' = (X + Y, E')$. X contains a vertex for each people pair in P and Y contains a vertex for each of the length 2 sequences in D. A vertex $x \in X$, corresponding to people pair $p = (a_i, b_i)$ from P, and a vertex $y \in Y$, corresponding to task assignment $d = (t_1, t_2)$ are connected in G' if p could have task assignment d. By this, we mean that both (a_i, t_1), $(b_i, t_2) \in E$. If $t_1 = \epsilon$ or $t_2 = \epsilon$, then the requirement corresponding to that task vertex may be dropped. Using this construction, G' encodes which parts of the task distribution D could be fulfilled by which pair of people vertices in P. The algorithm constructs the graph G' on lines 2-10.

Suppose that M' is a matching in G' with $|M'| = |Y|$. Then a matching M in G with task distribution D can easily be constructed by adding the correct edges between the task vertices in d and the people pair p for every $\{p, d\} \in M'$. This is the matching constructed on lines 14-19 of the algorithm. Since $|M'| = |Y|$, every length 2 sequence in D gets assigned a people pair with that sequence as its task assignment. Hence, M has task distribution D and we are thus correct to return this matching on line 20.

Likewise, if there does not exist a matching in G' which covers all vertices of Y, then there is no way of assigning unique people pairs to each of the length 2 sequences in D such that they could have this sequence as their task assignment in the same matching. I.e.: there is no matching M in G with TD_M. We are therefore correct to return NO on line 13.

Running time. Since D is a task distribution for a subset of tasks in T, it contains $\mathcal{O}(n)$ elements. For a given people pair, it can be verified in $\mathcal{O}(n)$ time whether it could have a given task assignment by going over the outgoing edges of the people pair. The nested for-loops on lines 3-9 to construct G' therefore execute in $\mathcal{O}(n^2)$ time and the resulting graph has $\mathcal{O}(n)$ vertices and $\mathcal{O}(n^2)$ edges. Finding a maximum matching M' in this graph can be done in $\mathcal{O}(n^{2.5})$ time [26] and transforming this into a matching M in G can be done in $\mathcal{O}(n)$ time. This gives the algorithm a total running time of $n^{\mathcal{O}(1)}$. □

Algorithm 2 Subroutine for FPT algorithm

function PARTIALLY-SOLVED-PM($G = (P + T, E), c, f, M$)
  Assume G is balanced (or apply the reduction from Lemma 3.4 otherwise).

  Let $T_1 \subseteq T$ be the tasks matched by M to a pair with no other task matched to it.
  Let $T_2 \subseteq T$ be the tasks matched by M to a pair with another task matched to it.
  Let p_2 denote the cost to match the vertices of T_2 using M.

  $V' \leftarrow T \setminus T_2$
  Let E' be the edge set of the complete graph with vertex set V'.
  $E' \leftarrow E' \setminus (T_1 \times T_1)$
  $G' \leftarrow (V', E')$
  for $\{t_1, t_2\} \in E'$ **do**
    if $t_1, t_2 \notin T_1$ **then**
      Define $w(\{t_1, t_2\}) = \min\{f(t_1, t_2), f(t_2, t_1)\}$.
    else
      Assume w.l.o.g. that t_1 is covered by M.
      Let (a_i, b_i) be the people pair that t_1 is matched to by M.
      Define $w(t_1, t_2) = \begin{cases} f(t_1, t_2) & \text{if } \{t_1, a_i\} \in M, \\ f(t_2, t_1) & \text{if } \{t_1, b_i\} \in M. \end{cases}$
    end if
  end for
  Let p_1 be the minimum w-weight of a perfect matching in G'.
  return $p_1 + p_2$
end function

Lemma 5.9. *When given an MPM instance $(G = (P + T, E), c, f)$ with $|P| \geq |T|$ and a matching M in G, such that the subgraph of G induced by the uncovered vertices of M is complete bipartite, then Algorithm 2 returns in $n^{\mathcal{O}(1)}$ time the minimum cost of a matching $M' \supseteq M$ with $|M'| = |T|$.*

Proof. First note that we can assume w.l.o.g. that G is balanced or apply the reduction from PM to BPM as presented in Lemma 3.4 otherwise. This reduction only adds task vertices that are, of course, not covered by M, but which are connected to all people vertices. By assumption, the subgraph induced by the uncovered vertices of G was complete bipartite, so this is then still the case in the balanced graph obtained from the reduction.

Next we note that people pairs with both vertices matched by M have a fixed contribution to the cost, regardless of how we extend it into a larger matching M'. By computing this cost beforehand (as p_2 in the algorithm), we can ignore these vertices

for the remainder of the algorithm. Moreover, the graph obtained by removing these people pairs and the task vertices that they are matched to by M yield again a balanced bipartite graph. The minimum cost of a matching $M' \supseteq M$ of size $|T|$ is then the sum of the cost to cover T_2 plus the minimum possible cost to cover the remaining vertices (p_1 in the algorithm).

How the remaining vertices should be matched using minimum cost is solved by a reduction to MWPM. By the restriction on the input, the subgraph of G induced by the uncovered vertices of M forms a complete bipartite graph, which makes the problem very similar to the one in Proposition 5.1. The proof of correctness for the algorithm is therefore also mostly analogous to that of Proposition 5.1, so we only highlight the differences between the problems and explain why these are correctly reflected in the algorithm. We assume that from now on there are no people pairs from which both vertices are covered by M, as they can be dealt with separately as explained above. This is also made explicit in the algorithm on line 6 by computing the contribution of these vertices separately and on line 8 by excluding these vertices from the reduction to MWPM.

The main insight from the proof of Proposition 5.1 is that it matters only which task vertices get matched to the same people pair and not to which particular people pair. This idea is then translated into finding a minimum weight perfect matching between the task vertices in a complete graph. Since we are already given a matching M that our solution M' should be a superset of, there are two differences that should be taken into account.

- Firstly, none of the task vertices already covered by M can be matched together in our solution M'. Since we assumed that at this point every people pair has at most one vertex covered by M, meaning that every covered task vertex is matched to a different people pair. This is reflected in the algorithm by a slight alteration of the reduction. Rather than solving MWPM on a complete graph of the task vertices, we leave out all edges between two covered vertices, which is explicitly done on line 10. This then prevents two covered task vertices to be mapped to the same people pair, but leaves all other options open, as desired.
- Secondly, when an uncovered task vertex t_2 gets matched to the same people pair (a_i, b_i) by M' as some already covered task vertex t_1, their cost is not necessarily given by the minimum of $f(t_1, t_2)$ and $f(t_2, t_1)$. We no longer have the freedom of picking which of the two task vertices gets matched to a_i and which gets matched to b_i respectively. Instead, it is already fixed to which specific person t_2 gets matched to, which is reflected by defining the weight function w as in the algorithm. Specifically, this case is covered on line 19: if t_1 is matched to a_i, then t_2 can only be matched to b_i, making the pair (a_i, b_i) contribute $f(t_1, t_2)$ to the cost. Likewise, if t_1 is matched to b_i, then t_2 can only be matched to a_i, making the pair (a_i, b_i) contribute $f(t_2, t_1)$ to the cost.

Running time. The auxiliary graph G' has $\mathcal{O}(n)$ vertices and $\mathcal{O}(n^2)$ edges and can be constructed in $\mathcal{O}(n^2)$ time, as can the weight function w. Finding the minimum weight of a perfect matching in G' can be done in $\mathcal{O}(n^{2.5})$ time [26], giving the algorithm a total running time of $n^{\mathcal{O}(1)}$. □

Algorithm 3 FPT algorithm for PM parameterized by the number of picky tasks

```
1: function PM-PICKY-TASKS(G = (P + T, E), c, f, d, k)
2:     Let T' ⊆ T be the k picky tasks.
3:     for every possible task distribution D ∈ D_{T'} do
4:         M ← MATCHING-FROM-TASK-DISTRIBUTION(G, D)
5:         if M = NO then skip this iteration.
6:         if PARTIALLY-SOLVED-PM(G, c, f, M) ≤ d then return YES
7:     end for
8:     return NO
9: end function
```

Proposition 5.10. *Let k be the number of picky task vertices in G. Then PM parameterized by k is FPT and Algorithm 3 solves it in $2^{\mathcal{O}(k \log k)} \cdot n^{\mathcal{O}(1)}$ time. Hence, this parameterization of PM is in FPT.*

Proof. Given the correctness of the subroutines MATCHING-FROM-TASK-DISTRIBUTION (Algorithm 1) and PARTIALLY-SOLVED-PM (Algorithm 2), the correctness of Algorithm 3 is not hard to see. We iterate over all possible task distributions for matching the picky vertices. If there exists a matching M with that task distribution, we compute the minimum cost of extending this matching into one of size $|T|$. Since M covers all picky tasks, the subgraph induced by the uncovered vertices of M is complete bipartite, meaning that we can use PARTIALLY-SOLVED-PM to do this. If there is an iteration in which we find this extension to be possible at a cost of at most d, then we have found a YES-instance and return YES at line 6.

If we encounter no such iteration, then there is apparently no matching of size $|T|$ with cost at most d. After all, we iterate over all task distributions in $\mathcal{D}_{T'}$ to match the picky task vertices so in particular also the one of a minimum solution. Hence, we return NO at line 8.

Running time. We have already determined the running times of the subroutines MATCHING-FROM-TASK-DISTRIBUTION and PARTIALLY-SOLVED-PM, both of which are $\mathcal{O}(n^{2.5})$. This gives us the running time of a single iteration of the main loop. The loop iterates over the elements in $\mathcal{D}_{T'}$ and by Lemma 5.7, we know that this can be done in $2^{\mathcal{O}(k \log k)}$ time. This gives the algorithm a total running time of $2^{\mathcal{O}(k \log k)} n^{\mathcal{O}(1)}$. □

5.4 PM parameterized by the number of picky people

The next parameterization of PM to consider is the parameterization by the number of picky people vertices. We use a similar idea to solve this as in the previous section: try all possible options to match the picky vertices and find the minimum cost extension of it into a matching of size $|T|$. When the parameter is the number of picky people vertices however, this technique yields an XP algorithm rather than an FPT algorithm, due to the asymmetry in the problem between the roles of the people vertices and the task vertices.

Algorithm 4 XP algorithm for PM parameterized by the number of picky people

```
1: function PM-PICKY-PEOPLE(G = (P + T, E), c, f, d, k)
2:     Assume G is balanced (or apply the reduction from Lemma 3.4 otherwise).
3:     Let P' ⊆ P be the people in a pair with at least one picky person.
4:     for T' ∈ (T choose |P'|) do
5:         for D ∈ S_{T'} do
6:             M ← MATCHING-FROM-TASK-DISTRIBUTION(G[P' ∪ T'], D)
7:             if M = NO then skip this iteration.
8:             if PARTIALLY-SOLVED-PM(G, c, f, M) ≤ d then return YES
9:         end for
10:    end for
11:    return NO
12: end function
```

Proposition 5.11. *Let k be the number of picky people vertices in G. Then PM parameterized by k is in XP and Algorithm 4 solves it in $2^{\mathcal{O}(k \log k)} n^{\mathcal{O}(k)}$ time. Hence, this parameterization of PM is in XP.*

Proof. Before arguing the correctness of the algorithm, we briefly highlight that we can assume w.l.o.g. that the input graph G is balanced. As mentioned on line 2 of the algorithm, we could apply the reduction from Lemma 3.4 otherwise. This yields a BPM instance that is not only equivalent to the original PM instance, but which also has the same number of picky people. After all, the only modification to the graph is the addition of a set of task vertices that are all connected to all people vertices.

The correctness of the algorithm now follows from the correctness of the two subroutines used and from the fact that the input graph is balanced. The latter property means that every vertex in P' is covered by a matching of size $|T|$. We therefore iterate over all subsets of $|T|$ of size $|P'|$ and for each of these subsets we iterate over all the possible task distributions involving exactly this subset of task vertices. For every such task distribution, we check if there is a matching M that covers exactly the vertices $P' \cup T'$ using that task distribution. If so, we compute the minimum cost of extending this matching into one of size $|T|$. Since M covers all picky people, the subgraph induced by the uncovered vertices of M is complete bipartite, meaning that we can use PARTIALLY-

SOLVED-PM (Algorithm 2) to do this. If there is an iteration in which we find this extension to be possible at a cost of at most d, then we have found a YES-instance and return YES at line 8.

If we encounter no such iteration, then there is apparently no matching of size $|T|$ with cost at most d. After all, we iterate over all task distributions of $|P'|$ tasks, in particular the one that is used in an optimal solution to match the vertices that are matched to P'. Hence, we return NO at line 11.

Running time. First note that one iteration of the inner loop of the algorithm runs in $n^{\mathcal{O}(1)}$ time, which follows from Lemmas 5.8 and 5.9. The outer loop runs for $\binom{2n}{2k} = n^{\mathcal{O}(k)}$ iterations [33]. The inner loop iterates over $\mathcal{S}_{T'}$ for various sets of size $|P'| = k$, which can be done using $2^{\mathcal{O}(k \log k)}$ iterations by Lemma 5.6. This gives the algorithm a total running time of $2^{\mathcal{O}(k \log k)} n^{\mathcal{O}(k)}$ time. □

5.5 PM parameterized by a vertex cover of the non-edges

The last parameter we consider for measuring the density of the graph is a vertex cover of the non-edges. While there could be a lot of picky vertices in the graph, we might need only a few of them to cover the non-edges of the graph. This makes that the size of such a vertex cover can be even smaller than the number of picky task vertices or people vertices. With a small modification to Algorithm 4, we can solve PM in XP time w.r.t. the size of the vertex cover of the non-edges. Algorithm 5 shows this modification. It takes as input a PM instance (G, c, f, d).

Algorithm 5 XP algorithm for PM parameterized by a vertex cover of the non-edges

```
1: function PM-VC-NON-EDGES(G = (P + T, E), c, f, d)
2:     Assume G is balanced (or apply the reduction from Lemma 3.4 otherwise).
3:     Compute a minimum size vertex cover X of the non-edges of G.
4:     Let P' ⊆ P be the people in a pair with at least one vertex from X ∩ P.
5:     for T' ∈ (T choose |P'|) do
6:         for D ∈ S_{T'} do
7:             M ← MATCHING-FROM-TASK-DISTRIBUTION(G[P' ∪ T'], D)
8:             if M = NO then skip this iteration.
9:             G' ← G − (P' ∪ T')
10:            Let k be the number of picky tasks in G'.
11:            if PM-PICKY-TASKS(G', c, f, d − cost(M), k) = YES then return YES
12:        end for
13:    end for
14:    return NO
15: end function
```

Lemma 5.12. *Let k be the size of a minimum vertex cover of the non-edges of G. Then PM parameterized by k is in XP and Algorithm 5 solves it in $2^{\mathcal{O}(k \log k)} \cdot n^{\mathcal{O}(k)}$ time.*

Proof. Before arguing the correctness of the algorithm, we briefly highlight that we can assume w.l.o.g. that the input graph G is balanced. Otherwise, the reduction from Lemma 3.4 would yield an equivalent graph in which the set of non-edges is the same. This is because the reduction only adds task vertices that are connected to all people vertices, meaning no additional non-edges are introduced. The size of a minimum size vertex cover of the non-edges therefore remains unaltered by this reduction.

Now let $P' \subseteq P$ be the people in a pair with at least one vertex from $X \cap P$ as defined on line 2 of the algorithm. Since the graph is balanced, every matching of size $|T|$ is a perfect matching. Similar to the previous algorithm, we try to find a matching M' of size $|P'|$ that covers exactly P' and T' for every $T' \in \binom{T}{|P'|}$. If such a matching exists, then we check if there exists a matching $M \supseteq M'$ of size $|T|$ with cost at most d. Of course, this is the case if and only if $G' := G - (P' \cup T')$ has a matching of size $|T| - |T'|$ and cost at most $d - \mathit{cost}(M)$. However, G' has at most $|X|$ picky tasks, meaning that this subproblem can be solved in FPT time w.r.t. $|X|$, using PM-PICKY-TASKS (Algorithm 3). We show that this is the case, by arguing that all picky task vertices in G' must be in X.

Suppose for the sake of contradiction that there is a picky task vertex t in G' which is not in X. Since it is picky, there must be a vertex $p \in P \setminus P'$ such that $\{p, t\}$ is a non-edge of G'. Since X is a vertex cover of the non-edges and $t \notin X$, it must hold that $p \in X$. This is however a contradiction with the assumption that $p \notin P'$, which implies that also $p \notin X$. Hence, all picky task vertices in G' are in X and the number of picky task vertices is at most $|X|$.

If an iteration is encountered in which PM-PICKY-TASKS (Algorithm 3) returns YES, then a matching of cost at most d is found in G, hence yes is returned on line 11.

If no such iteration occurs, then there is apparently also no such matching. After all, G is balanced, so any matching of size $|T|$ is a perfect matching and therefore it covers in particular all vertices in P'. In the algorithm, every subset $T' \subseteq T$ of size $|P'|$ is considered as the set of task vertices to which P' is matched. If none of these choices however can be extended into into a perfect matching of cost at most d, there is no such matching in the graph, so no is returned on line 14.

Running time. Before the main loop of the algorithm, we determine a minimum size vertex cover X of the non-edges of G. Although this task is NP-hard in general graphs, the non-edges of G form, by definition, a bipartite graph. By Königs Theorem, the size of a minimum vertex cover equals the size of a maximum matching in bipartite graphs and a minimum vertex cover can then be found in polynomial time [5]. Let $k = |X|$.

Next, the nested for-loops are executed. The outer loop runs for $n^{\mathcal{O}(k)}$ iterations [33] and for each of those, the inner loop runs for $2^{\mathcal{O}(k \log k)}$ iterations by Lemma 5.6. In the

body of the inner loop, two subroutines are called, forming the bottleneck for running time in each iteration. The first, MATCHING-FROM-TASK-DISTRIBUTION (Algorithm 1), runs in $n^{\mathcal{O}(1)}$ time by Lemma 5.8. The second, PM-PICKY-TASKS (Algorithm 3), runs in $2^{\mathcal{O}(k \log k)} \cdot n^{\mathcal{O}(1)}$ time by Proposition 5.10.

This gives the algorithm a total running time of $2^{\mathcal{O}(k \log k)} \cdot n^{\mathcal{O}(k)}$. □

6 Parameterized complexity of PM: equivalence classes

Another measure for the complexity of the problem could be the number of distinguishable vertices in the graph. Some vertices could have identical properties and the fact that there are multiple of them may not add significant difficulty to solving the problem. To this end, we introduce the notion of *equivalence classes* in the graph, partitioning the vertices into sets of vertices with identical properties. We give definitions for this notion separately for people pairs and for task vertices. For a given PM instance with graph $G = (P + T, E)$, we let $P^* = \{(a_1, b_1), (a_2, b_2), \ldots, (a_n, b_n)\}$ denote the partition of P into ordered people pairs.

Definition 6.1 (Equivalence class of people pairs)**.** *Let (G, c, f, d) be a PM instance. Then $\mathcal{C}_{P^*} \subseteq 2^{P^*}$ is the unique partition of P^* such that two pairs $(a_1, b_1), (a_2, b_2) \in P^*$ belong to the same partite set if and only if $N_G(a_1) = N_G(a_2)$ and $N_G(b_1) = N_G(b_2)$. The elements of $\mathcal{C}_{P^*}$ are referred to as the equivalence classes of the people pairs of G.*

Two people pairs from the same equivalence class are then called *equivalent* or *indistinguishable*. In some sense, such people pairs really are indistinguishable. Consider e.g. some matching M in G and let t_1 and t_1' be the tasks matched by M to $(a_1, b_1) \in P^*$ in that order and let t_2 and t_2' be the tasks matched to $(a_2, b_2) \in P^*$ in that order. If (a_1, b_1) and (a_2, b_2) belong to the same equivalence class, we can "swap" them in the matching M (i.e.: change M to match t_2 and t_2' to (a_1, b_1) and t_1 and t_1' to (a_2, b_2)) to obtain a matching of the same cost. Because the pairs have identical neighborhoods, this swap can always be made and by definition of the cost function, the combined contribution to the cost of these pairs in either matching is given by $f(t_1, t_1') + f(t_2, t_2')$ and thus remains unchanged. This property makes it a useful definition to work with.

It is also of particular interest for the parameterized complexity of the Paired Matching problem, as the number of equivalence classes is typically very small in real-world instances from the application in pick-and-place machines. With this motivation, we first consider PM parameterized by $|\mathcal{C}_{P^*}|$ in Section 6.1. Then, in Section 6.2 we define a similar notion of equivalence classes for task vertices and consider the parameterization by the number of equivalence classes of this type.

6.1 PM parameterized by the number of equivalence classes of people pairs

To obtain results on the complexity of PM parameterized by $|\mathcal{C}_{P^*}|$, we introduce a different matching problem, which we show PM to be equivalent to. This different interpretation of PM is easier to work with and is more similar to other fundamental matching problems already studied in literature.

Color-Constrained Perfect Matching (CCPM)	
Given:	A multigraph $G = (V, E)$ in which each edge e is assigned a color $c(e)$ in $[k] := \{1, \ldots, k\}$ and a weight $w(e) \geq 0$. Also given are color budgets $c_1, \ldots, c_k$ and a weight threshold d.
Objective:	Determine whether a perfect matching exists in G with weight at most d and in which there are exactly c_i edges of color i.

This is a useful problem to consider as it relates to other problems that have already been studied, such as the Bounded Color Matching problem [34] and the Perfect Rainbow Matching problem [27]. Most notably, it generalizes the Exact Matching problem [30].

Theorem 6.2. *PM parameterized by $|\mathcal{C}_{P^*}|$ is equivalent under FPT reductions to CCPM parameterized by the number of edge colors k.*

Lemmas 6.3 and 6.5 each show one of the two directions of the required reduction to prove Theorem 6.2. Lemma 6.3 gives a reduction from PM to CCPM and Lemma 6.5 gives a reduction from CCPM to PM, together proving Theorem 6.2. For the sake of presentation, we call matchings in which each color $i \in [k]$ occurs exactly c_i times *color-constrained matchings*.

Lemma 6.3. *Every PM instance with $|\mathcal{C}_{P^*}| = m$ can be reduced in polynomial time to an equivalent CCPM instance with m different edge colors.*

Proof. Rather than showing that PM reduces to CCPM, we show that BPM reduces to CCPM. Note that this suffices to prove the statement, as the reduction from PM to BPM presented in Section 3.3 only adds task vertices with identical neighborhoods, thereby not altering the equivalence classes of people pairs. The resulting BPM instance therefore has the same number of equivalence classes of people pairs as the PM instance.

Let $(G = (P + T, E), f, d)$ be a BPM instance with $|\mathcal{C}_{P^*}| = m$ such that $\mathcal{C}_{P^*} = \{P_1^*, P_2^*, \ldots, P_m^*\}$. We reduce this to a CCPM instance $(G', w, c, (c_i)_{i \in [m]}, d)$ with m edge colors. Figure 6 shows the reduction performed on an example instance.

We start with the construction of the graph G' and the weight and color functions w and c. G' will have T as vertex set. For each combination of an integer $i \in [m]$ and two vertices $t_1, t_2 \in T$, we add an edge with color i between t_1 and t_2 in G' if t_1 and t_2 could be matched to the same pair $(a, b) \in P_i^*$. The weight of this edge will equal the minimum cost required to match t_1 and t_2 to the same pair in P_i^*. More specifically, let $(a, b) \in P_i^*$, then we do the following for every (unordered) combination of two distinct vertices $t_1, t_2 \in T$.

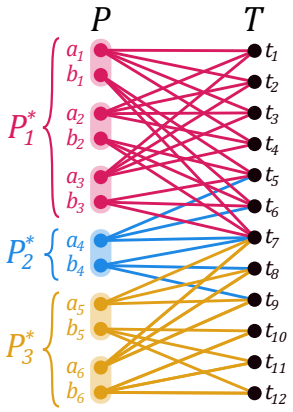
- If E contains all four of the edges $\{t_1, a\}, \{t_2, a\}, \{t_1, b\}, \{t_2, b\}$, we add an edge with weight $\min\{f(t_1, t_2), f(t_2, t_1)\}$ and color i between t_1 and t_2 in G'.
- Otherwise, if $\{t_1, a\} \in E$ and $\{t_2, b\} \in E$, we add an edge with weight $f(t_1, t_2)$ and color i between t_1 and t_2 in G'.

- Otherwise, if $\{t_1, b\} \in E$ and $\{t_2, a\} \in E$, we add an edge with weight $f(t_2, t_1)$ and color i between t_1 and t_2 in G'.
- Otherwise, t_1 and t_2 cannot (simultaneously) be matched to the pair (a, b), so we do not add an edge.

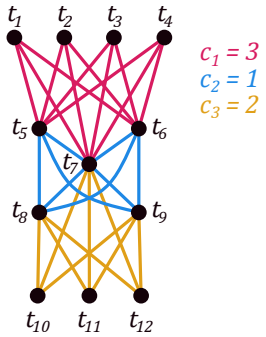
Doing this once for a pair from each of the m different equivalence classes yields G' and the weight and color functions w and c. When constructing a matching in G', the inclusion of an edge e with color i will correspond to matching the two task vertices at its endpoints to the same pair from P_i^* in G.

The last step of our reduction will be to define the color budgets and weight threshold. We will add m color budgets and set $c_i = |P_i^*|$ for $i \in [m]$. We simply set the weight threshold to d, which concludes the construction of the CCPM instance $(G', w, c, (c_i)_{i \in [m]}, d)$.

Since we have that $V(G') = T$, G' has $2n$ vertices. It also has $\mathcal{O}(m \cdot n^2)$ edges, for each of which the existence, color and weight can be determined in polynomial time. Hence, the reduction can be done in polynomial time. It remains to show that the reduction is correct, which we will do by showing that (G, f, d) is a YES-instance if and only if $(G', w, c, (c_i)_{i \in [m]}, d)$ is a YES-instance.



(a) A graph G belonging to an example PM instance. Each of the three equivalence classes P_1^*, P_2^* and P_3^* and their incident edges are indicated using a different color. Note that this is merely for visualization purposes and that no elements of a PM instance are actually colored.



(b) The graph G' obtained after reducing G to a CCPM instance. Note that edges in this graph are actually colored, so the colors in this image are not merely for visualization purposes. The edge weights are not included in the image, but the color budgets are.

Figure 6: An example PM instance reduced to a CCPM instance. The graph G has three equivalence classes of people pairs. For example, every pair $(a, b) \in P_1^*$ we have $N_G(a) = \{t_1, t_2, t_3, t_4\}$ and $N_G(b) = \{t_5, t_6, t_7\}$, meaning they belong to the same equivalence class. Each edge color in the CCPM instance corresponds to one of these three equivalence classes and the color budgets correspond to their sizes.

($\Rightarrow$)
Suppose that (G, f, d) is a YES-instance. Then there exists a matching M of size $|T|$ in G whose cost according to the definition on page 17 is $d^* \leq d$. We continue by constructing a color-constrained matching M' in G' of at most weight d^*.

For each $i \in [m]$, for each $(a, b) \in P_i^*$, let t_1 and t_2 be such that $\{a, t_1\}, \{b, t_2\} \in M$. These two task vertices exist for every pair (a, b), since G is balanced and M has size $|T|$, meaning that M is a perfect matching and in particular that a and b are both covered by M. We then add the i-colored edge $\{t_1, t_2\}$ to M'. By construction of G', this edge exists in G' and has weight at most $f(t_1, t_2)$. When we do this for every pair in P^*, it follows that our constructed edge set M' is indeed a perfect matching from the fact that M is a perfect matching. We also have that M' has at most weight d^*. Moreover, every color i is used exactly c_i times in M', making it a color-constrained matching. $(G', w, c, (c_i)_{i\in[m]}, d)$ is therefore a YES-instance.

($\Leftarrow$)
Suppose that $(G', w, c, (c_i)_{i\in[m]}, d)$ is a YES-instance. Then a color-constrained matching M' exists in G' of weight $d^* \leq d$. We continue by constructing a perfect matching M in G with the same cost d^*.

M is constructed by doing the following for each $\{t_1, t_2\} \in M'$. First let the color of $\{t_1, t_2\}$ be i. Then we choose a pair (a, b) from P_i^*. The existence of $\{t_1, t_2\}$ in G' implies that t_1 and t_2 can be matched to the pair (a, b) in G at a cost of $w(\{t_1, t_2\})$. We add the two edges in G achieving this to M.

When doing this for every edge $\{t_1, t_2\} \in M'$, we must be careful to choose a different people pair $(a, b) \in P^*$ every time to ensure that M' remains a matching at every step. Since there are c_i edges of color i in M' and c_i pairs in P_i^* for $i \in [m]$, this is possible. Following the above procedure for every edge in M' therefore yields a perfect matching M in G with cost $d^* \leq d$. (G, f, d) is therefore a YES-instance. □

As mentioned earlier, the reduction to CCPM is interesting because it is a more fundamental formulation of the problem, allowing us to see that it generalizes other problems already known in literature. One of these problems is the previously introduced Weighted Perfect Matching problem (see Section 5.1), which is obtained from CCPM by setting $k = 1$. Since WPM is solvable in polynomial time, we can also draw the following conclusion from Lemma 6.3.

Corollary 6.4. *Every PM instance with $|\mathcal{C}_{P^*}| = 1$ can be reduced to an equivalent WPM instance in polynomial time. Hence, PM is solvable in polynomial time for instances in which $|\mathcal{C}_{P^*}| = 1$.*

Also note that complete bipartite graphs are a special case of the restriction to graphs with $|\mathcal{C}_{P^*}| = 1$, meaning that Proposition 5.1, where we show PM to be solvable in polynomial time on complete bipartite graphs, can also be seen as a corollary of Lemma 6.3.

Lemma 6.3 shows only one of two directions needed to prove Theorem 6.2. We continue with the second direction.

Lemma 6.5. *Every CCPM instance with k different edge colors can be reduced in polynomial time to an equivalent PM instance with $|\mathcal{C}_{P^*}| = k + 1$.*

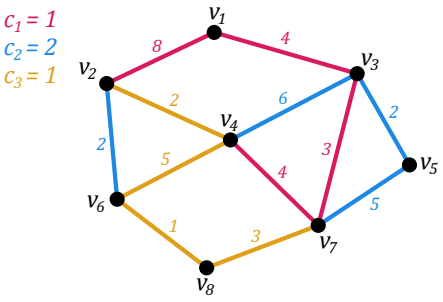
Proof. Let $(G, w, c, (c_i)_{i\in[k]}, d)$ be a CCPM instance, with a graph $G = (V, E)$, edge weight function $w : E \to \mathbb{R}+$, and edge coloring $c : E \to [k]$. Let $n := |V|$ and number the vertices in V as $v_1, v_2, \ldots, v_n$ arbitrarily. We assume w.l.o.g. that each pair of vertices is connected by at most one edge of each color. It is easy to see that it otherwise suffices to only keep the minimum weight edge of each color between each pair of vertices.

We reduce this CCPM instance to an equivalent BPM instance (G', f, d) with $|\mathcal{C}_{P^*}| = k + 1$. This suffices to show the statement as BPM is generalized by PM. First the reduction itself is explained, after which the intuition behind it is explained. See also Figure 7 for the reduction of an example input.

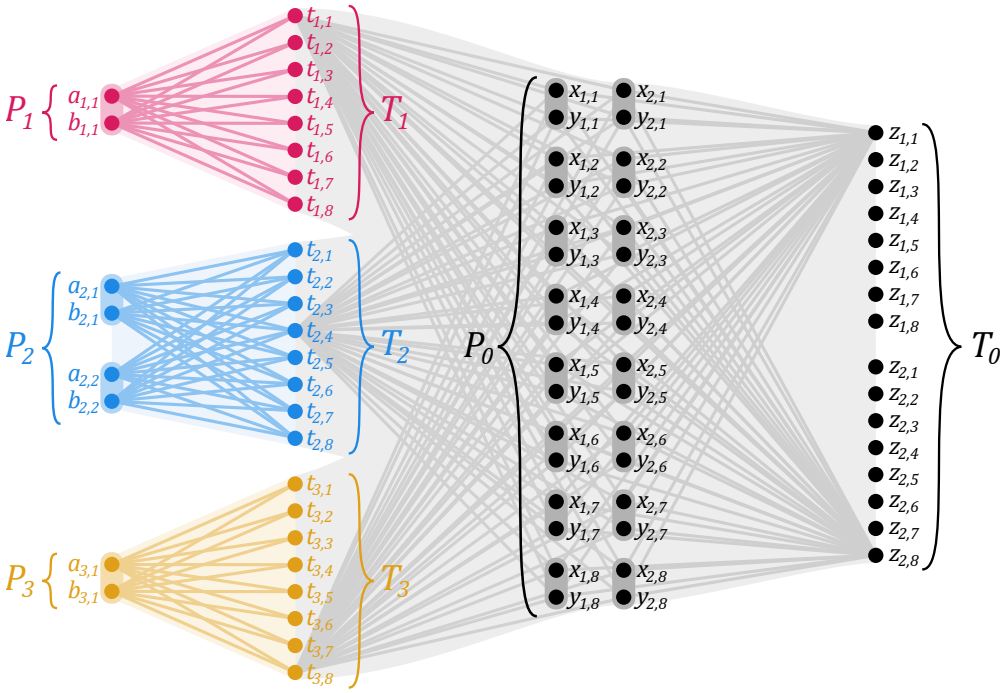
To construct the graph $G' = (P + T, E')$ for the PM instance, we start by defining the sets P and T. We will define each of these sets as the union of $k+1$ sets of vertices such that $P = P_0 \cup P_1 \cup P_2 \cup \ldots \cup P_k$ and $T = T_0 \cup T_1 \cup T_2 \cup \ldots \cup T_k$. Firstly, for each color $i \in [k]$ and each vertex $v_j \in V$, we introduce a vertex $t_{i,j}$ to the set T_i. Additionally, we add vertices $z_{1,j}, z_{2,j}, \ldots z_{k-1,j}$ to T_0 for each vertex $v_j \in V$ to conclude the construction of T.

We continue by adding c_i pairs $(a_{i,1}, b_{i,1}), (a_{i,2}, b_{i,2}), \ldots, (a_{i,c_i}, b_{i,c_i})$ to set P_i for $i \in [k]$. Additionally, we add pairs $(x_{i,j}, y_{i,j})$ to P_0 for $i \in [k-1]$ and $j \in [n]$.

To conclude the construction of G', we need to add the edges to it. For every $i \in [k]$, we add edges from every vertex in P_i to every vertex in T_i. This will make every subgraph $G'[P_i \cup T_i]$ for $i \in [k]$ a complete bipartite graph. Lastly, we connect every vertex in P_0 to every vertex in T. Figure 7 shows this part of the reduction for an example instance.



(a) An example input graph G to the CCPM problem with $k = 3$ colors.



(b) The resulting graph G' after reducing the example input to a PM instance. Note: for the sake of visualization, not all edges are shown. All vertices in P_0 should be connected to all vertices in $T_0 \cup T_1 \cup T_2 \cup T_3$. Also note that the adjacency information of G and the edge weights in it are encoded in the cost function f and therefore not visible in this drawing of G'.

Figure 7: An example CCPM instance reduced to a PM instance.

We can verify that G' is balanced by computing the sizes of P and T. We have that $|T_i| = n$ for $i \in [k]$ and $|T_0| = n(k-1)$, so $|T| = n(2k-1)$. We have that $|P_i| = 2 \cdot c_i$ for $i \in [k]$ and we assume that $\sum_{i=1}^{k} c_i = n/2$ as every YES instance for CCPM must satisfy this property. Hence, we obtain $|P_1 \cup P_2 \cup \ldots \cup P_k| = n$. We have $|P_0| = 2n(k-1)$, so we get $|P| = n(2k-1)$. Since $|P| = |T|$, we conclude that G' is indeed balanced.

Now we continue the reduction by defining the function $f : T \times T \to \mathbb{R}+$. In our construction of f, we want to exclude some pairs of vertices from T to be paired together at all and we indicate this in the cost function as it evaluating to ∞. Note that it suffices to take a sufficiently large finite value here instead, such as $d+1$, but we use the value ∞ to highlight that no valid solution will pair two vertices from T together that have this contribution to the total cost. With this in mind, we continue by defining f and we do so in three parts.

Firstly, for two vertices $t_{i,j}, t_{i',j'}$ from $T_1, T_2, \ldots, T_k$, we define f to be given by:

$$f(t_{i,j}, t_{i',j'}) = \begin{cases} w(\{v_j, v_{j'}\}) & \text{if } i = i' \text{ and } \{v_j, v_{j'}\} \in E \text{ with } c(\{v_j, v_{j'}\}) = i \\ \infty & \text{otherwise} \end{cases}$$

Recall that we assumed there to only be at most one edge of each color between any given pair of vertices in G, meaning that the weight of such an edge is uniquely defined.

Secondly, for a vertex $t_{i,j}$ from $T_1, T_2, \ldots, T_k$ and a vertex $z_{i',j'} \in T_0$ we define f to be given by:

$$f(t_{i,j}, z_{i',j'}) = f(z_{i',j'}, t_{i,j}) = \begin{cases} 0 & \text{if } j = j' \\ \infty & \text{otherwise} \end{cases}$$

Lastly, for any two vertices $z_{i,j}, z_{i',j'} \in T_0$, we define f to be given by:

$$f(z_{i,j}, z_{i',j'}) = \infty.$$

Finally, we simply reuse the threshold d to obtain the BPM instance (G', f, d). The idea is now that a matching in G can be encoded as a matching in $G'[P_1 \cup P_2 \cup \ldots \cup P_k \cup T_1 \cup T_2 \cup \ldots \cup T_k]$. This is because G' is constructed in such a way that if there is an edge in G between some $v_1, v_2 \in V$ with color i and weight $w(\{v_1, v_2\})$, then $t_{i,1}$ and $t_{i,2}$ could be matched in G' to the same pair in P_i at cost $w(\{v_1, v_2\})$. By encoding all the edges of a matching in G like this in G', a matching is obtained that covers all the vertices in $P_1 \cup P_2 \cup \ldots \cup P_k$ and n vertices in $T_1 \cup T_2 \cup \ldots \cup T_k$. The remaining $n(k-1)$ vertices of the latter set can, together with the $n(k-1)$ vertices from T_0, be matched to P_0 at a cost of 0.

Before proving the correctness more rigorously, we briefly argue that G' indeed has $k+1$ equivalence classes of people pairs. All vertices from P_i have the same neighborhood for any fixed $i \in [k]$ and the vertices in P_0 also all have the same neighborhoods as one another. Also, all pairs are formed from two vertices belonging to the same set from the partition $P_0 \cup P_1 \cup P_2 \cup \ldots \cup P_k$, so these $k+1$ sets themselves form the equivalence classes.

It remains to prove that $(G, w, c, (c_i)_{i \in [k]}, d)$ is a YES-instance if and only if (G', f, d) is a YES-instance.

($\Rightarrow$)
Suppose that $(G, w, c, (c_i)_{i\in[k]}, d)$ is a YES-instance. Then there exists a color-constrained matching M in G with weight $d^* \leq d$. We continue by constructing a perfect matching M' in G' with cost $d^* \leq d$.

For every edge $\{v_j, v_{j'}\} \in M$ with color $c(\{v_j, v_{j'}\}) = i$ we add the edges $\{a_{i,\ell}, t_{i,j}\}$ and $\{b_{i,\ell}, t_{i,j'}\}$ to M' for some ℓ such that the pair $(a_{i,\ell}, b_{i,\ell})$ has not yet been covered by M'. Note that there are c_i edges of color i and there are c_i pairs in P_i, meaning that there are enough pairs in P_i to realize this for every edge of color i. By the construction of G', the graph $G'[P_i \cup T_i]$ is complete bipartite, meaning that the edges $\{a_{i,\ell}, t_{i,j}\}$ and $\{b_{i,\ell}, t_{i,j'}\}$ exist. Then by the construction of f, the inclusion of these two edges makes the pair $(a_{i,\ell}, b_{i,\ell})$ contribute a value of $w(\{v_j, v_{j'}\})$ to the cost of M'.

After doing this for every edge in M, we have arrived at a (not yet perfect) matching M' with cost d^* in G'. All the vertices in $P_1 \cup P_2 \cup \ldots \cup P_k$ have been covered by M' and none of the vertices in P_0 have been covered. For every vertex $v_j \in V$, we have that exactly one of the k vertices $t_{i,j}$ has been covered by M', leaving the other $k-1$ of them still unmatched. The $k-1$ vertices $z_{1,j}, z_{2,j}, \ldots, z_{k-1,j} \in T_0$ are also still unmatched. For every $j \in [n]$ we match every $t_{i,j}$ that is not yet covered by M' to a vertex $x_{\ell,j} \in P^*$. Since there are equally many vertices of either type, we can do this in such a way that M' remains a matching. Lastly, we add the edges $\{y_{\ell,j}, z_{\ell,j}\}$ to M' for $\ell \in [k-1]$ and $j \in [n]$. Since every pair $(x_{\ell,j}, y_{\ell,j})$ gets matched to vertices $t_{i,j}$ and $z_{i',j'}$ respectively with $j = j'$ in this way, all these pair contribute 0 to the cost.

M' is now a perfect matching in G' and its cost is $d^* \leq d$, so (G', f, d) is a YES-instance.

($\Leftarrow$)
Suppose that (G', f, d) is a YES-instance. Then there exists a perfect matching M' in G' with cost $d^* \leq d$. Since M' is a perfect matching, in particular all vertices from T_0 are matched. They are only adjacent to vertices in P_0, so each vertex from T_0 will be matched by M' to a vertex from P_0. We also know that the cost of M' is at most $d < \infty$, so every pair (x, y) from P_0 in which one vertex is matched to a vertex $z_{\ell,j} \in T_0$ will have its other vertex matched to some vertex $t_{i,j'} \in T_1 \cup T_2 \cup \ldots \cup T_k$ with $j = j'$.

Note that because $|P_0| = 2 \cdot |T_0|$, this applies to every pair in P_0. I.e.: we have that every pair from P_0 has one of its vertices matched to some $z_{\ell,j} \in T_0$ and the other to some $t_{i,j'}$ with $j = j'$. This has a contribution of 0 to the cost. And because for every $j \in [n]$ there are $k-1$ vertices $z_{\ell,j} \in T_0$, we also get that exactly $k-1$ of the k vertices $t_{i,j} \in T_1 \cup T_2 \cup \ldots \cup T_k$ are matched to a vertex in P_0 by M'. Hence, exactly one vertex $t_{i,j}$ is matched to a vertex in $P_1 \cup P_2 \cup \ldots \cup P_k$ by M' for $j \in [n]$. It is this insight that allows us to now construct a matching M in G with the desired properties.

Let (a, b) be a pair from P_i and let $t_{i,j}$ and $t_{i,j'}$ be the two vertices matched to them by M'. Because we know that the cost of M' is at most $d < \infty$, there must exist an edge in G between v_j and $v_{j'}$ with color i and that $w(\{v_j, v_{j'}\}) = f(t_{i,j}, t_{i,j'})$. We add this edge to M. Doing this for every pair in $P_1 \cup P_2 \cup \ldots \cup P_k$ concludes the construction of M.

Because P_0 had a contribution of 0 to the cost of M', we must have that $P_1 \cup P_2 \cup \ldots \cup P_k$ has a contribution of d^* to the cost of M'. Since we added an edge to M for every pair in $P_1 \cup P_2 \cup \ldots \cup P_k$ whose weight was equal to the contribution of the pair to the cost of M' we must have that $w(M) = d^* < d$.

Next, we show that M is indeed a perfect matching. Recall that for every $j \in [n]$ exactly one vertex $t_{i,j}$ was matched to a vertex in $P_1 \cup P_2 \cup \ldots \cup P_k$. So in our construction of M, we have added exactly one incident edge for each $v_j \in V$, meaning that M is a perfect matching. Finally, we know that M contains exactly c_i edges of color i, because $|P_i| = 2 \cdot c_i$ for $i \in [k]$ and one edge of color i was added for each pair in P_i. □

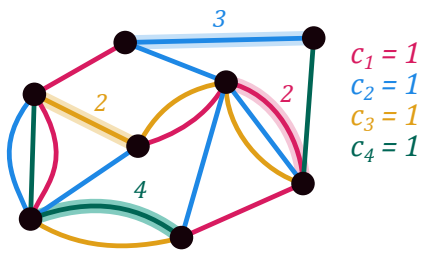
By proving Lemmas 6.3 and 6.5, we have shown the correctness of Theorem 6.2. By showing the equivalence of PM and CCPM, we allow ourselves to find results for CCPM, knowing that in many cases they can easily be translated to results for PM. Since CCPM is more similar to many other matching problems, this problem will prove easier to work with. We defined CCPM on multigraphs however, while many other matching problems are defined on simple graphs. The following observation is therefore a useful one to make.

Observation 6.6. *Every CCPM instance with a multigraph can be reduced in polynomial time to an equivalent CCPM instance with a simple graph.*

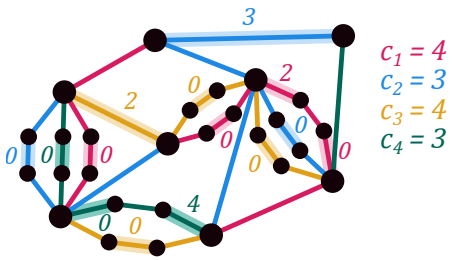
Proof. Clearly, CCPM on multigraphs is a generalization of CCPM restricted to simple graphs, so it suffices to argue that any instance to the problem with a multigraph can be reduced to an instance with a simple graph. Such a reduction can easily be made by making the following two modifications for every edge e whose endpoints are also connected by another edge.

- Let u and v be the endpoints of e. Then we remove e from the graph and replace it with a path (u, x_e, y_e, v) in which all three edges have color $c(e)$. We give the newly introduced edge $\{u, x_e\}$ a weight of $w(e)$, while giving the other two a weight of 0.
- We increase the budget for the color $c(e)$ by 1.

See Figure 8 for an example. A color-constrained matching in the multigraph can now easily be translated to a color-constrained matching in the simple graph and vice versa. This is also indicated in Figure 8 for an example solution. Including an edge e with endpoints u and v in a matching in the multigraph can be seen as equivalent to including the edges $\{u, x_e\}$ and $\{y_e, v\}$ in the simple graph: both options cover the vertices u and v at a weight of $w(e)$, while the latter option provides one more edge of color $c(e)$ than the first option. Likewise, *not* including an edge e with endpoints u and v in the multigraph can be seen as equivalent to including the edge $\{x_e, y_e\}$ in the simple graph: both options leave u and v uncovered at a weight of 0, while the latter option again provides one more edge of color $c(e)$ than the first option. □



(a) An example CCPM instance with a multigraph as input. The highlighted edges indicate a color-constrained perfect matching. For the sake of visualization, only the weights of the edges in this matching are included in the image.



(b) The resulting CCPM instance with a simple graph when transforming the instance from Figure 8a using the procedure described above. The highlighted edges indicate the color-constrained perfect matching that is equivalent to the one in Figure 8a. Again, only the weights of these edges are shown in the image.

Figure 8: Example of a multigraph CCPM instance reduced to an equivalent CCPM instance with a simple graph.

Now that we have proven an equivalence relation between PM and CCPM, it is possible to prove statements about the complexity of PM via CCPM. Corollary 6.4 is an example of this as it relied on the observation that CCPM generalizes WPM. Another problem that is generalized by CCPM is the Exact Matching problem (EM), which can be obtained by setting $k = 2$ and ignoring edge weights.

Exact Matching (EM)	
Given:	A simple graph G in which each edge is colored either red or blue and a budget r.
Objective:	Determine whether a perfect matching exists in G with exactly r red edges.

The problem was posed in 1982 by Papadimitriou and Yannakakis [32], after which Mulmuley et al. showed in 1987 that it can be solved in polynomial time by a *randomized* algorithm [30]. Despite much attention in literature [11, 18, 12, 34, 37, 41], no deterministic polynomial time algorithm is known, making EM one of the few problems for which a randomized polynomial algorithm is known, but not a deterministic one. Since a deterministic FPT algorithm or even a deterministic XP algorithm for CCPM parameterized by k would yield a deterministic polynomial time algorithm for EM, such an algorithm would solve this long standing open problem. We present no such algorithm, but the ingredients of the randomized polynomial time algorithm for EM can be combined into an algorithm that solves CCPM in randomized XP time w.r.t. k if the edge weights are integer valued, bounded by a constant and stored in unary.

6.1.1 Randomized XP algorithm

Mulmuley et al. have given RP algorithms for the Exact Matching problem and the Minimum Weight Perfect Matching problem [30]. The ideas behind them can be combined and extended to construct a randomized algorithm that solves CCPM parameterized by k in XP time when the edge weights are integer valued and bounded by a constant. The algorithm is constructed using the insights from Lemma 6.8, for which we first introduce the notion of the *modified Tutte matrix* of a graph. The modified Tutte matrix is defined for simple graphs with edge weights and edge colors. Its structure is similar to the (regular) Tutte matrix as defined in Definition 2.3 for simple, unweighted, uncolored graphs.

Definition 6.7 (Modified Tutte matrix)**.** *Let $G = (V, E)$ be a simple graph with vertices $V = \{v_1, v_2, \ldots, v_n\}$ and with edge weights as given by $w : E \to \mathbb{R}$ and edge colors as given by $c : E \to [k]$. We introduce indeterminates y_i for $i \in [k]$. The modified Tutte matrix B with entries $b_{i,j}$ is then given by:*

$$b_{i,j} = \begin{cases} 0 & \text{if } \{v_i, v_j\} \notin E \\ 2^{w(\{v_i,v_j\})} \cdot y_{c(\{v_i,v_j\})} & \text{if } \{v_i, v_j\} \in E \text{ and } j > i \\ -2^{w(\{v_i,v_j\})} \cdot y_{c(\{v_i,v_j\})} & \text{if } \{v_i, v_j\} \in E \text{ and } i > j \end{cases}$$

Using this definition, we can state Lemma 6.8, which states that information about color-constrained matchings in a graph can be obtained by investigating the Pfaffian of the modified Tutte matrix of the graph. The lemma and its proof rely heavily on the preliminaries presented in Section 2.1, as does Proposition 6.9 following afterwards.

Lemma 6.8. *Let $(G = (V, E), w, c, (c_i)_{i\in[k]}, d)$ be a CCPM instance such that G is a simple graph with $2n$ vertices and with only positive, integer edge weights. Let B be the modified Tutte matrix obtained from G as above. The Pfaffian of B is then a polynomial on the variables $y_1, y_2, \ldots, y_k$. Let λ be the coefficient for the term $\prod_{i=1}^{k} y_i^{c_i}$. We distinguish three cases.*

(a) If there is no color-constrained matching in G, then $\lambda = 0$.

Otherwise, let w_{min} denote the minimum weight of a color-constrained matching.

(b) If there are multiple color-constrained matchings with weight w_{min}, then either $\lambda = 0$ or the largest power of 2 which divides λ is at least $2^{w_{min}}$.

(c) If there is only one color-constrained matching with weight w_{min}, then the largest power of 2 which divides λ is $2^{w_{min}}$.

Proof. We first note that B is a skew-symmetric matrix of size $2n \times 2n$. Recall from Definition 2.5 that the Pfaffian of B with entries $b_{i,j}$, is then defined as

$$\mathrm{pf}(B) = \sum_{\alpha \in \Pi_{2n}} B_\alpha,$$

where

$$B_\alpha = \mathrm{sign}(\pi_\alpha) \cdot b_{i_1,j_1} \cdot b_{i_2,j_2} \cdot \ldots \cdot b_{i_n,j_n}.$$

For more details on this definition we refer to the preliminaries in Section 2.1.

Now note that $B_\alpha \neq 0$ if and only if $b_{i_1,j_1} \cdot b_{i_2,j_2} \cdot \ldots \cdot b_{i_n,j_n} \neq 0$. By construction of B, this property is equivalent to $\{v_{i_1}, v_{j_1}\}, \{v_{i_2}, v_{j_2}\}, \ldots, \{v_{i_n}, v_{j_n}\} \in E$. We call the terms B_α for which this holds *non-vanishing*. We see that non-vanishing terms correspond, by definition, to perfect matchings in G. If we let M_α denote the perfect matching corresponding to B_α, then we have

$$B_\alpha = \pm 2^{w(M_\alpha)} \cdot \prod_{e \in M_\alpha} y_{c(e)}.$$

This allows us to see that the only terms contributing to λ are the terms whose corresponding matching contains exactly c_i edges of color i for $i \in [k]$. After all, λ was defined to be the coefficient for the term in which each variable y_i has exponent c_i.

We continue writing an expression for λ. To this end, let Z denote the set of integers such that there exists a color-constrained matching with that integer as its weight. Finally, for $z \in Z$ we define $p(z)$ to be the difference between the number of positively and negatively contributing terms whose corresponding matching is color-constrained and has weight z:

$$p(z) := \sum_{\substack{\alpha \in \Pi \text{ s.t.:} \\ M_\alpha \text{ is color-constrained} \\ \text{and } w(M_\alpha) = z}} sign(\pi_\alpha).$$

This allows us to write

$$\lambda = \sum_{z \in Z} p(z) \cdot 2^z.$$

Clearly, if there are no color-constrained matchings in G, then $\lambda = 0$, proving case (a) of the statement. If there are color-constrained matchings in G, these terms could all cancel each other out, leaving us again with $\lambda = 0$. Otherwise, if $\lambda \neq 0$, then we claim that $2^{w_{min}}$ divides λ. This becomes easy to see after rewriting λ. Because $w_{min} = \min Z$, we can rewrite

$$\lambda = 2^{w_{min}} \cdot \sum_{z \in Z} p(z) \cdot 2^{z - w_{min}}$$

such that the exponent in the power of two in each term is still a non-negative integer. Hence, if $\lambda \neq 0$, then $2^{w_{min}}$ divides λ, which proves case (b) of the statement.

It remains to prove case (c). Since $p(w_{min}) = \pm 1$ in this case, we have that

$$\lambda = \pm 2^{w_{min}} + \sum_{z \in Z \setminus \{w_{min}\}} p(z) \cdot 2^z.$$

We know that $(\sum_{z \in Z \setminus \{w_{min}\}} p(z) \cdot 2^z)/2^{w_{min}}$ is even, because every element in $Z \setminus \{w_{min}\}$ is strictly larger than w_{min}. Moreover, this makes $\lambda/2^{w_{min}}$ odd, meaning that $\lambda \neq 0$ and $2^{w_{min}}$ is the largest power of 2 dividing λ. This proves the third and final case of the statement. □

This lemma indicates a strategy to solve CCPM for instances with integer edge weights and a unique optimal solution: obtain the modified Tutte matrix B from G, determine its Pfaffian and determine the coefficient λ from which the weight of the optimal solution can be derived. We modify this strategy to also work for instances in which the optimal solution is not necessarily unique. This is achieved by randomly altering the edge weights in the input graph. When done correctly, the answer to the CCPM instance remains unaltered and the isolating lemma ensures that there is a high probability ($\geq \frac{1}{2}$) that there is a unique optimal solution.

Proposition 6.9. *Let $(G, w, c, (c_i)_{i \in [k]}, d)$ be a CCPM instance, where G is a simple graph with n vertices in which each edge receives a positive integer weight of at most W. Then there exists a randomized algorithm that solves it in $n^{\mathcal{O}(k)} \cdot (n + W)^{\mathcal{O}(1)}$ time.*

Proof. We give a high-level overview of a procedure that satisfies the above requirements in Algorithm 6.

The algorithm starts by scaling and then randomly perturbing the edge weights. First, every edge weight is scaled by a factor nm. Since all edges started with an integer weight by assumption, any two perfect matchings that did not originally have the same weight differed by at least 1 before scaling and therefore differ by at least nm after scaling. Next, we add a random value from $[2m]$ to the weight of each edge to obtain an updated weight function w'. Then a perfect matching that originally had w-weight w_0 now has a w'-weight in the range $[nm \cdot w_0 + n, nm \cdot w_0 + nm]$.

Note that for different integer values for w_0, these intervals are disjoint. Since w was assumed to only assign integer values, the weights of all perfect matchings under weight function w are integer. Therefore, it holds for any two perfect matchings M_1 and M_2 in G that $w(M_1) > w(M_2) \implies w'(M_1) > w'(M_2)$.

In particular, this holds for color-constrained matchings. Suppose that such a matching exists in G and that the minimum w-weight of such a matching is w_{min}. Then the minimum w'-weight w'_{min} of a color-constrained matching must then be attained by one or more of the color-constrained matchings with w-weight w_{min}. It follows then that w'_{min} lies in the interval $[nm \cdot w_{min} + n, nm \cdot w_{min} + nm]$. Moreover, since the perturbations r_e are chosen uniformly and independently from $[2m]$ for each edge $e \in E$, it follows from

Algorithm 6 Randomized XP algorithm for CCPM parameterized by k

1: **function** CCPM-RXP($G = (V, E), w, c, (c_i)_{i \in [k]}, d$)
2: Let $n \leftarrow |V|/2$ and let $m \leftarrow |E|$.
3: For $e \in E$, let r_e be a random integer chosen uniformly and independently from $[2m]$.
4: Assign each $e \in E$ a new weight $w'(e) = nm \cdot w(e) + r_e$.
5:
6: Let B be the modified Tutte matrix of G using weights w' and colors c.
7: Let $y_1, y_2, \ldots, y_k$ be the indeterminates of B as in Definition 6.7.
8: **for** $y^* := (y_1^*, y_2^*, \ldots, y_k^*) \in \{0, 1, 2, \ldots, n\}^k$ **do**
9: Let B_{y^*} be the matrix obtained from B by replacing each y_i by y_i^*.
10: Compute $\sqrt{|B_{y^*}|}$ and store the point $(y^*, \sqrt{|B_{y^*}|})$.
11: **end for**
12: Determine $\sqrt{|B|}$ through interpolation from the $(n+1)^k$ points $(y^*, \sqrt{|B_{y^*}|})$. [23]
13: Let λ be the coefficient for $\prod_{i=1}^{k} y_i^{c_i}$ in $\sqrt{|B|}$.
14:
15: **if** $\lambda = 0$ **then return** NO.
16: Otherwise, let b be the exponent in the largest power of 2 that divides λ.
17: **if** $b \leq nm(d+1)$ **then return** YES.
18: Otherwise, **return** NO.
19: **end function**

the isolating lemma (see Lemma 2.2) that there is a unique color-constrained matching of minimum w'-weight with probability at least $\frac{1}{2}$.

This is a key insight in proving that Algorithm 6 is indeed a randomized algorithm to solve CCPM. To prove this, we must show that the algorithm always returns NO for NO-instances and that it returns YES with probability at least $\frac{1}{2}$ for YES-instances (see also Definition 2.7).

The algorithm now continues by constructing the modified Tutte matrix B and computing $\mathrm{pf}(B)$ on lines 6-12. More specifically, the algorithm computes $\sqrt{|B|}$, but recall from Section 2.1 that $\mathrm{pf}(B) = \sqrt{|B|}$ because B is skew-symmetric.

Although there exist efficient algorithms to compute the determinant of matrices with integer entries, B contains indeterminates, making the computation of its determinant, and in particular the square root thereof, non-trivial. B contains k different indeterminates and each entry in B contains exactly one of them as a linear term. If we recall the definition of the Pfaffian from Definition 2.5, we see that every term in $\mathrm{pf}(B)$ is a constant times the product of n entries of B. This means that $\mathrm{pf}(B)$ is a polynomial on k indeterminates in which the maximum degree of each indeterminate is at most n.

Rather than computing $|B|$ or $\sqrt{|B|} = \mathrm{pf}(B)$ directly, we therefore do so via interpolation. On lines 8-11, we iterate over $(n+1)^k$ possible assignments of the indeterminates

$y_1, y_2, \ldots, y_k$, together forming the k-dimensional integer lattice $\{0, 1, 2, \ldots, n\}^k$, spreading $n+1$ points wide in each dimension. The fact that these assignments together form such a lattice, mean that they uniquely identify $\sqrt{|B|}$ as this is a polynomial on k indeterminates that each have a degree of at most n [35]. Moreover, $\sqrt{|B|}$ can be determined from these points in $n^{\mathcal{O}(k)}\mathcal{O}(W)$ time on line 12, using e.g. an algorithm presented by Kapusta and Smarzewski [23].

After having computed $\sqrt{|B|}$ in this way, the coefficient λ for the term $\prod_{i=1}^{k} y_i^{c_i}$ in $\sqrt{|B|} = \mathrm{pf}(B)$ is determined and based on the value of λ, either YES or NO is returned.

- **Case 1:** $(G, w, c, (c_i)_{i\in[k]}, d)$ is a NO-instance.
  There are two possible reasons for an instance to be a NO-instance. Either the graph contains no color-constrained matchings or all color-constrained matchings have a weight greater than d.

  - **Case a:** G has no color-constrained matchings.
    In this case, Lemma 6.8 tells us that $\lambda = 0$, meaning that we return NO on line 15.

  - **Case b:** There do exist color-constrained matchings in G, but they all have a w-weight greater than d.
    Let w_{min} be the minimum w-weight of a color-constrained matching in G. As established, the minimum w'-weight w'_{min} must be attained by one or more of the same color-constrained matchings attaining the minimum w-weight. We therefore have $w'_{min} \in [nm \cdot w_{min} + n, nm \cdot w_{min} + nm]$. By assumption we have that $w_{min} > d$ and because all edge weights are integer, it follows that $w_{min} \geq d + 1$. Hence, we obtain $w'_{min} > nm \cdot w_{min} \geq nm(d+1)$.

    Lemma 6.8 tells us that we either have $\lambda = 0$ or that the exponent in the largest power of 2 dividing λ is at least w'_{min}. In the first case, the algorithm returns NO on line 15, in the latter case, the algorithm returns NO on line 18, because $w'_{min} > nm(d+1)$.

  In either case, Algorithm 6 returns NO if it is given a NO-instance as input.

- **Case 2:** $(G, w, c, (c_i)_{i\in[k]}, d)$ is a YES-instance.
  In this case, there exists a color-constrained matching in G with w-weight at most d. Let w_{min} be the minimum w-weight of a color-constrained matching in G, meaning that in particular $w_{min} \leq d$. Like above, by denoting the minimum w'-weight of a color-constrained matching in G by w'_{min}, we find that $w'_{min} \in [nm \cdot w_{min} + n, nm \cdot w_{min} + nm]$.

  Moreover, we established that with probability at least $\frac{1}{2}$ the minimum w'-weight is attained by just one unique color-constrained matching. In that case, Lemma 6.8 tells us that the exponent in the largest power of 2 dividing λ is w'_{min}. By assumption, we have $w_{min} \leq d$, meaning that $w'_{min} \leq nm \cdot w_{min} + nm \leq nm(d+1)$, leading the algorithm to return YES on line 17 in that case.

This concludes the correctness of the algorithm. We continue by determining the running-time of the algorithm, which can be split into three parts.

In the first part of the algorithm (lines 2–4), all edge weights are updated. Each edge receives a weight of at most $nmW + 2m$, so this can be done in $(n + W)^{\mathcal{O}(1)}$ time.

In the second part of the algorithm (lines 6–14), the modified Tutte matrix B is constructed and the square root of its determinant is determined, after which the coefficient of a term in this expression is evaluated. Each entry in the modified Tutte matrix is at least -2^{nmW+2m} times an indeterminate and at most 2^{nmW+2m} times an indeterminate. If the edge weights are encoded in binary, each entry can be computed in $(n + W)^{\mathcal{O}(1)}$ time. Since there is a quadratic number of terms to compute, the entire modified Tutte matrix can be computed in $(n + W)^{\mathcal{O}(1)}$ time as well.

Next, the Pfaffian of B is computed. To do this via the interpolation procedure described above, we iterate over $(n+1)^k$ integer assignments y^* of the indeterminates $y_1, y_2, \ldots, y_k$ on lines 8-11. We let B_{y^*} denote the matrix obtained from B by using the assignment y^* on its indeterminates. Now for each such assignment y^*, B_{y^*} is an integer matrix with values in the range $[-n2^{nmW+2m}, n2^{nmW+2m}]$. When encoding these values in binary, the determinant of B and the square root thereof can then be computed in $(n+W)^{\mathcal{O}(1)}$ time, e.g. using Gaussian elimination. The loop on lines 8-11 therefore consists of $n^{\mathcal{O}(k)}$ iterations that each take $(n + W)^{\mathcal{O}(1)}$ time, giving the entire loop a running time of $n^{Oh(k)}(n + W)^{\mathcal{O}(1)}$.

The next step is then on line 12 to determine $\mathrm{pf}(B) = \sqrt{|B|}$ via interpolation from the $(n+1)^k$ assignments for which we already evaluated it. As established, this can be done in $n^{\mathcal{O}(k)}\mathcal{O}(W)$ time [23]. Since the resulting expression for $\mathrm{pf}(B)$ is a polynomial with $n^{O(k)}$ terms, the coefficient for a given term can be determined in $n^{O(k)}$ time, which is in particular true for λ on line 13.

In the third and final part of the algorithm (lines 15–18), the value of λ is evaluated and an answer is returned based on this value. The only non-trivial computation made in this part is to determine the largest power of 2 that divides λ, which can be done in $(n + W)^{\mathcal{O}(1)}$ time e.g. by simply iterating through powers of 2 in ascending order, until one is found that does not divide λ.

This gives the algorithm a total running time of $n^{\mathcal{O}(k)} \cdot (n + W)^{\mathcal{O}(1)}$. □

Before moving on to the next parameterization, we make the final remark that the algorithm above is a randomized XP algorithm for CCPM parameterized by the number of edge colors k, whenever the edge weights are integers that are bounded by a *constant* value W and which are stored in unary.

6.2 PM parameterized by the number of equivalence classes of tasks

After having discussed one set of equivalence classes, namely those of the people pairs, we turn our attention to the second set of equivalence classes: the equivalence classes of task vertices.

We can define a similar notion of equivalence for the task vertices of the graph. These vertices are however not only defined by their adjacency in the graph, but also by their behaviour in the cost function f. Both these properties therefore need to be taken into account to define equivalence classes for the task vertices.

Definition 6.10 (Equivalence class of tasks)**.** *Let (G, c, f, d) be a PM instance. Then $\mathcal{C}_T \subseteq 2^T$ is the unique partition of T such that two vertices $t_1, t_2 \in T$ are partitioned into the same set if and only if the following four properties hold:*

- $N_G(t_1) = N_G(t_2)$,
- $f(t_1, t_3) = f(t_2, t_3)$ *for all* $t_3 \in T$,
- $f(t_3, t_1) = f(t_3, t_2)$ *for all* $t_3 \in T$ *and*
- $f(t_1, t_2) = f(t_2, t_1)$.

We then call t_1 and t_2 equivalent or indistinguishable and the elements of $\mathcal{C}_T$ are referred to as the equivalence classes of the task vertices of G.

We call task vertices belonging to the same equivalence class *equivalent* or *indistinguishable*. Such task vertices follow a similar concept of "indistinguishability", namely that two equivalent vertices may be swapped in a matching without affecting its validity or cost. Consider two equivalent task vertices $t_1, t_2 \in T$ and some matching M that matches them to $p_1, p_2 \in P$ respectively. By the first property in the above definition t_1 and t_2 have identical neighborhoods, meaning that M can be altered by matching p_2 to t_1 and p_1 to t_2 instead. Moreover, by the other three properties, the resulting matching will have the same cost as M.

We can make the following two observations relating $|\mathcal{C}_{P^*}|$ and $|\mathcal{C}_T|$.

Observation 6.11. *Every PM instance has $|\mathcal{C}_{P^*}| \leq 4^{|\mathcal{C}_T|}$.*

Proof. Since the neighborhoods of task vertices in the same class $C \in \mathcal{C}_T$ are identical, each people vertex can only be connected to either all or none of the vertices in a given $C \in \mathcal{C}_T$. For each vertex $p \in P$, the neighborhood is then completely defined by the subset of equivalence classes of task vertices it has edges to. This gives $2^{|\mathcal{C}_T|}$ different possible neighborhoods for vertices in P. Every pair in P^* is an ordered combination of two such vertices, for which $(2^{|\mathcal{C}_T|})^2$ different possibilities exist. Hence, we can conclude that $|\mathcal{C}_{P^*}| \leq 4^{|\mathcal{C}_T|}$. □

Observation 6.12. *$|\mathcal{C}_T|$ can be arbitrarily large compared to $|\mathcal{C}_{P^*}|$ in a PM instance.*

Proof. To see that $|\mathcal{C}_T|$ can be arbitrarily large compared to $|\mathcal{C}_{P^*}|$, consider a complete bipartite graph and a cost function which assigns different values for every combination of two task vertices. Such an input would have $|\mathcal{C}_T| = |T|$, but $|\mathcal{C}_{P^*}| = 1$. □

Observation 6.11, combined with Proposition 6.9, allows us to conclude that PM parameterized by $|\mathcal{C}_T|$ admits a randomized XP algorithm for instances in which c and f are integer valued and bounded by a constant. However, Observation 6.12 also suggests that it could be possible to construct algorithms with more favorable running times for this parameterization, since $|\mathcal{C}_T|$ could be very large even when $|\mathcal{C}_{P^*}|$ is small. Indeed, we find that PM parameterized by $|\mathcal{C}_T|$ is FPT.

This result is achieved by rewriting the problem as an Integer Linear Program (ILP), with a limited number of variables. Lenstra proved in 1983 that the problem of deciding whether an ILP admits a feasible solution can be solved in FPT time when parameterized by the number of variables [25]. The formulation from the book Parameterized Algorithms by Cygan et al. [9] (Theorem 6.5) nicely summarizes the result in a way that is easily applicable for our purposes.

Lemma 6.13. [9, 25] *For an ILP of size L with p variables, it can be determined using $p^{\mathcal{O}(p)} \cdot \mathcal{O}(L)$ arithmetic operations whether there exist a feasible solution.*

To prove now that BPM is FPT when parameterized by $|\mathcal{C}_T|$, we write an ILP that correctly models BPM, where the number of variables depends only on $|\mathcal{C}_T|$.

Proposition 6.14. *PM parameterized by $|\mathcal{C}_T|$ is FPT.*

Proof. To prove the statement, we prove that BPM is FPT w.r.t. $|\mathcal{C}_T|$. Lemma 3.4 shows how to reduce PM to BPM by adding additional task vertices and this is done in such a way that every newly added task vertex belongs to the same equivalence class: these newly added task vertices have identical neighborhoods and no distinction is made between these vertices in the cost function. This means that reducing PM to BPM in this way increases $|\mathcal{C}_T|$ by at most 1. It follows then that not only BPM, but also PM is FPT w.r.t. $|\mathcal{C}_T|$.

Next, we prove the fixed parameter tractability of BPM. To do so, we first show how to write a BPM instance as an ILP which has a valid assignment of its variables if and only if the BPM instance is a YES-instance. Next, we argue that the obtained ILP indeed correctly models BPM. Finally, we use Lemma 6.13 to determine the time complexity of solving the program.

Now let $(G = (P+T, E), f, d)$ be a BPM instance. Since G is always balanced in a BPM instance, a matching of size $|T|$ is always a perfect matching, meaning that in particular all vertices from P are matched. So in any valid solution, every people pair has both its vertices matched to a task vertex. Our ILP will use variables $x_{P^*_\ell, T_i, T_j}$ denoting for

some solution the number of people pairs in $P^*_\ell \in \mathcal{C}_{P^*}$ in which the first vertex gets matched to a vertex from $T_i \in \mathcal{C}_T$ and the second vertex gets matched to a vertex from $T_j \in \mathcal{C}_T$. Furthermore, we let f_{T_i,T_j} denote the cost that each such pair contributes in a solution. Note that by definition of $\mathcal{C}_T$, this value is the same for every such pair and it can be obtained directly from the cost function f. Now we can define the following ILP to represent our BPM instance.

$$x_{P^*_\ell,T_i,T_j} \in \mathbb{Z} \qquad \forall P^*_\ell \in \mathcal{C}_{P^*}, T_i, T_j \in \mathcal{C}_T$$

$$x_{P^*_\ell,T_i,T_j} \geq 0 \qquad \forall P^*_\ell \in \mathcal{C}_{P^*}, T_i, T_j \in \mathcal{C}_T$$

$$x_{P^*_\ell,T_i,T_j} = 0 \qquad \begin{array}{r} \forall P^*_\ell \in \mathcal{C}_{P^*},\ T_i, T_j \in \mathcal{C}_T \\ \text{s.t. for } (a,b) \in P^*_\ell,\ t \in T_i,\ u \in T_j \\ \text{we have } \{a,t\} \notin E \vee \{b,u\} \notin E \end{array}$$

$$\sum_{\substack{P^*_\ell \in \mathcal{C}_{P^*} \\ T_i,T_j \in \mathcal{C}_T \\ \text{s.t.: } T_i \neq T_j \text{ and} \\ T_i = T' \vee T_j = T'}} x_{P^*_\ell,T_i,T_j} + 2 \cdot \sum_{P^*_\ell \in \mathcal{C}_{P^*}} x_{P^*_\ell,T',T'} = |T'| \qquad \forall T' \in \mathcal{C}_T$$

$$\sum_{T_i,T_j \in \mathcal{C}_T} x_{P^*_\ell,T_i,T_j} = |P^*_\ell| \qquad \forall P^*_\ell \in \mathcal{C}_{P^*}$$

$$\sum_{\substack{P^*_\ell \in \mathcal{C}_{P^*} \\ T_i,T_j \in \mathcal{C}_T}} f_{T_i,T_j} \cdot x_{P^*_\ell,T_i,T_j} \leq d$$

We continue by arguing that this ILP correctly models BPM. Recall that the intention is to let $x_{P^*_\ell,T_i,T_j}$ denote the number of people pairs in P^*_ℓ in which the first vertex gets matched to a vertex from $T_i \in \mathcal{C}_T$ and the second vertex gets matched to a vertex from $T_j \in \mathcal{C}_T$ in a solution. In a valid solution, all these variables are therefore assigned a non-negative integer value. The first two sets of constraints ensure this property by enforcing the integrality and non-negativity of the variables respectively.

Under these first two sets of constraints, the variables can each be seen as indicating the presence of $2 \cdot x_{P^*_\ell,T_i,T_j}$ edges in a solution. For a solution including the edges $\{a,t\}$ and $\{b,u\}$ for $(a,b) \in P^*_\ell$ and $t \in T_i$, $u \in T_j$, the value of $x_{P^*_\ell,T_i,T_j}$ would be increased by 1. The third set of constraints ensures that we do not assign positive values to variables corresponding to situations where these two edges are not both present.

The fourth set of constraints ensures that exactly $|T'|$ task vertices from any given equivalence class $T' \in \mathcal{C}_T$ are used to be matched to people vertices in a solution. For T_i, T_j with $T_i \neq T_j$, each variable $x_{P^*_\ell,T_i,T_j}$ indicates the presence of $x_{P^*_\ell,T_i,T_j}$ edges incident

with a vertex of T_i and $x_{P_\ell^*,T_i,T_j}$ edges incident with a vertex of T_j. For variables in which these two sets of equivalence classes are equal, say $x_{P_\ell^*,T',T'}$, the variable indicates the presence of $2 \cdot x_{P_\ell^*,T',T'}$ edges incident with T'. The summation of these quantities for every variable in which a given equivalence class T' occurs, must then equal $|T'|$. The fourth set of constraints hence correctly models this requirement.

Recall that task vertices from the same equivalence class are indistinguishable from one another and may be swapped freely with one another in a solution without affecting its validity or cost. So if we enforce every valid solution to include exactly $|T'|$ edges to be incident with vertices from $|T'|$, we can in particular choose these edges so that every vertex in $|T'|$ is incident with exactly one of them.

The fifth set of constraints similarly ensures that the vertices of exactly $|P_\ell^*|$ people pairs from any given equivalence class $P_\ell^* \in \mathcal{C}_{P^*}$ are used to be matched to task vertices in a solution. Each variable $x_{P_\ell^*,T_i,T_j}$ indicates the presence of two edges covering exactly $x_{P_\ell^*,T_i,T_j}$ pairs from the equivalence class P_ℓ^*. The fifth set of constraint hence correctly models this constraint. Again, from the fact that people pairs from the same equivalence class are indistinguishable, we can conclude that in particular a solution under this constraint can be constructed in which every people pair in P_ℓ^* has both its edges covered by exactly one edge.

When combining the first five set of constraints, we see that every feasible solution to the ILP corresponds to a perfect matching in G. For such a matching to be a valid solution to the BPM instance, its cost must be at most d. This property is taken care of by the final constraint. Let the variables $x_{P_\ell^*,T_i,T_j}$ together indicate a valid solution to the BPM instance. Note that every people pair in which the first vertex gets matched to a vertex from $T_i \in \mathcal{C}_T$ and the second vertex gets matched to a vertex from $T_j \in \mathcal{C}_T$ contributes f_{T_i,T_j} to the total cost of a solution. If there are $x_{P_\ell^*,T_i,T_j}$ such pairs, they make a combined contribution of $f_{T_i,T_j} \cdot x_{P_\ell^*,T_i,T_j}$. Then by summing this value for every $P_\ell^* \in \mathcal{C}_{P^*}$ and $T_i, T_j \in \mathcal{C}_T$, we obtain the total cost of the solution indicated by the variables. The final constraint in the ILP requires this cost to be at most d, meaning that any feasible assignment of the variables in the program correspond to a valid solution in the BPM instance.

Running time. Now that we have established that the program correctly models BPM it remains to determine the complexity of constructing and solving the program. First we remark that all constants appearing in the program also appear in the BPM instance. Either as the size of a set or as function value of the cost function f. The storage needed for these values is then of course polynomial in the size of the original input. Encoding these values in binary uses space logarithmic in their value, so for the sake of the running time analysis we assume w.l.o.g. that these values are all $\mathcal{O}(2^n)$, where $n = |T|/2 = |P|/2$ as defined on page 1. This allows for easier notation as we can use this assumption to write the size of the BPM instance as $n^{\mathcal{O}(1)}$.

It also means that the ILP representation of BPM is of size $n^{\mathcal{O}(1)}$ and can be obtained in $n^{\mathcal{O}(1)}$ time for every instance, since every individual constraint can be determined in polynomial time and there are polynomially many constraints.

Next, we determine the number of variables in the program. Since variables are defined for a combination of one equivalence class of people pairs and two equivalence classes of tasks, the program contains $\mathcal{O}(|\mathcal{C}_{P^*}| \cdot |\mathcal{C}_T|^2)$ variables. From Observation 6.11, we know that $|\mathcal{C}_{P^*}| \leq 4^{|\mathcal{C}_T|}$, so the bound on the number of variables can be rewritten as $\mathcal{O}(4^{|\mathcal{C}_T|})$.

Lemma 6.13 now tells us that the feasibility of the ILP can be determined using $2^{|\mathcal{C}_T|^{\mathcal{O}(1)}} \cdot n^{\mathcal{O}(1)}$ arithmetic operations. Since all operations are applied to values that can be stored in $n^{\mathcal{O}(1)}$ space, it follows that the feasibility of the ILP can be determined in FPT time. By extension, BPM and PM parameterized by $|\mathcal{C}_T|$ are in FPT. □

7 Conclusion and discussion

We introduced the Paired Matching problem (PM) and explored its (parameterized) complexity. In Section 3, we showed PM to be NP-complete and in Sections 4–6 we gave parameterized algorithms for a multitude of parameterizations of the problem. Most notably, we gave a randomized XP algorithm for a restriction of PM parameterized by $|\mathcal{C}_{P^*}|$ in Section 6.1 (see Proposition 6.9). This parameter was motivated by the observation that realistic inputs typically attain very low values for it. Although, the presented randomized XP algorithm is likely to be too slow for the intended practical application, Corollary 6.4 shows that PM can be solved in polynomial time on a restricted class of inputs, namely those with $|\mathcal{C}_{P^*}| = 1$.

Additionally, by showing that PM can be represented as the more naturally formulated matching problem CCPM (see Theorem 6.2), we related our newly introduced matching problem to existing literature on e.g. the Exact Matching problem (EM). It also allows us to put our open questions into the context of other known open problems.

Most importantly, we have two open questions regarding PM parameterized by $|\mathcal{C}_{P^*}|$, which could both analogously be posed for CCPM parameterized by the number of different edge colors.

(1) Does a deterministic XP algorithm exist for these problems?

(2) Does a randomized FPT algorithm exist for these problems?

Algorithms of either type would provide a theoretical improvement over our randomized XP algorithm. The first by removing the need for randomness and the second by an improvement in running time guarantees. An answer to the first question however would also resolve the long-standing open question of whether EM is in P. The same does not hold for the second question, as neither proving nor disproving the existence of a randomized FPT algorithm for the two problems would directly imply EM to be in P or not.

The existence of a randomized FPT algorithm could for example be proven via improvements to our algorithm presented in Proposition 6.9. Another possible route to take would be to provide a reduction from CCPM to EM such that the size of the resulting EM instance is bounded by $g(k) \cdot n^{\mathcal{O}(1)}$ for some computable function $g : \mathbb{N} \to \mathbb{N}$. Since EM is known to be in RP, this would yield a randomized FPT algorithm for CCPM parameterized by k or equivalently for PM parameterized by $|\mathcal{C}_{P^*}|$.

One could also attempt to disprove the existence of a randomized FPT algorithm for these two problems. Convincing evidence for this could for example be obtained by a parameterized reduction from a different parameterized problem for which such an algorithm has already been proven to be unlikely to exist.

References

[1] A.E. Andreev, E.E.F. Clementi, and J.D.P. Rolim. "Hitting Sets Derandomize BPP". In: *Automata, Languages and Programming, 23rd International Colloquium*. Ed. by F.M. Heide and Burkhard M. Vol. 1099. Lecture Notes in Computer Science. Springer, 1996, pp. 357–368. DOI: 10.1007/3-540-61440-0_142.

[2] A. Anwar and A.N. Mahmood. "Anomaly detection in electric network database of smart grid: Graph matching approach". In: *Electric Power Systems Research* 133 (2016), pp. 51–62. DOI: 10.1016/j.epsr.2015.12.006.

[3] A.A. Bertossi, P. Carraresi, and G. Gallo. "On some matching problems arising in vehicle scheduling models". In: *Networks* 17.3 (1987), pp. 271–281. DOI: 10.1002/net.3230170303.

[4] H.L. Bodlaender, B.M.P. Jansen, and S. Kratsch. "Kernel bounds for path and cycle problems". In: *Theoretical Computer Science* 511 (2013), pp. 117–136. DOI: 10.1016/j.tcs.2012.09.006.

[5] J.A. Bondy and U.S.R. Murty. *Graph Theory with Applications*. Macmillan Education UK, 1976. DOI: 10.1007/978-1-349-03521-2.

[6] J. Cheng, H. Saigo, and P. Baldi. "Large-scale prediction of disulphide bridges using kernel methods, two-dimensional recursive neural networks, and weighted graph matching". In: *Proteins: Structure, Function, and Bioinformatics* 62.3 (2006), pp. 617–629. DOI: 10.1002/prot.20787.

[7] D. Conte, P. Foggia, C. Sansone, and M. Vento. "Thirty years of graph matching in pattern recognition". In: *International journal of pattern recognition and artificial intelligence* 18.03 (2004), pp. 265–298. DOI: 10.1142/s0218001404003228.

[8] S.A. Cook. "The Complexity of Theorem-Proving Procedures". In: *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing*. Ed. by M.A. Harrison, R.B. Banerji, and J.D. Ullman. ACM, 1971, pp. 151–158. DOI: 10.1145/800157.805047.

[9] M. Cygan, F.V. Fomin, L. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized Algorithms*. Springer, 2015. DOI: 10.1007/978-3-319-21275-3.

[10] J. Edmonds. "Paths, trees, and flowers". In: *Canadian Journal of mathematics* 17 (1965), pp. 449–467. DOI: 10.4153/cjm-1965-045-4.

[11] N. El Maalouly. "Exact Matching: Algorithms and Related Problems". In: *Computing Research Repository* 2203.13899 (2022). arXiv: 2203.13899.

[12] N. El Maalouly and R. Steiner. "Exact Matching in Graphs of Bounded Independence Number". In: *Computing Research Repository* 2202.11988 (2022). arXiv: 2202.11988.

[13] M.R. Fellows, C. Knauer, N. Nishimura, P. Ragde, F.A. Rosamond, U. Stege, D.M. Thilikos, and S. Whitesides. "Faster Fixed-Parameter Tractable Algorithms for Matching and Packing Problems". In: *Algorithmica* 52.2 (2008), pp. 167–176. DOI: 10.1007/s00453-007-9146-y.

[14] H.N. Gabow. "Data Structures for Weighted Matching and Extensions to b-matching and f-factors". In: *ACM Transactions on Algorithms* 14.3 (2018), 39:1–39:80. DOI: 10.1145/3183369.

[15] Z. Galil. "Efficient Algorithms for Finding Maximum Matching in Graphs". In: *ACM Computing Surveys* 18.1 (1986), pp. 23–38. DOI: 10.1145/6462.6502.

[16] C.D. Godsil. *Algebraic combinatorics*. Chapman and Hall mathematics series. Chapman and Hall, 1993. DOI: 10.1201/9781315137131.

[17] S. Gupta, S. Roy, S. Saurabh, and M. Zehavi. "Quadratic Vertex Kernel for Rainbow Matching". In: *Algorithmica* 82.4 (2020), pp. 881–897. DOI: 10.1007/s00453-019-00618-0.

[18] R. Gurjar, A. Korwar, J. Messner, S. Straub, and T. Thierauf. "Planarizing Gadgets for Perfect Matching Do Not Exist". In: *ACM Transactions on Computation Theory* 8.4 (2016), 14:1–14:15. DOI: 10.1145/2934310.

[19] J.E. Hopcroft and R.M. Karp. "An $n^{5/2}$ Algorithm for Maximum Matchings in Bipartite Graphs". In: *SIAM Journal on Computing* 2.4 (1973), pp. 225–231. DOI: 10.1137/0202019.

[20] R. Impagliazzo and A. Wigderson. "P = BPP if E Requires Exponential Circuits: Derandomizing the XOR Lemma". In: *Proceedings of the Twenty-Ninth Annual ACM Symposium on the Theory of Computing*. Ed. by F.T. Leighton and P.W. Shor. ACM, 1997, pp. 220–229. DOI: 10.1145/258533.258590.

[21] S.M. Johnson. "Generation of permutations by adjacent transposition". In: *Mathematics of computation* 17.83 (1963), pp. 282–285. DOI: 10.1090/s0025-5718-1963-0159764-2.

[22] M. Kano and X. Li. "Monochromatic and Heterochromatic Subgraphs in Edge-Colored Graphs - A Survey". In: *Graphs and Combinatorics* 24.4 (2008), pp. 237–263. DOI: 10.1007/s00373-008-0789-5.

[23] J. Kapusta and R. Smarzewski. "Fast algorithms for multivariate interpolation and evaluation at special points". In: *Journal of Complexity* 25.4 (2009), pp. 332–338. DOI: 10.1016/j.jco.2009.02.001.

[24] D.E. Knuth. "Overlapping Pfaffians". In: *Electronic Journal of Combinatorics* 3.2 (1996). DOI: 10.37236/1263.

[25] H.W. Lenstra. "Integer Programming with a Fixed Number of Variables". In: *Mathematics of Operations Research* 8.4 (1983), pp. 538–548. DOI: 10.1287/moor.8.4.538.

[26] S. Micali and V.V. Vazirani. "An $O(\sqrt{|V|}|E|)$ Algorithm for Finding Maximum Matching in General Graphs". In: *21st Annual Symposium on Foundations of Computer Science*. IEEE Computer Society, 1980, pp. 17–27. DOI: 10.1109/SFCS.1980.12.

[27] J. Monnot. "The labeled perfect matching in bipartite graphs". In: *Information Processing Letters* 96.3 (2005), pp. 81–88. DOI: 10.1016/j.ipl.2005.06.009.

[28] H. Moser and S. Sikdar. "The parameterized complexity of the induced matching problem". In: *Discrete Applied Mathematics* 157.4 (2009), pp. 715–727. DOI: 10.1016/j.dam.2008.07.011.

[29] H. Moser and D.M. Thilikos. "Parameterized complexity of finding regular induced subgraphs". In: *Journal of Discrete Algorithms* 7.2 (2009), pp. 181–190. DOI: 10.1016/j.jda.2008.09.005.

[30] K. Mulmuley, U.V. Vazirani, and V.V. Vazirani. "Matching is as easy as matrix inversion". In: *Combinatorica* 7.1 (1987), pp. 105–113. DOI: 10.1007/BF02579206.

[31] N. Nisan and A. Wigderson. "Hardness vs Randomness". In: *Journal of Computer and System Sciences* 49.2 (1994), pp. 149–167. DOI: 10.1016/S0022-0000(05)80043-1.

[32] C.H. Papadimitriou and M. Yannakakis. "The complexity of restricted spanning tree problems". In: *Journal of the ACM* 29.2 (1982), pp. 285–309. DOI: 10.1145/322307.322309.

[33] W.H. Payne and F.M. Ives. "Combination Generators". In: *ACM Transactions on Mathematical Software* 5.2 (1979), pp. 163–172. DOI: 10.1145/355826.355830.

[34] I. Rusu. "Maximum weight edge-constrained matchings". In: *Discrete Applied Mathematics* 156.5 (2008), pp. 662–672. DOI: 10.1016/j.dam.2007.08.021.

[35] T. Sauer and Y. Xu. "On multivariate Lagrange interpolation". In: *Mathematics of computation* 64.211 (1995), pp. 1147–1170. DOI: 10.1090/s0025-5718-1995-1297477-5.

[36] M. Sipser. "A Complexity Theoretic Approach to Randomness". In: *Proceedings of the 15th Annual ACM Symposium on Theory of Computing*. Ed. by D.S. Johnson, R. Fagin, M.L. Fredman, D. Harel, R.M. Karp, N.A. Lynch, C.H. Papadimitriou, R.L. Rivest, W.L. Ruzzo, and J.I. Seiferas. ACM, 1983, pp. 330–335. DOI: 10.1145/800061.808762.

[37] G. Stamoulis. "Approximation Algorithms for Bounded Color Matchings via Convex Decompositions". In: *Mathematical Foundations of Computer Science 2014*. Ed. by E. Csuhaj-Varjú, M. Dietzfelbinger, and Z. Ésik. Vol. 8635. Lecture Notes in Computer Science. Springer, 2014, pp. 625–636. DOI: 10.1007/978-3-662-44465-8_53.

[38] L.J. Stockmeyer and V.V. Vazirani. "NP-Completeness of Some Generalizations of the Maximum Matching Problem". In: *Information Processing Letters* 15.1 (1982), pp. 14–19. DOI: 10.1016/0020-0190(82)90077-1.

[39] W.T. Tutte. "The Factorization of Linear Graphs". In: *Journal of the London Mathematical Society* s1-22.2 (1947), pp. 107–111. DOI: 10.1112/jlms/s1-22.2.107.

[40] M. Yannakakis. "Node- and Edge-Deletion NP-Complete Problems". In: *Proceedings of the 10th Annual ACM Symposium on Theory of Computing*. Ed. by R.J. Lipton, W.A. Burkhard, W.J. Savitch, E.P. Friedman, and A.V. Aho. ACM, 1978, pp. 253–264. DOI: 10.1145/800133.804355.

[41] R. Yuster. "Almost Exact Matchings". In: *Algorithmica* 63.1-2 (2012), pp. 39–50. DOI: 10.1007/s00453-011-9519-0.

[42] M. Zaslavskiy, F. Bach, and J. Vert. "Global alignment of protein–protein interaction networks by graph matching methods". In: *Bioinformatics* 25.12 (2009), pp. i259–1267. DOI: 10.1093/bioinformatics/btp196.