ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

The exact matching problem

Semester Project
Autumn Semester 2017-18

Author:
Kirtan Padh

Supervisors:
Ola Svensson

Abstract

A matching in a graph is a set of edges without common vertices. The matching problem is one of the most widely studied problems in complexity theory and combinatorial optimization. A *blue-red graph* is a graph with each of its edges coloured either red or blue. Given a parameter k, the exact matching problem asks for a perfect matching with exactly k red edges in a blue-red graph.
We survey the exact matching problem, one of the lesser understood problems in the class of matching problems. We also propose an additive approximation of the problem which gives a matching with $n-1$ edges with exactly k red edges for a graph with $2n$ vertices, or correctly asserts that no perfect matching with exactly k red edges exists in the graph.

Contents

1 Introduction **1**
1.1 Combinatorial optimization . . . 1
1.1.1 Network flows . . . 2
1.1.2 Spanning trees . . . 3
1.1.3 The travelling salesman problem . . . 4
1.1.4 Matching . . . 4
1.1.4.1 Maximal matching . . . 5
1.1.4.2 Perfect matching and maximum matching . . . 5

2 The exact matching problem **7**
2.1 The blue-red matching problem . . . 7
2.2 The power of randomness . . . 8
2.2.1 $P \stackrel{?}{=} BPP$ . . . 8
2.3 Perfect matching in RNC . . . 8
2.4 Exact matching in RNC . . . 11

3 An almost exact matching **12**
3.1 Open problems and further research . . . 14

Bibliography **15**

1 Introduction

Due to their wide and natural applicability optimization problems are studied in a wide range of fields. For example the problem of energy minimization of molecules in chemical systems, or the problem of allocating production of a product to different machines, with different capacities and operating cost, to meet production target at minimum cost. In physics, the problem of determining the ground state for a spin glass is an optimization problem. Optimization problems come up in almost every field of academic study and several aspects of daily life.

Optimization involves finding the maximum of minimum of an objective function over a specified domain. A *combinatorial optimization problem* is typically an optimization problem for which the domain is finite. Matching problems in graphs, including the exact matching problem fall under this category. We give a short introduction to combinatorial optimization by focusing on some specific types of problems, before moving on to looking at the matching problem in graphs in greater detail, the main subject of this report. The interested reader can refer to Papadimitriou and Steiglitz [21], Schrijver [23] or Korte and Vygen [15] for a more in-depth view into the subject.

1.1 Combinatorial optimization

Combinatorial optimization is relatively young as a coherent mathematical discipline. Before the advent of linear programming (LP), the field involved a number of independent lines of research, separately considering problems such as the shortest spanning tree, transportation and the travelling salesman problem. According to Schrijver [24], "linear programming forms the hinge in the history of combinatorial optimization". Since the formulation of linear programming and the development of the simplex algorithm by Dantzig [5], it has been used to tackle many combinatorial optimizations successfully.

We will assume basic familiarity with linear programming and give a short description of the following combinatorial optimization problems: network flows, spanning trees, the travelling salesman problem and matchings. For the sake of brevity we will not be perfectly rigorous in this section. The interested reader is referred to Schrijver [24] for an interesting history of some important problems in combinatorial optimization. $G = (V, E)$ always refers to an undirected graph with vertex set V and edge set E, and $G = (V, A)$ refers to a directed graph (digraph) with vertex set V and edge set A.

1.1.1 Network flows

The simplest and most common problem in network flows is maximum flow.

Maximum flow: We are given a digraph G with edge capacities $c : A(G) \to \mathbb{R}_+$ and two special vertices, one of which is specified as s (*source*) and the other as t (*sink*).

Definition 1.1 (Flow)**.** A flow is a function $f : A(G) \to \mathbb{R}_+$ such that $f(e) \leq c(e)\ \forall e \in E(G)$.

We say that a vertex v is balanced if the incoming flow at v is equal to the outgoing flow from v. Figure 1.1 gives an example of a maximum flow in a directed graph.

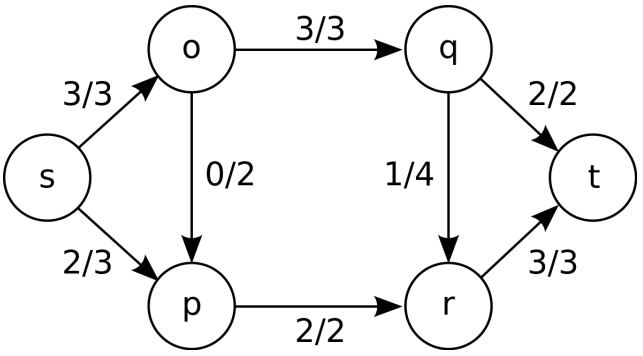

FIGURE 1.1: Example of max flow with flow/capacity

A directed graph is often called a network, hence the name network flow. A network can be used to model traffic in a roads, fluids in pipe networks, currents in an electrical circuit, or anything similar in which something travels through a network of nodes and is therefore of great practical importance.

We define $s-t$-flow as a flow in which every vertex except s and t is balanced, and the outgoing flow from s is equal to the incoming flow into t. The maximum flow problem is then to find an $s-t$-flow of maximum value in a digraph with given capacity constraints. The most common solution to this problem is the Edmonds-Karp algorithm, which is a greedy algorithm based on the Ford-Fulkerson method. The idea is to greedily find paths from s to t in which more flow could be sent, called augmenting paths, and send as much flow as we can through such a path until no such paths exist. This also leads to a proof of the max-flow min-cut theorem.

Minimum cost flow: Here we have a cost $k(e)$ associated with each edge, and the cost of sending flow $f(e)$ through the edge e is $f(e)k(e)$. The objective is to minimize the cost of the flow while sending at least a specified amount of flow from the source to the sink.

Multi-commodity flow: Here we have multiple sources and sinks, and several commodities, each of which is to flow from a given source to a given sink. We could have here a cost associated with each edge and define a problem of minimizing cost of the flow subject to a certain minimum flow akin to min-cost flow. This is called the minimum cost multi-commodity flow problem.

We could also have both an upper and a lower bound on the flow through each edge and define analogous problems as above.

1.1.2 Spanning trees

For simplicity we assume that G is connected for this subsection. A tree is a connected undirected graph with no cycles. A spanning tree of an undirected graph G is a subgraph of G that is a tree and contains all the vertices of G. Spanning tree problems have many applications. Some pathfinding algorithms such as Dijkstra's and the A^* algorithm use spanning trees as a subroutine. The simplest and most common problem in spanning trees is the minimum spanning tree problem. An example of a spanning tree is given the the figure 1.2.

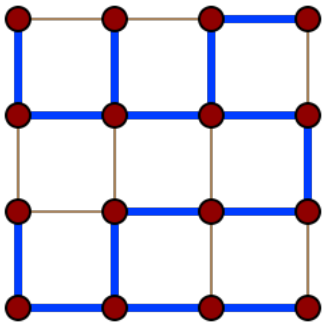

FIGURE 1.2: Example of a spanning tree on a grid graph

Minimum spanning tree problem
This is the problem of finding a spanning tree of minimum weight in a weighted graph. This has applications in minimizing the cost of power networks, piping networks, wiring connections, etc. This problems admits several efficient greedy algorithms, including Prim's algorithm [3] and Kruskal's algorithm [16].

Kruskal's algorithm simply proceeds by initializing an empty subgraph T of G and at each step adding the minimum weight edge which can be added to T without adding a cycle and update T to include that edge. Prim's algorithm works in quite the same way, but it adds the minimum weight edge which connects the current tree T to a vertex outside it.

Other optimization problems which have been studied on spanning trees include the maximum spanning tree; the minimum weight spanning tree that spans at least k vertices, called the k-minimum spanning tree problem; the spanning tree with the largest number of leaves; the spanning tree with the least value of maximum degree of its vertices, and many more. Interestingly, the existence of spanning trees in infinite graphs is known to be equivalent to the axiom of choice [25].

1.1.3 The travelling salesman problem

A Hamiltonian path in a graph is a path that visits every vertex exactly once. A Hamiltonian cycle is a Hamiltonian path that is also a cycle, meaning that the first and last vertices are the same.

Perhaps the most famous combinatorial optimization problem is the travelling salesman problem (TSP), and unlike the previous problems that we saw, is a computationally hard problem. TSP has many important real-world applications. Grötschel [11] showed that a direct application of TSP is in the drilling problem of printed circuit boards. It is also known to have applications in X-ray crystallography, vehicle routing and many simple situations which we might encounter on a daily basis. The problem can be formulated as that of finding a Hamiltonian cycle of least weight in a given weighted complete graph. The weights are assumed to follow the triangle inequality, which is to say that the distance between vertex i and j is no more than the sum of distances between the vertices i and k and vertices k and j for any vertex j. If we have no assumption on the weights, a reduction from deciding whether the graph is Hamiltonian shows that even approximating the problem to any constant factor is NP-hard. So the problem is relaxed to allow each city to be visited more than once. This happens to be equivalent to assuming that the weights in the graph satisfy triangle inequality.

The symmetric TSP: If the distance between vertices is symmetric, that is, the distance from i to j is the same as the distance from j to i, then it is called the symemtric TSP. This is the same as saying that the graph is undirected There is a classic algorithm from Christofides for this problem which gives a 3/2 approximation for the problem. The standard LP relaxation for the problem is conjectured to give an approximation ratio of 4/3.

The Asymmetric TSP (ATSP): Here the graph can be considered to be directed, so that the distance from i to j is not necessarily the same as the distance from j to i. The standard LP relaxation for the problem is known as the Held-Karp relaxation and is conjectured to give a small constant approximation ratio. However, in contrast to the symmetric TSP, we did not have a constant factor approximation for the problem until very recently, when Svensson et al. [27] proposed the first constant factor approximation for this problem.

A lot of open questions remain still in the study of TSP problems and the approximation ratios are yet far from settled.

1.1.4 Matching

A matching in a graph is a set of pairwise disjoint edges. We describe some algorithms for different kinds of matching problems.

1.1.4.1 Maximal matching

Definition 1.2 (Maximal matching)**.** A matching M of G is maximal if it is not a subset of any other matching of G.

So the maximal matching M has the property that if any edge not in M is added to M, it is not a matching any longer. Figure 1.3 shows some examples of maximal matching.

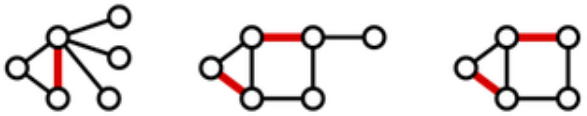

FIGURE 1.3: Some examples of maximal matching. The red edges are the ones in the matching

Maximal matching are easy to find using a greedy algorithms. We can initialize and empty list M and iterate over all the edges, adding the edge to M is it still remains a matching. Clearly, we end up with a maximal matching since we cannot add any more edges to it. However, no polynomial time algorithm is known for the minimum maximal matching, that is, a maximal matching with the least number of edges.

1.1.4.2 Perfect matching and maximum matching

Definition 1.3 (Perfect matching)**.** A matching M in G is perfect if it covers every vertex in G, that is, every vertex in G is an end point of one of the edges in M.

Definition 1.4 (Maximum matching)**.** A maximum matching in G is a matching with the largest number of edges.

Clearly, every maximum matching is also maximal, and every perfect matching is maximum. Figure 1.4 has some examples of maximum matchings and perfect matchings.

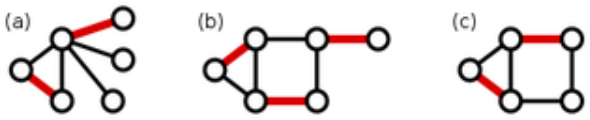

FIGURE 1.4: Some examples of maximal matching. The red edges are the ones in the matching. (b) is a perfect matching, while the other two are only maximum.

Bipartite graphs

Bipartite graphs have a simple characterization for the existence of perfect matchings in the form of Hall's marriage theorem. Given a graph $G = (V, E)$ and $S \subseteq V$, let $N(S)$ denote the neighborhood of S. So $N(S)$ contains every vertex in V that has an edge to one of the vertices in S.

Theorem 1.5 (Hall's condition)**.** *Let $G = (A \cup B, E)$ be an undirected bipartite graph. G has a perfect matching if and only if $|N(S)| \geq |S|$ for every $S \subseteq A$.*

The concept of *duality* is quite important in combinatorial optimization problems. It is the principle that some optimization problems may be looked at as one of two equivalent but distinct perspectives, the *primal* and the *dual*. A solution to the dual provides an upper or a lower bound to the optimal value of the primal and vice versa. The optimal values of the primal and the dual may or may not be equal. If the optimal values of the primal and dual are equal, *strong duality* is said to hold.

For the maximum matching problem in bipartite graphs, the dual problem is the vertex cover problem, another well know combinatorial optimization algorithm.

Definition 1.6 (Vertex cover)**.** A vertex cover C is a subset of vertices of a graph $G = (V, E)$ such that each vertex in V has an edge incident to at least one vertex in C.

The duality theorem is as follows

Theorem 1.7 (König 1931)**.** *For bipartite graphs, the size of the maximum matching is equal to the size of the smallest vertex cover.*

There are many different algorithms for finding a maximum matching in bipartite graphs. There is an algorithm which uses augmenting paths to iteratively build matching and end up with the maximum matching. It can also be formulated as a flow problem which can be solved efficiently. It can also be modeled as a linear program relaxation, which luckily happens to have an integral polytope. It can also be written as a matroid intersection problem. We skip the details of these algorithms to keep the report concise, but the interested reader can find them in these lecture notes from Prof. Ola Svensson's course. [26].

General graphs

Hall's condition does not suffice to give a characterization of perfect matchings in general graphs. Such a characterization was given by Tutte in 1947 and is as follows.

Theorem 1.8 (Tutte's theorem)**.** *Let $o(G)$ denote the number of odd sized components of G. A graph G has a perfect matching if and only if*

$$\forall S \subseteq V, \quad o(G \setminus S) \leq |S|$$

The proof can be found in Lovász and Plummer [17]. The first efficient algorithm for the problem was the so-called Blossoms algorithm by Edmonds et al. [8]. The matching is constructed by iteratively improving an initially empty matching along augmenting paths in the graph, albeit with some key differences when compared to the algorithm for bipartite graphs. This algorithm led to a linear programming formulation of the general matching problem, and therefore a polyhedral description of the matching polytope [7]. This description was considered a breakthrough in polyhedral combinatorics. Finally we come to the problem which is the focus of this report, the exact matching problem.

2 The exact matching problem

Given a graph $G = (V, E)$ with each edge coloured red or blue, the exact matching problem is to find a perfect matching in G with exactly k red edges or certify that it doesn't exist. The problem was introduced by Papadimitriou and Yannakakis [22] in 1982. The first efficient randomized algorithm for it was given by Mulmuley et al. [18] in 1987, who gave an RNC algorithm for the problem, along with other matching problems. We shall describe the algorithm later. However, unlike the other matching problems which which saw in the previous section, this problem has yet to yield an efficient deterministic algorithm. It is known to be polynomial time solvable only for specific families of graphs. The problem also has some practical applications in many different areas such as bus-driver scheduling, Statistical mechanics, DNA sequencing [2] and robust assignment problems [6].

Barahona and Pulleybank [1] gave a pseudo-polynomial algorithm for planar graphs in 1987, a result which was generalized by Gallucio and Loeble [9] in 1999 to give an analogous result for graphs which can be embedded into a fixed orientable surface. For complete and complete bipartite graphs, Karzanov [14] in 1987 gave a characterization to the exact matching problem. Karzanov also gave a polynomial time algorithm to construct an exact perfect matching. Yi et al. [29] in 2002 gave a simpler construction for complete bipartite graphs. Karzanov had separate characterization theorems for the complete bipartite and complete non-bipartite cases. Geerdes and Szabó [10] in 2011 gave a single characterization theorem for the separate cases considered by Karzanov. However, they did not give a unifying construction algorithm. Gurjar et al. [12] gave a unifying construction algorithm for both the cases.

2.1 The blue-red matching problem

Nomikos et al. [19] defined a generalization of the problem which they call the blue-red matching problem (BRM).

Definition 2.1 (Blue-red matching problem (BRM))**.** Given a graph G with edges coloured blue or red, and a parameter k, find a maximum matched containing at most k edges of each colour.

They give a randomized algorithm for the problem as well as a simple 1/2-approximation and a slightly more involved 3/4-approximation, in the sense that they give a maximal matching which has at most k edges of each colour and $3/4^{th}$ the number of edges in an optimal solution.

Before describing the randomized algorithm for the exact matching problem, we give a short overview of the power of randomness and the possibility of derandomization of some hard problems.

2.2 The power of randomness

There are many important problems that we know can efficiently be solved using randomness, but are nowhere near to giving an efficient deterministic algorithm for them. The hall of fame of such problems probably include generating primes, polynomial factoring and approximating the permanent. This leads to the question of whether there is some inherent power in the use of randomness that deterministic algorithms cannot give us. To formalize this we define the class *BPP*

Definition 2.2 (Class BPP)**.** A function $f : I \to I$ is in BPP is there exists a probabilistic algorithm $A(x)$ such that $Pr[A(x) \neq f(x)] \leq 1/3$.

BPP is the probabilistic equivalent of P.

2.2.1 $P \stackrel{?}{=} BPP$

Randomized algorithms for hard problems such as polynomial factoring might lead us to believe that there are problems in BPP which are not in P. However, Impagliazzo and Wigderson [13], to a surprise of many, proved that exponential circuit lower bounds on SAT imply that randomness can always be eliminated from algorithms without sacrificing efficiency. Formally

Theorem 2.3. *If SAT cannot be solved in circuits of size* $2^{\lambda(n)}$ *then* BPP = P.

Based on our beliefs about the lower bounds of NP-complete problems, this suggests that randomness does not in fact give us any extra power. For the exact matching problem, this means that in theory, it is very likely that it must have a deterministic polynomial time algorithm.

2.3 Perfect matching in RNC

The class NC can be thought of as problems that can be efficiently solved on a parallel computer, analogously to how P is considered to be the class of tractable problems. The class NC is the set of decision problems decidable in polylogarithmic time on a parallel computer with a polynomial number of processors. So a problem is in NC if there exist constants c and k such that it can be solved in time $\mathcal{O}(\log^c n)$ using $\mathcal{O}(nk)$ parallel processors. RNC is an extension of NC with access to randomness.

The *RNC* algorithm for the matching problems, including the perfect matching problem as well as the exact matching problem rely on the *isolation lemma*.

Theorem 2.4 (The isolation lemma)**.** *Let $S = \{e_1, e_2, \ldots, e_m\}$ and $S_1, \ldots, S_k \subseteq S$. Each e_i is given weight w_i, picked uniformly at random from $\{0, 1, \ldots, 2m-1\}$. We say that the weight of a subset S_j is $\sum_{e_i \in S_j} w_i$, denoted by $w(S_j)$. Then*

$$Pr[\textit{The minimum weight set in } \{S_1, \ldots, S_k\} \textit{ is unique}] \geq \frac{1}{2}$$

We will first describe the RNC algorithm for the perfect matching problem in general graphs. The algorithm for exact matching differs only in the set system that it considers for the isolation lemma and will be described easily once the algorithm for perfect matching has been described. We will need the Tutte matrix for the algorithm. We denote by a_{ij} the entry in the $(i,j)^{th}$ position of a matrix A, and by A_{ij} the minor obtained by removing the i^{th} row and j^{th} column of A. $|A|$ is the determinant of the matrix A and its adjoint is $Adj(A)$.

Definition 2.5 (Tutte matrix)**.** For a graph $G = (V, E)$ with $|V| = n$, let D denote the adjacency matrix of G. That is, $d_{ij} = 1$ if an only if $(i, j) \in E$. To obtain the Tutte matrix, for all entries such that $d_{ij} = d_{ji} = 1$, we replace them by indeterminates x_{ij} and $-x_{ij}$, with the positive one being above the diagonal. We leave the zeros alone. The matrix obtained after this substitutions in D is the Tutte matrix A of G.

This matrix characterizes perfect matchings in a graph as follows

Theorem 2.6 (Tutte [28])**.** *Let A be the Tutte matrix of a graph G. G has a perfect matching if and only if $|A| \neq 0$.*

Given a graph $G = (V, E)$ with $m = |E|$, we assign each edge (i, j) a weight w_{ij} chosen uniformly at random from $\{0, 1, \ldots, 2m-1\}$. Considering the set of edges to be the set of elements S and the set of perfect matchings in G to be the collection of subsets $S_1, \ldots, S_k \subseteq S$; by the isolation lemma G has a unique minimum weight perfect matching. We use this to construct a matrix B, by substituting each x_{ij} by $2^{w_{ij}}$ in the tutte matrix A of G. Then we get the following lemma as a result of Theorem 2.6.

Claim 2.7. *Let $G = (V, E)$ have a unique minimum weight perfect matching and its weight be w. Then $|B| \neq 0$ for B as defined above, and the highest power of 2 which divides $|B|$ is $2w$.*

Proof. Suppose $|V| = n$. Let σ be a permutation on $\{1, 2, \ldots n\}$. We define

$$v(\sigma) = \Pi_{i=1}^{n} b_{i\sigma(i)}$$

For a given σ, if $b_{i\sigma(i)} = 0$ for some i then of course $v(\sigma) = 0$. Therefore, $v(\sigma) \neq 0$ if and only if $(i, \sigma(i))$ is an edge in E for every i. By the definition of B, we have

$$|B| = \sum_{\sigma} sign(\sigma) v(\sigma)$$

Here $sign(\sigma)$ denotes the sign of the permutation. If $v(\sigma)$ is non-zero, we say that σ is non-zero. We borrow the terminology from the original paper and say that the *trail* of a non-zero σ is the set of edges $\{(i, \sigma(i)) : i \in [n]\}$. Each vertex i has two edges incident to it in every trail, $(i, \sigma(i))$ and $(\sigma^{-1}(i), i)$, possibly with repetition. So every trail is then a union of cycles and isolated edges. Note that the trail for the permutations corresponding to a perfect matching M is the matching M itself. Moreover, the terms corresponding to permutations whose trails have an odd cycle in them will be cancelled, due to two different terms with opposing signs corresponding to the direction of the permutation on the odd cycle.

We have $v(\sigma) = (-1)^{n/2}2^{2w}$ corresponding to the σ whose trail is the unique minimum weight matching. We claim that all the other terms in $|B|$ which are not cancelled out have higher powers of 2. This is certainly true for other perfect matchings, since w is the weight of the unique minimum weight perfect matching. Furthermore, we observe that each even cycle can be decomposed into two matchings matchings. So every trail that does not contain odd cycles can be written as the union of the trails of two distinct perfect matchings, say M_1 and M_2. Clearly, we must have $w(M_1) \geq w$ and $w(M_2) \geq w$. For such a trail, we have $|v(\sigma)| = 2^{w(M_1)+w(M_2)} > 2^{2w}$. This proves our claim, and shows that the highest power of 2 which divides $|B|$ is 2^{2w}. □

This gives us the weight of the perfect matching, but it still remains to actually find the perfect matching parallely. The following claim helps us to do that.

Claim 2.8. *The edge (i, j) belongs to the minimum weight perfect matching if and only if*

$$\frac{|B_{ij}|2^{w_{ij}}}{2^{2w}} \text{ is odd.}$$

Proof. Note that the numerator in the term is simply the sum of $v(\sigma)sign(v)$ over permutations whose trail has (i, j), that is $j = \sigma(j)$. We already know that this sum contains only trails which are either perfect matchings or can be written as a union of perfect matchings. If (i, j) belongs to the minimum weight perfect matching, the sum in the numerator has exactly one term with modulus 2^{2w} and all the other ones are higher powers of 2, and therefore the ratio with 2^{2w} is odd. In the other case all terms have a power greater than two as seen in the previous proof, and the ratio is then even. □

This naturally gives an efficient parallel algorithm for the perfect matching problem. The non-trivial steps in the algorithm are the computation of the determinant and the adjoint of B. This can be done in *RNC* by Pan [20]. Therefore the algorithm is *RNC*. As the proof shows, the algorithm will output all the edges which are in the min weight perfect matching, which exists with probability 1/2 by the isolation theorem. Therefore the algorithm has a success rate of 1/2.

Algorithm 1 Perfect matching

1: **procedure** FINDMATCH(G)
2: Assign weights to edges uniformly at random from $\{0, 1, \ldots, 2m-1\}$
3: Compute $|B|$ and get the value w of the minimum weight perfect matching
4: Compute the Adjoint of B, to get all the values of $|B_{ij}|$
5: Check in parallel for each edge the condition in Claim 2.8 to see if the edge is in the minimum weight perfect matchings.
6: **end procedure**

Szabó

2.4 Exact matching in RNC

The set system now consists of all perfect matchings with exactly k red edges. Assume that polynomially bound weights w_e are given to the edges $e \in E$ of G and here is a unique minimum weight perfect matching as with k red edges. Let E' be the set of red edges in G.

In the Tutte matrix, substitute x_e by 2^{w_e} if $e \in E - E'$ and by $2^{w_e}y$ if $e \in E'$. Let the resulting skew-symmetric matrix be B. The dimension n of the matrix B must be even if there exists a perfect matching in the graph. We then use the well known identity that for an even-dimensional skew symmetric matrix B [4],

$$|B| = Pf^2(B)$$

where $Pf(B)$ is the Pfaffian of B. We can get $Pf(B)$ from B by computing the square root of $|B|$ using a fast parallel algorithm and interpolating. The power of 2 in the coefficient of y^k is the weight of the matching we are looking for.

3 An almost exact matching

We propose an algorithm which ,in some sense goes as close to giving an exact perfect matching as we possibly can. Given a graph $G = (V, E)$ with each edge coloured either red or blue, and a parameter k, we return a matching with $n-1$ edges having exactly k red edges, or correctly assert that there exists no perfect matching with exactly k red edges. We found later that this has already been proved previously Yuster [30].

Our claim is the following

Theorem 3.1 (An almost exact matching)**.** *We are given a graph $G = (V, E)$ with each edge coloured either red or blue, and an integer parameter k. We have $|V| = 2n$ and $|E| = m$. Then we either certify that G has no perfect matching with exactly k red edges or return a matching of size $n-1$ with exactly k red edges.*

The idea of the proof is to find two perfect matchings in G, one having more than k red edges and one having less than k red edges, and combine them intelligently to obtain a matching as required in the theorem.

Before we give the proof we prove the following claim, which will be crucial to the proof.

Claim 3.2. *We are given an even cycle $C = (V, E)$ with each coloured red or blue. Let $|V| = 2n$ and therefore $|E| = 2n$. We consider it as a union of two perfect matchings M_1 and M_2. Suppose M_1 has k_1 red edges and M_2 has k_2 red edges and $k_1 < k < k_2$. Them we can find a matching maximal in C with exactly k red edges and size $n-1$.*

Proof. We order the edges of each matching separately, by a cyclic clockwise ordering, choosing any edge arbitrarily to be the first one for each matching. A set of contiguous edges refers to contiguity in the sense of this ordering. We observe that we can assume k_1 to be greater than zero, because if k_1 is zero, this means that M_1 has only blue edges. In this case we can choose a set of contiguous edges from M_2 so that there are a total of k red edges in this set and union it with every edge from M_1 we can choose while maintaining a maximal matching. This will leave only two vertices unmatched and therefore give a maximal matching with the required properties. It is always possible to choose such a contiguous set of edges in M_2 because it is known to have more than k red edges.

So now we have $0 < k_1 < k < k_2$. We start by initializing a maximal matching M of size $|E|/2 - 1$ with at least k red edges. The initialization is as follows

$$M = \{\text{A red e edge in } M_1\} \cup \{\text{All edges from } M_2 \text{ which can be added to e to make M maximal}\}$$

An example of the initialization is given in Figure 3.1. The algorithm then is to update M iteratively

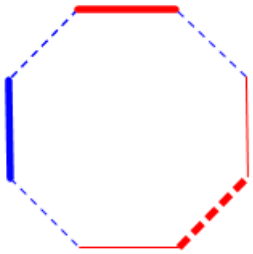

FIGURE 3.1: M_1 is shown by the dotted edges and M_2 is shown by the solid edges. The thickened edges are the ones which are selected.

by adding to it an edge of M_1 which is contiguous to the edges of M_1 which are already in M and remove appropriately an edge of M_2 from M to still maintain M to be a maximal matching. We do this iteratively until we have an M with the properties we need. An example of the iteration is given in Figure 3.2. The correctness of the algorithm relies on the following observations.

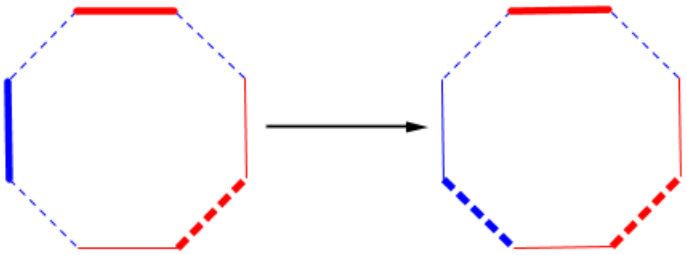

FIGURE 3.2: M_1 is shown by the dotted edges and M_2 is shown by the solid edges. The thickened edges are the selected ones.

Observation 1: The initial M has at least k red edges. This is because it has 1 red edge from M_1 and at least $k_2 - 2$ red edges from M_2, and therefore at least $k_2 - 1$ red edges. The observation follows from the fact that $k_2 > k$.

Observation 2: The number of red edges in M changes by at most one in each iteration. This is easy to see since we are only adding one and removing one edge in each iteration.

Observation 3: If we do enough iterations, the number of red edges in M will be at most k at some point. The limiting case is when we are left with at most one edge from M_2, and therefore have at most $k_1 + 1$ red edges in M. Since $k_1 < k$, we have $k_1 + 1 \geq k$.

The proof follows from the fact that the number of red edges in M_2 go from being at least k to at most k while changing by at most one in each iteration, and therefore must hit exactly k after some iteration. □

We can now give the proof of the main theorem.

Proof of Theorem 3.1 We start by finding perfect matchings M_r and M_b in G, where M_r is the perfect matching of G with the most number of red edges and M_b is the perfect matching with the most number of blue edges. Note that M_r or M_b might not be unique but it does not matter for us. We can get M_r simply by setting the weight of each red edge to be 2 and the weight of each blue edge to be 1 and finding the maximum weight perfect matching in this weighted graph. Similarly we can also get M_b.

Now if the number of red edges in M_r is less than k or the number of blue edges in M_b is less than $n - k$ then we cannot have a perfect matching with exactly k red edges and we return that there exists no perfect matching with exactly k red edges. In the other case, we look at the symmetric difference D of M_r and M_b and use it to construct the required matching.

We note that the symmetric difference D of two perfect matchings is simply a union of disjoint even cycles. The algorithm is then simply as follows: The correctness follows from the fact that

Algorithm 2 Almost exact matching

```
1: procedure AlmostFindMatch(G, k)
2:     Initialize M = M_b
3:     Order even cycles in D as l
4:     while M has less than k red edges do
5:         Invert the next even cycle in l to update M
6:     end while
7:     Use claim 3.2 on the cycle at which the number of red edges in M jumps over k, with the
   appropriate parameters
8: end procedure
```

there must be some cycle on whose inversion the number of red edges in M goes over k, because the limiting case is that if we invert all cycles, M will become M_r and that has more than k red edges. Let C be the cycle at which this jump happens in the algorithm. Then we freeze all the other cycles except C to be as they are at this point in M. C follows the conditions of Claim 3.2, in that the number of red edges we need from C lies in between the number of red edges in either matching of C as it is the first cycle whose inversion takes the number of red edges above k. So we can find an appropriate maximal matching of C to add to M along with the edges from the rest of the cycles which were already in M at this point, to get M to have the required properties. This completes the proof. □

3.1 Open problems and further research

Of course, the biggest open problem here is to show that the exact matching problem is in P. It would also be interesting to have different randomized algorithms for the problem.

Bibliography

[1] F. Barahona and W. R. Pulleyblank. Exact arborescences, matchings and cycles. *Discrete Applied Mathematics*, 16(2):91–99, 1987.

[2] J. Błażewicz, P. Formanowicz, M. Kasprzak, P. Schuurman, and G. J. Woeginger. A polynomial time equivalence between dna sequencing and the exact perfect matching problem. *Discrete Optimization*, 4(2):154–162, 2007.

[3] O. Borůvka. O jistém problému minimálním (about a certain minimum problem). *Práce moravské přírodovědecké společnosti v Brne*, 111(3):37–58, 1926.

[4] A. Cayley. Sur les déterminants gauches.(suite du mémoire t. xxxii. p. 119). *Journal für die reine und angewandte Mathematik*, 38:93–96, 1849.

[5] G. B. Dantzig. *Origins of the simplex method*. ACM, 1990.

[6] V. G. Deı, G. J. Woeginger, et al. On the robust assignment problem under a fixed number of cost scenarios. *Operations Research Letters*, 34(2):175–179, 2006.

[7] J. Edmonds. Maximum matching and a polyhedron with 0, 1-vertices. *Journal of Research of the National Bureau of Standards B*, 69(125-130):55–56, 1965.

[8] J. Edmonds. Paths, trees, and flowers. *Canadian Journal of mathematics*, 17(3):449–467, 1965.

[9] A. Galluccio and M. Loebl. On the theory of pfaffian orientations. i. perfect matchings and permanents. *Electron. J. combin*, 6(1):R6, 1999.

[10] H.-F. Geerdes and J. Szabó. A unified proof for karzanov's exact matching theorem. Technical Report QP-2011-02, Egerváry Research Group, Budapest, 2011. `www.cs.elte.hu/egres`.

[11] M. Grötschel, M. Jünger, and G. Reinelt. Optimal control of plotting and drilling machines: a case study. *Mathematical Methods of Operations Research*, 35(1):61–84, 1991.

[12] R. Gurjar, A. Korwar, J. Messner, and T. Thierauf. Exact perfect matching in complete graphs. In *Electronic Colloquium on Computational Complexity (ECCC)*, volume 20, page 112, 2013.

[13] R. Impagliazzo and A. Wigderson. P= bpp unless e has sub-exponential circuits: Derandomizing the xor lemma (preliminary version). *In Proceedings of the 29th STOC*, pages 220–229, 1996.

[14] A. Karzanov. Maximum matching of given weight in complete and complete bipartite graphs. *Cybernetics and Systems Analysis*, 23(1):8–13, 1987.

[15] B. Korte and J. Vygen. *Combinatorial optimization*, volume 2. Springer, 2012.

[16] J. B. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical society*, 7(1):48–50, 1956.

[17] L. Lovász and M. D. Plummer. *Matching theory*, volume 367. American Mathematical Soc., 2009.

[18] K. Mulmuley, U. V. Vazirani, and V. V. Vazirani. Matching is as easy as matrix inversion. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 345–354. ACM, 1987.

[19] C. Nomikos, A. Pagourtzis, and S. Zachos. Randomized and approximation algorithms for blue-red matching. In *MFCS*, pages 715–725. Springer, 2007.

[20] V. Pan. Fast and efficient parallel algorithms for the exact inversion of integer matrices. In *Foundations of Software Technology and Theoretical Computer Science*, pages 504–521. Springer, 1985.

[21] C. H. Papadimitriou and K. Steiglitz. *Combinatorial optimization: algorithms and complexity*. Courier Corporation, 1998.

[22] C. H. Papadimitriou and M. Yannakakis. The complexity of restricted spanning tree problems. *Journal of the ACM (JACM)*, 29(2):285–309, 1982.

[23] A. Schrijver. *Combinatorial optimization: polyhedra and efficiency*, volume 24. Springer Science & Business Media, 2003.

[24] A. Schrijver. On the history of combinatorial optimization (till 1960). *Handbooks in operations research and management science*, 12:1–68, 2005.

[25] L. Soukup. Infinite combinatorics: from finite to infinite. In *Horizons of combinatorics*, pages 189–213. Springer, 2008.

[26] O. Svensson. Course lecture note: Topics in theoretical computer science (http://theory.epfl.ch/courses/topicstcs/index2015.html), 2015.

[27] O. Svensson, J. Tarnawski, and L. A. Végh. A constant-factor approximation algorithm for the asymmetric traveling salesman problem. *arXiv preprint arXiv:1708.04215*, 2017.

[28] W. T. Tutte. The factorization of linear graphs. *Journal of the London Mathematical Society*, 1(2):107–111, 1947.

[29] T. Yi, K. G. Murty, and C. Spera. Matchings in colored bipartite networks. *Discrete Applied Mathematics*, 121(1):261–277, 2002.

[30] R. Yuster. Almost exact matchings. *Algorithmica*, 63(1):39–50, 2012.