A $\mathcal{T}$-path packing algorithm

Csaba Berki
Vrije Universiteit, Amsterdam

Supervisor:

Alexander Schrijver
CWI, Amsterdam

July 11, 2002

Preface

This master's thesis in mathematics was written within the frame of a one-year program at the Vrije Universiteit, after completing four years of studies at Eötvös Loránd University, Budapest, Hungary. During the stay in the Netherlands I attended the combinatorial optimization seminar at CWI (Centre for Mathematics and Computer Science) where I got in closer connection to my supervisor, Alexander Schrijver.

Following my supervisor's advice I started to study the algorithm of Gabow and Stallmann [2], and also the chapter of T-paths in Schrijver's forthcoming book [8]. In March and April I gave a lecture on the algorithm at the seminar at CWI. This thesis was written during the spring semester in 2002.

I would like to express my grateful thanks to everyone who helped me writing this thesis. Especially I would like to thank everything for my supervisor, Lex Schrijver. It would be too long to mention everything he did for me. He is not only one of the greatest mathematicans but he is also very polite and follows students with special attention. It was a great experience to work with him.

Contents

Chapter 1

Introduction

The goal of this thesis is to present an algorithm for finding disjoint $\mathcal{T}$-paths (this will be defined later) which is also efficient and not too difficult to implement on a computer. There are good tools in this relatively new area of graph theory such as minimum-maximum theorems by Mader (see subsection 1.1.3). These theorems give us the maximum number of disjoint $\mathcal{T}$-paths. However using these theorems we cannot calculate this algorithmically in polynomial time and they also don't help us to find these $\mathcal{T}$-paths.

On the other hand, it was proved by Lovász [8] that we can find the maximum number of disjoint $\mathcal{T}$-paths in polynomial time using linear matroid parity algorithms. Linear matroids and matroid parity problem will be also defined.

I will take the most efficient linear matroid parity algorithm (so far) by Gabow and Stallmann [2] and use it to solve our problem by constructing a linear matroid which will be also easy to implement on a computer. I will show a method to speed up the general linear matroid parity algorithm in our case. The complexity of the algorithm will be $O(mn^2)$ where m denotes the number of edges and n the number of vertices in the graph.

1.1 $\mathcal{T}$-paths

We start with some definitions and theorems to show where $\mathcal{T}$-paths came from and what we know about them. I'll use the term *openly disjoint* paths for those disjoint paths which can have common endpoints and simply *disjoint* paths whose endpoints must be also different.

1.1.1 Menger's theorem

Consider an undirected graph $G = (V, E)$ with two special vertices, s and t[1]. If we are interested in the maximum number of openly disjoint paths between s and t then Menger's theorem (vertex-disjoint, undirected case) helps us.

[1] From words *s*ource and *t*arget, but in the undirected case the role of them is the same.

Theorem 1.1 (Menger's vertex-disjoint theorem) *Suppose $\{s,t\} \notin E$. Then the maximum number of openly disjoint paths between s and t is equal to the minimum number of vertices such that each path between s and t meets at least one of these vertices.*

We don't prove this theorem here since it is part of the basic graph theory material. We would like to generalize this theorem for more sources and targets.

If we have two disjoint sets of vertices $S = \{s_1, s_2, \ldots, s_k\}$ and $T = \{t_1, t_2, \ldots, t_l\}$ and we would like to know the maximum number of disjoint paths connecting these two sets then with an easy trick we can again use Menger's theorem. Add two new vertices s and t to graph G, and connect them with the vertices in S and T respectively, i.e. add edges $\{s, s_i\}$ and $\{t, t_j\}$ for all i and j. Then obviously the maximum number of disjoint paths connecting the sets is equal to the maximum number of openly disjoint paths between s and t; so Menger's theorem gives the answer.

In the openly disjoint case the above trick doesn't work. We can multiplicate all points in S and T to have degree one vertices and connecting them to the new s and t vertices but it's easier to merge all vertices in S and T to a single vertex s and t, so that they will be connected to all such vertices which were connected to at least one of the vertices in S and T. Parallel edges can be deleted since only one of the original edges can be used by the openly disjoint paths and it doesn't matter which one. Then Menger's theorem for s and t gives the answer.

1.1.2 Gallai's theorem

If we don't know which vertex will be a "source" and which is a "target" vertex, so only one set of vertices is given as the endpoints of paths then the answer doesn't follow directly from Menger's theorem. Tibor Gallai proved a min-max theorem for this question using Tutte – Berge formula. Let us first present the formula and Schrijver's proof for it.

We call a component of a graph *odd* if it has an odd number of vertices, and denote by $o(G)$ the number of odd components of graph G. In 1947 Tutte proved a condition for graphs having a perfect matching:

Theorem 1.2 (Tutte's theorem) *A graph $G = (V, E)$ has a perfect matching if and only if $\forall X \subseteq V \colon o(G - X) \leq |X|$.*

Let $\nu(G)$ denote the cardinality of a maximum-size matching in a graph G. Then in 1958, Berge proved the following min-max theorem which implies theorem 1.2:

Theorem 1.3 (Tutte – Berge formula) *In a graph $G = (V, E)$*

$$\nu(G) = \min_{U \subseteq V} \tfrac{1}{2}(|V| + |U| - o(G - U)). \tag{1.1}$$

Proof. The maximum is at most the minimum, since for each $U \subseteq V$:

$$\nu(G) \leq |U| + \nu(G - u) \leq |U| + \tfrac{1}{2}(|V \setminus U| - o(G - U)) = \tfrac{1}{2}(|V| + |U| - o(G - U)).$$

We prove the reverse inequality by induction on $|V|$, the case $V = \emptyset$ being trivial. We can assume that G is connected, as otherwise we can apply induction to the components of G.

First assume that there exist a vertex v covered by all maximum-size matchings. Then $\nu(G-v) = \nu(G) - 1$, and by induction there exists a subset U' of $V \setminus \{v\}$ with

$$\nu(G-v) = \tfrac{1}{2}\left(|V \setminus \{v\}| + |U'| - o(G - v - U')\right).$$

Then $U := U' \cup \{v\}$ gives equality in (1.1).

So we can assume that there is no such v. In particular, $\nu(G) < \frac{1}{2}|V|$. We show that there exists a matching of size $\frac{1}{2}(|V|-1)$, which implies the theorem (taking $U := \emptyset$).

Indeed suppose to the contrary that any maximum-size matching M misses at least two distinct vertices u and v. Among all such M, u, v choose them such that the distance $\operatorname{dist}(u, v)$ of u and v in G is as small as possible.

If $\operatorname{dist}(u, v) = 1$, then u and v are adjacent, and hence we can augment M by uv, contradicting the maximality of $|M|$. So $\operatorname{dist}(u, v) \geq 2$, and hence we can choose an intermediate vertex t on a shortest $u-v$ path. By assumption, there exists a maximum-size matching N missing t. Choose such an N with $|M \cap N|$ maximal.

By the minimality of $\operatorname{dist}(u, v)$, N covers both u and v. Hence, as M and N cover the same number of vertices, there exists a vertex $x \neq t$ covered by M but not by N. Let $x \in e = xy \in M$. Then y is covered by some edge $f \in N$, since otherwise $N \cup \{e\}$ would be a matching larger than N. Replacing N by $(N \setminus \{f\}) \cup \{e\}$ increases the intersection with M contradicting the choice of N. ∎

Now we continue with Gallai's theorem. Let $G = (V, E)$ be an undirected graph and let T be a subset of the vertices. We call a path a *T-path* if its endpoints are different vertices in T but its other vertices are in $V \setminus T$. Now we prove the theorem as in [8].

Theorem 1.4 (Gallai's T-path theorem) *The maximum number of disjoint T-paths is equal to the minimum value of*

$$|U| + \sum_K \lfloor \tfrac{1}{2}|K \cap T| \rfloor$$

taken over $U \subseteq V$, where K ranges over the components of $G - U$.

Proof. Each T-path intersects U or traverses at least two vertices in $K \cap T$ for some component K of $G - U$, hence the maximum is at most the minimum.

To see equality, let μ be equal to the minimum. Let the graph $\tilde{G} = (\tilde{V}, \tilde{E})$ arise from G by adding a disjoint copy G' of $G - T$, and making the copy v' of each $v \in V \setminus T$ adjacent to v and to all neighbours of v in G. By the Tutte–Berge formula, $\tilde{G}$ has a matching M of size $\mu + |V \setminus T|$. To see this, we must prove that for any $\tilde{U} \subseteq \tilde{V}$:

$$|\tilde{U}| + \sum_{\tilde{K}} \lfloor \tfrac{1}{2}|\tilde{K}| \rfloor \geq \mu + |V \setminus T|, \tag{1.2}$$

where $\tilde{K}$ ranges over the components of $\tilde{G} - \tilde{U}$. Now if for some $v \in V \setminus T$ exactly one of v, v' belongs to $\tilde{U}$, then we can delete it from $\tilde{U}$, thereby not increasing the left-hand side of the inequality (1.2).

So we can assume that for each $v \in V \setminus T$, either $v, v' \in \tilde{U}$ or $v, v' \notin \tilde{U}$. Define $U := \tilde{U} \cap V$. Then each component K of $G - U$ is equal to $\tilde{K} \cap V$ for some component of $\tilde{G} - \tilde{U}$. Hence

$$|\tilde{U}| + \sum_{\tilde{K}} \lfloor \tfrac{1}{2} |\tilde{K}| \rfloor = |U| + \sum_{K} \lfloor \tfrac{1}{2} |K \cap T| \rfloor + |V \setminus T| \geq \mu + |V \setminus T|,$$

where K ranges over the components of $G - U$. This exactly proves (1.2).

So $\tilde{G}$ has a matching M of size $\mu + |V \setminus T|$. Let N be the matching $\{ vv' \mid v \in V \setminus T \}$ in $\tilde{G}$. As $|M| = \mu + |V \setminus T| = \mu + |N|$, the union $M \cup N$ has at least μ components with more edges in M than in N. Each such component is a path connecting two vertices in T. Then contracting the edges in N yields μ disjoint T-paths in G. ∎

1.1.3 Mader's theorem

Now we generalize Gallai's theorem, too. We would not like the paths to connect any two vertices of T, but some pairs may not be connected by a path. If transitivity holds for this then we have an equivalent relation, i.e., we have some groups of vertices and all vertices can be connected only to other group's vertices. Mader's theorem gives us the maximum number of such (openly) disjoint paths.

Let $G = (V, E)$ a graph, $T \subseteq V$ a subset of vertices and $\mathcal{T} = \{T_1, T_2, \ldots, T_k\}$ a partition on T. Let furthermore $t := |T| = \sum |T_i|$.

Definition 1.5 *A path is called* $\mathcal{T}$-path *if its endpoints are in different sets T_i, T_j and all other vertices are all in $V \setminus T$.*

For an example, figure 1.1 shows two $\mathcal{T}$-paths (thick edges).

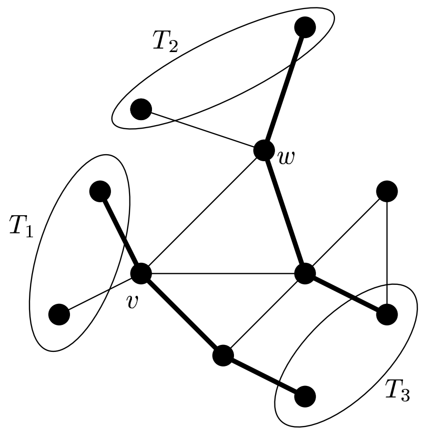

Figure 1.1: Two $\mathcal{T}$-paths in a graph

First we present Schrijver's short proof [9] for the disjoint case of Mader's theorem. The openly disjoint case, which was the original form of Mader's theorem will follow from this.

Theorem 1.6 (Mader's disjoint $\mathcal{T}$-paths theorem) *The maximum number of disjoint $\mathcal{T}$-paths is equal to the minimum value of*

$$|U_0| + \sum_{i=1}^{n} \left\lfloor \tfrac{1}{2}|B_i| \right\rfloor,$$

taken over all partitions $U_0, \ldots, U_n$ of V such that each $\mathcal{T}$-path intersects U_0 or traverses some edge spanned by some U_i. Here B_i denotes the set of vertices in U_i that belong to T or have a neighbour in $V \setminus (U_0 \cup U_i)$.

Proof. Let μ be the minimum value. Trivially, the maximum value is at most μ, since any $\mathcal{T}$-path disjoint from U_0 and traversing an edge spanned by U_i, traverses at least two vertices in B_i.

Fixing V, choose a counterexample $E, \mathcal{T}$ minimizing

$$|E| - |\{\{r,s\} \mid r,s \in V, \exists R,S \in \mathcal{T} : r \in R, s \in S, R \neq S\}|. \tag{1.3}$$

Then each $S \in \mathcal{T}$ is a stable set of G since any egde e spanned by T can be deleted without changing the maximum and minimum value in Mader's theorem (as no $\mathcal{T}$-paths can traverse e and deleting e does not change any set B_i), while decreasing (1.3).

If $|S| = 1$ for each $S \in \mathcal{T}$, the theorem reduces to Theorem 1.4. So we can choose $S \in \mathcal{T}$ with $|S| \geq 2$. Choose $s \in S$. Replacing $\mathcal{T}$ by $\mathcal{T}' := (\mathcal{T} \setminus \{S\}) \cup \{S \setminus \{s\}, \{s\}\}$ decreases (1.3), but not the minimum in Mader's theorem (as each $\mathcal{T}$-path is a $\mathcal{T}'$-path and as $\bigcup \mathcal{T}' = T$). So there exists a collection $\mathcal{P}$ of μ disjoint $\mathcal{T}'$-paths.

Necessarily, there is a path $P_0 \in \mathcal{P}$ connecting s with another vertex in T, all other paths in $\mathcal{P}$ being $\mathcal{T}$=paths. Let u be an internal vertex of P_0. Replacing $\mathcal{T}$ by $\mathcal{T}'' := (\mathcal{T} \setminus \{S\}) \cup \{S \cup \{u\}\}$ decreases (1.3), but not the minimum in Mader's theorem (as each $\mathcal{T}$-path is a $\mathcal{T}''$-path and as $\bigcup \mathcal{T}'' \supset T$). So there exists a collection $\mathcal{Q}$ of μ disjoint $\mathcal{T}''$-paths. Choose $\mathcal{Q}$ such that $\mathcal{Q}$ uses a minimal number of edges not used by $\mathcal{P}$.

Necessarily, u is an end of some path $Q_0 \in \mathcal{Q}$, all other paths in $\mathcal{Q}$ being $\mathcal{T}$-paths. As $|P| = |Q|$ and as u is not an end of any path in $\mathcal{P}$, there exists an end v of some path $P \in \mathcal{P}$ that is not an end of any path in $\mathcal{Q}$. Now $\mathcal{P}$ intersects at least one path in $\mathcal{Q}$ (since otherwise $P \neq P_0$, and $(\mathcal{Q} \setminus \{Q_0\}) \cup \{P\}$ would consist of μ disjoint $\mathcal{T}$-paths). So when following P starting at v, there is a first vertex w that is on some path in $\mathcal{Q}$, say on $Q \in \mathcal{Q}$.

For any end x of Q let Q^x be the $x - w$ part of Q, let P^v be the $v - w$ part of P, and let U be the set in $\mathcal{T}''$ containing v. Then for any end x of Q we have that Q^x is part of P or the other end of Q belongs to U, since otherwise by rerouting part Q^x of Q along P^v, Q remains a $\mathcal{T}''$-path disjoint from the other paths in $\mathcal{Q}$, while we decrease the number of edges used by $\mathcal{Q}$ and not by $\mathcal{P}$, contradicting the minimality assumption.

Let y, z be the ends of Q. We can assume that $y \notin U$. Then Q^z is part of P, hence Q^y is not part of P (as Q is not part of P, as otherwise $Q = P$, and hence v is an end of Q), so $z \in U$. As z is on P and as also v belongs to U and is on P, we have $P = P_0$. So $U = S \cup \{u\}$ and $Q = Q_0$ (since Q^z is part of P, so $z = u$). But then rerouting part Q^z of Q along P^v gives μ disjoint $\mathcal{T}$-paths, contradicting our assumption. ∎

Theorem 1.7 (Mader's openly disjoint T-paths theorem) *Let $G = (V, E)$ be a graph and let T be a stable subset of V. Then the maximum number of openly disjoint T-paths is equal to the minimum value of*

$$|U_0| + \sum_{i=1}^{n} \left\lfloor \tfrac{1}{2} |B_{G-U_0}(U_i)| \right\rfloor ,$$

where $U_0, U_1, \ldots, U_n$ partition $V \setminus T$ such that each T-path disjoint from U_0, traverses some edge spanned by some U_i. Here $B_H(U)$ for any graph H denotes the set of vertices in U having a neighbour in H that is not in U.

Proof. Trivially, the maximum is not more than the minimum. To see equality, we can assume that no two vertices in T have a common neighbour v. Otherwise we can apply induction by deleting v, which reduces both the maximum and minimum by 1.

Now the theorem follows from disjoint case (theorem 1.6) applied to $G - T$ and the collection $\mathcal{T} := \{N(s) \mid s \in T\}$. ∎

We call a set of disjoint $\mathcal{T}$-paths a *$\mathcal{T}$-path packing*. In figure 1.1 the maximum number of $\mathcal{T}$-paths is two since choosing $U_0 := \{v, w\}$ and $\{U_i\} = \{\{x\} \mid x \in V \setminus \{v, w\}\}$ in theorem 1.6 result that the minimum is also 2. So in the figure we saw a maximum $\mathcal{T}$-path packing.

We can use the same tricks as after Menger's theorem (theorem 1.1) to see that in theorem 1.7 it is enough to consider single vertices in T, i.e. if there were subsets of T they could be merged to single vertices. The other trick shows that from Mader's openly disjoint theorem (theorem 1.7) also follows the disjoint case (theorem 1.6) so they are equivalent.

Furthermore the proof of the disjoint case (theorem 1.6) shows that Gallai's theorem (theorem 1.4) is a special case of it, and openly disjoint case (theorem 1.7) imply Menger's theorem (theorem 1.1) by taking $T = \{s, t\}$.

If $\mathcal{T} = \{\{v\} \mid v \in V\}$ then the $\mathcal{T}$-path problem is simply the graph matching problem and theorem 1.6 reduces to the Tutte – Berge formula (theorem 1.3) with $\mathcal{T}$-paths being simply matched edges, $U_0 = U$ and U_i-s have to cover all edges so they have to be the components of $G - U$. Of course we can also use the openly disjoint case (theorem 1.7) with the first trick (i.e. duplicating V, connect copies with originals and let T be the set of copies).

1.2 Matroids

Now we define matroids and then matroid parity problem which we will use in chapter 2 and later. There are two equivalent definitions of the matroids in use. We will use the definition of the matroids through bases.

Definition 1.8 *Let E be an arbitrary set and $\mathcal{B}$ a set of subsets of E. We call the pair $\mathcal{M} = (E, \mathcal{B})$* a matroid *if $\mathcal{B} \neq \emptyset$ and from $B_1, B_2 \in \mathcal{B}$ follows that for all $x \in B_1 \setminus B_2$ there exists a $y \in B_2 \setminus B_1$ such that $B_2 \setminus \{y\} \cup \{x\} \in \mathcal{B}$.*

We call the elements of $\mathcal{B}$ the *bases*. A set $F \subseteq E$ is *independent* if there exists a base $B \in \mathcal{B}$ such that $F \subseteq B$. If there is no such B then F is *dependent*. A minimal dependent set is called *circuit*. The *rank function* r of $\mathcal{M}$ is denoted by $r(M) = \max\{|F| : F \subseteq M, F \text{ is independent in } \mathcal{M}\}$ for all $M \subseteq E$.

A pair of elements $a, b \in E$ called a *swap* for a base $B \in \mathcal{B}$ if $b \in B$ and $B \setminus \{b\} \cup \{a\}$ is also a base.

A *linear matroid* is a special matroid where E is a subset of a vector space $\mathbb{F}^n$ and the bases are the bases of E in linear algebra sence.

For another example of matroids consider the next theorem.

Theorem 1.9 *Let $G = (V, E)$ be a graph and $\mathcal{T}$ a collection of disjoint sets such that $\bigcup \mathcal{T} = T \subseteq V$. Define $\mathcal{B} := \{B \subseteq T \mid B$ is the set of endpoints of a maximum size $\mathcal{T}$-path packing$\}$. Then $(T, \mathcal{B})$ is a matroid, we call it* Mader-matroid.

Proof. $\mathcal{B}$ is not empty even if there is no $\mathcal{T}$-path, since then $\mathcal{B} = \{\emptyset\}$. Suppose $B_1, B_2 \in \mathcal{B}$ and $x \in B_1 \setminus B_2$. Let $\mathcal{P}$ and $\mathcal{Q}$ be two maximum size $\mathcal{T}$-path packings such that $\text{ends}(\mathcal{P}) = B_1$ and $\text{ends}(\mathcal{Q}) = B_2$ where $\text{ends}(\mathcal{R})$ denotes the set of endpoints of the paths in $\mathcal{R}$.

Let $P \in \mathcal{P}$ be the path ending in x and $T_x \in \mathcal{T}$ the set containing x. If P is disjoint from the paths in $\mathcal{Q}$ then $\mathcal{Q} \cup \{P\}$ is a bigger $\mathcal{T}$-path packing contradicting the maximality of $\mathcal{Q}$. Hence we can choose the closest vertex w to x on the path P such that w is also on a path $Q \in \mathcal{Q}$. At least one endpoint z of Q is not in T_x, and let y be the other endpoint of Q. Redirect the $y - w$ part of Q along $x - w$ part of P to obtain a new $\mathcal{T}$-path R connecting x and z. Then $\mathcal{R} := \mathcal{Q} \setminus \{Q\} \cup \{R\}$ is a $\mathcal{T}$-path packing too.

If $y \notin B_1$ then we are done since $B_2' :=\text{ends}(\mathcal{R}) = B_2 \setminus \{y\} \cup \{x\}$. Otherwise start from the beginning replacing x with y, $\mathcal{Q}$ with $\mathcal{R}$ and B_2 with B_2'. The process will stop since the number of edges used by $\mathcal{Q}$ but not by $\mathcal{P}$ decreases in each step. $\text{ends}(\mathcal{Q}) \subsetneq B_2 \cup \{x\}$ in each step, B_1 and $\mathcal{P}$ don't change so the final $y \notin B_1$ will be in the original B_2 and final B_2' will be really $B_2 \setminus \{y\} \cup \{x\}$. ∎

1.3 Matroid parity problem

Now we define the matroid parity problem. Again there are more equivalent definitions. Let $\mathcal{M} = (S, \mathcal{B})$ be a matroid and let E be a set[2] of pairs of S, i.e. $E = \{E_1, E_2, \ldots, E_m\}$ such that $|E_i| = 2$ for all i. We call a set $M \subseteq E$ a *matroid matching* if $r(M) := r(\bigcup M) = 2|M|$ so if the pairs of M are disjoint and the union $\bigcup M$ is an independent set in $\mathcal{M}$.

The *matroid parity problem* or *matroid matching problem* is to find the maximum size matching in a matroid. This problem generalizes many other problems, the most important two is graph matching and matroid intersection. When Lawler invented the matroid parity problem in 1971, he also wanted to find a common generalization of graph matching and matroid intersection problems. Taking a vertex set V of a graph G, the trivial matroid $(V, \{V\})$, and let E be the edges of graph, the matroid parity problem reduces to graph matching in G. Moreover by taking two matroids $\mathcal{M}_1 = (S, \mathcal{B}_1)$ and $\mathcal{M}_2 = (S, \mathcal{B}_2)$ we can obtain their intersection problem as follows. Consider a copy S' of S, let $E := \{\{s, s'\} \mid s \in S\}$, and let $\mathcal{B}_2'$ be the collection of the same subsets of S' as the subset of S in $\mathcal{B}_2$. Then the matroid parity on $\mathcal{M} := (S \cup S', \{B_1 \cup B_2' \mid B_1 \in \mathcal{B}_1, B_2' \in \mathcal{B}_2'\})$ reduces to matroid intersection.

[2] S and E can be infinite but we will only need the finite case.

In general the matroid parity problem is NP-complete, or unsolvable in polynomial time, if the matroid is given by an "independence oracle". Lovász [5] (and independently also Jensen and Korte [4]) proved that it requires exponential time in general.

We can ask the rank of a set from the oracle. Assume S is a $4k$-size set. Let the set of pairs E be a partition on S and let the bases $\mathcal{B} := \{B : |B| = 2k, B \text{ is not a union of } k \text{ pairs of } E\}$. It is easy to check that this is a matroid. Then the rank of all sets having more than $2k$ elements is $2k$, all sets having at most $2k-1$ elements is independent. Let $\mathcal{B}' := \{B : |B| = 2k, B \text{ is not a union of } k \text{ pairs of } E \text{ or } B = B_0\}$, where B_0 is a union of k pairs. $(S, \mathcal{B}')$ is again a matroid for each B_0. In $(S, \mathcal{B}')$ the maximum size matching is B_0, but in $(S, \mathcal{B})$ there is no matching of size k. The only way we can distinguish this two matroids is asking the rank of B_0 since all other sets have the same rank in both matroids. There are $\binom{2k}{k}$ possibilities for B_0, and in the worst case we have to ask all these sets to decide which is the oracle's matroid. So the running time of a matroid parity algorithm is exponential since the maximum size matching is different in this two matroids.

However if the matroid is linearly representable then this problem can be solved in polynomial time, this was proved also by Lovász and he gave an algorithm, too [5, 7]. This was an important and pioneering result, however the algorithm itself is complicated and its running time is a high degree polynomial. Then other algorithms were developed to solve linear matroid parity problem, they are more efficient then Lovász's algorithm. In 1986 Gabow and Stallmann made an algorithm using augmenting paths [2] with running time $O(mn^3)$. Then in 1990 Orlin and Vande Vate presented an other algorithm using a sequence of matroid intersection algorithms [6] with running time $O(mn^4)$. This two algorithms can be asymptotically speeded up to $O(mn^{2.376})$ and $O(mn^{3.376})$ if we use the method of fast matrix multiplication by Coppersmith and Winograd [1]. Here $n = r(S)$ denotes the rank of the matroid and $m = |S|$ denotes the number of elements.

Chapter 2

First part of the algorithm

In the introduction we've already mentioned that we would like to construct a linear matroid representing the graph and the given subsets. Then we will solve the $\mathcal{T}$-path packing problem as a matroid parity problem on this matroid. In this chapter, first I present this linear matroid and I prove that the matroid parity problem is equivalent to the problem of finding the maximum-size $\mathcal{T}$-path packing. Then I show a method to efficiently construct the coefficient matrix which will be needed in the second part of the algorithm. I close this chapter with some words about the complexity of this part of algorithm. From now I'll call a matroid matching simply a *matching*.

In one step the algorithm will augment the matching by one pair only, if possible, but this will also increment the number of $\mathcal{T}$-paths by one. After some initializations it will start with a matching (with one $\mathcal{T}$-path in it), then after $s-1$ steps of augmentation it gets the maximum-size matching (with s $\mathcal{T}$-paths in it) and finally with one more step it checks that we cannot augment this, where s is the maximum number of disjoint $\mathcal{T}$-paths in the graph. Each step will take $O(mn)$ time, so finding the maximum size $\mathcal{T}$-path packing will take $O(mns)$ time which is $O(mn^2)$ since there are at most $\frac{|T|}{2} \leq \frac{n}{2}$ $\mathcal{T}$-paths.

Each step can be separated into two main parts. The first part always constructs a coefficient matrix from the given matching (this matching will be simply the matching resulted by the previous step, except in the first step, of course) and then the second step, using this coefficient matrix will augment the matching by one pair, if possible. The coefficient matrix will represent the elements in the linear matroid as a linear combination of the matched element plus some additional elements.

2.1 $\mathcal{T}$-paths as a matroid parity problem

So first I show how $\mathcal{T}$-path's problem can be formulated as a linear matroid parity problem.

Let $G = (V, E)$ be a graph with vertices $V = \{v_1, v_2, \dots v_n\}$ and (nondirected) edges $E = \{e_1, e_2, \dots e_m\}$. Let again $\mathcal{T} = \{T_1, T_2, \dots T_k\}$ be disjoint subsets of V, $T := \bigcup \mathcal{T}$ and $t := |T| = \sum_i |T_i|$.

Choose rowvectors from vector space $\mathbb{R}^{2n}$ in the following way. Every edge $e_i = v_j v_l$ in the graph will be represented by a pair of vectors $\overline{e}_i, \underline{e}_i$ which I'll call *upper* and *lower* vectors of the edge e_i and they will look like

$$
\begin{array}{cccccccccccl}
 & & {\scriptstyle 1} & & {\scriptstyle 2j-1} & {\scriptstyle 2j} & & {\scriptstyle 2l-1} & {\scriptstyle 2l} & & {\scriptstyle 2n} & \\
\overline{e}_i & = & (0, & \ldots & 1, & 0, & \ldots & -1, & 0, & \ldots & 0) & \multirow{2}{*}{\text{if } v_j, v_l \in V \setminus T,} \\
\underline{e}_i & = & (0, & \ldots & 0, & 1, & \ldots & 0, & -1, & \ldots & 0) & \\
 & & & & & & & & & & & \\
\overline{e}_i & = & (0, & \ldots & 1, & 0, & \ldots & -1, & 0, & \ldots & 0) & \text{if } v_j \in T_p,\ v_l \in V \setminus T, \\
\underline{e}_i & = & (0, & \ldots & p, & 0, & \ldots & 0, & -1, & \ldots & 0) & \\
 & & & & & & & & & & & \\
\overline{e}_i & = & (0, & \ldots & 1, & 0, & \ldots & -1, & 0, & \ldots & 0) & \text{if } v_j \in T_p,\ v_l \in T_q. \\
\underline{e}_i & = & (0, & \ldots & p, & 0, & \ldots & -q, & 0, & \ldots & 0) &
\end{array}
$$

Table 2.1: Coordinates of the vectors

Let e' denote the mate of a vector e either it is an upper or lower vector, so $\overline{e}'_i := \underline{e}_i$ and $\underline{e}'_i := \overline{e}_i$.

Then consider the linear matroid formed by the vectors above. The next theorem shows the relationship between the matchings of the linear matroid and the forests of the graph G. It also gives a motivation why we chose the coordinates of vectors in the way we did.

Theorem 2.1 *A set of pairs in the matroid forms a matching if and only if the corresponding edges in the graph form a forest such that every component has at most two vertices in T and if there are two of them then they are in different T_p, T_q.*

Proof. First assume we have a matching, i.e. there is no non-trivial linear combination of the vectors in the matching.

Then all components of the corresponding forest in the graph have at most two vertices in T. Otherwise consider a component G_1 with at least three vertices in T. If G_1 has n_1 vertices then it have at least $n_1 - 1$ edges since a component is connected. It means that we have at least $2(n_1 - 1)$ vectors which are all zero except at the columns corresponding to vertices of G_1. There are at least three vertices of G_1 in T so they give 3 additional zero-column. Hence at most $2n_1 - 3$ columns are non-zero in these (at least) $2n_1 - 2$ vectors which means they are dependent.

Next I prove that the edges really form a forest i.e. circuit is not allowed. Indeed, if we have a minimal circuit then we can construct a non-trivial linear combination of vectors. First choose an edge of the circuit and "multiply" the corresponding two vectors by 1. The two vectors of the next edge in the circuit can eliminate the common vertex's coordinates if we multiply them by ± 1 in a proper way. Continuing this we get zeros at all coordinates since easy induction shows that always two vertices' coordinates remain non-zero and they have contrary sign; so finally the first and last vertices are the same and therefore its coordinates are eliminated, too.

Next assume that we have a component with two vertices v, w in the same T_p. Then do a similar construction as above for the path connecting v and w. Multiply the first edge's upper vector by p and the lower vector by -1. So v's coordinates add up to zero. Continuing the elimination along the path with coefficients $\pm p$ and ∓ 1 we get finally that w's coordinate eliminates, too.

For the other direction's proof suppose we have such a forest as in the theorem. Each component's vectors form a block in the matrix consisting of all vectors of the forest so if the vectors have non-trivial linear combination then there is one in a component too. Hence it suffices to prove that a tree's vectors connecting at most two vertices of T (from different T_p, T_q) form a matching.

Assume it is not true and choose a counter-example, a subtree with minimal number of edges such that the corresponding vectors are dependent. If it has a degree one vertex not in T then this vertex's coordinates are ± 1 in different columns and they appear only once. So a linear combination can be zero only if the corresponding two vectors were multiplied by zero i.e. this edge doesn't need to be in the counter-example, contradicting minimality.

So the subtree is just a path connecting the two vertices in T. It is easy to see that the linear combination can be zero only if all upper vectors were multiplied by the same $\pm a$ number and lower vectors were multiplied by $\pm b$ (otherwise inner vertices' coordinates don't add up to zero). To eliminate the coordinates of vertex in T_p we have $\frac{a}{b} = \pm p$ and to eliminate the other end we have $\frac{a}{b} = \pm q$ which is a contradiction since $|p| \neq |q|$. ∎

So we have proved that from a matching we can produce a forest and vice versa. From a forest we can get $\mathcal{T}$-paths by simply deleting edges which are not on a path connecting two vertices in T. The next theorem shows that the maximum size matchings and the maximum $\mathcal{T}$-path packings are in strong relation.

Theorem 2.2 *If G is connected, the size of a maximum $\mathcal{T}$-path packing in a graph is s and the maximum size matching has u pairs in it, then $u - s = n - t$.*

Proof. First take s disjoint $\mathcal{T}$-paths and add edges to this forest such that each component remains a tree and contain at most two vertices from T (from different T_p, T_q). Since G is connected we can add edges if a component doesn't contain any point from T. So finally we get a forest with s components which intersects T in two vertices and $t - 2s$ components which contains only one vertex from T. So it has $n - (s + t - 2s) = n - t + s$ edges and this gives $u \geq n - t + s$.

For the other inequality now take a maximum size matching and consider the corresponding forest. There are at most s $\mathcal{T}$-paths so deleting at most s edges in the forest (one in each $\mathcal{T}$-path) we get a forest with each component containing at most one vertex in T. This means that the forest has at least t components so at most $n - t$ edges. This gives $u - s \leq n - t$ which proves the theorem. ∎

Note that equality means that a maximum size matching corresponds to a forest with all components containing a vertex from T and exactly s of them contains two. So it already contains the s $\mathcal{T}$-paths, too.

2.2 Calculating the coefficient matrix

We can assume that G is connected since the maximum $\mathcal{T}$-path packing can be obtained as the union of the maximum $\mathcal{T}$-path packings in the components (with $T_i^K = T_i \cap K$ for any component K). We can also assume that there are no loops and no parallel edges in the graph since a path can't contain any loops and parallel edges and if two different paths use parallel edges then they are not disjoint.

Sometimes in $\mathcal{T}$-path's terminology the sets T_i are stable but if not then we can assume this since any edge in any T_i can't occur in a $\mathcal{T}$-path. If there is a vertex in T_i with degree more than one, we can make a simple transformation to make all $v \in T$ a degree one vertex (see figure 2.1). This will simplify the algorithm only, there will be less cases to check. First add a new vertex v' to the graph and connect it to v, then change T_i (which contains v) to be $T_i \setminus \{v\} \cap \{v'\}$.

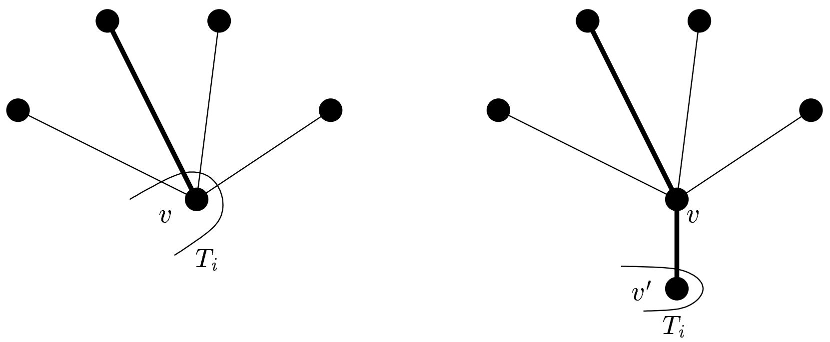

Figure 2.1: Transformation made to vertices of T

Theorem 2.3 *Applying the above transformation the size of maximum $\mathcal{T}$-path packing (s) doesn't changes.*

Proof. It's easy to see that s doesn't decreases. Choose s $\mathcal{T}$-paths. The only $\mathcal{T}$-path can be affected by the transformation is the $\mathcal{T}$-path starting from v if such exists. Then adding the new edge to this path gives a $\mathcal{T}$-path packing of the same size.

Now assume there is an $\mathcal{T}$-path packing of size $s+1$ after the transformation. If a $\mathcal{T}$-path uses the new edge (i.e. starts from v') then deleting this edge we get $s+1$ $\mathcal{T}$-paths in the original graph which is a contradiction. If v is not in any of the $\mathcal{T}$-paths then v' also isn't and the transformation doesn't change the paths, so this is also not possible. Finally if v is touched by a path P but v' is not its end then change P in the following way. Clear the part of P after v and add edge vv' to it. This new path is also a $\mathcal{T}$-path, disjoint from others, so we get a $\mathcal{T}$-path packing of size $s+1$ with P ending in v'. Using the first case this also leads to contradiction. ∎

Now let $M = \{\overline{e}_1, \underline{e}_1, \overline{e}_2, \underline{e}_2, ...\overline{e}_r, \underline{e}_r\}$ be the current matching which we would like to increase. The corresponding edges in the graph form a forest F. If there is a pair of vectors we can add to the matching then add it. So we can also assume that all components (trees) of F contain at least one vertex in T since if there is a component K with $K \cap T = \emptyset$ then adding an edge connecting K to an other component gives a tree with at most two vertices in T. Such a connecting edge always exists because G is connected.

To choose such a starting matching or forest without many calculations simply choose an arbitrary tree in $G \setminus T$ and add two $(T, V \setminus T)$-edges but starting from different T_p and T_q. $G \setminus T$

is also connected since all vertices in T has degree 1 and G is connected. So in this forest there are $t-2$ components with only 1 vertex and one component with $n-t+2$ vertices.

In section 4.1 we will see that if we start with this forest then after any steps of augmentation the improved M' matching will correspond to a forest F' with similar structure as F has, i.e., some components with two vertices in T and all other components consisting only from a vertex from T.

We would like to get the coefficients for all vectors that are not in M so the linear combination which is equal to the vector. Since M is usually not a base (else M is maximal and we are ready) this is not always possible. But if we use the above fact, how F looks like than we can easily add vectors to M becoming a base.

We can add a vector to M for all trivial components in the following way. Let v_i be a vertex which is also a trivial component (so $v_i \in T$). v_i has degree 1, so if we add to M the upper or lower vector corresponding to this edge in G then they are still independent since this vector has a unique non-zero coordinate at the $(2i-1)$th column. We can add one vector for all such components since they have all different unique non-zero coordinates.

If F has x trivial components (and $\frac{t-x}{2}$ non-trivial with two vertices in T) then F has $n-x-\frac{t-x}{2}$ edges ($x+\frac{t-x}{2}$ trees with n vertices total) so if $|M|=2r=2|F|$ then $x=2n-2r-t$. Adding these x vectors to M we get an independent set of $2r+x=2n-t$ vectors. They form a base since every vertex in T has a corresponding zero-column i.e. there are only $2n-t$ non-zero columns, hence there is at most $2n-t$ independent vectors.

Suppose we added only upper vectors to obtain the base. Let B denote this base. Now we can talk about unique $c_{i,b}$ and $d_{i,b}$ values ($b \in B$) such that

$$\overline{e}_i = \sum_{b\in B} c_{i,b} b$$
$$\underline{e}_i = \sum_{b\in B} d_{i,b} b$$

for all $\overline{e}_i, \underline{e}_i \notin B$. We will see that the added vectors are only needed once, for their pairs only.

First we will direct temporarily the edges of F to easily find paths we need. Choose a vertex v in T and direct the edges of its component to the direction of v (for example simply direct backward all edges of the tree when doing a breadth-first search from v). Also sign all vertices that belong to this component. When it reaches the other vertex w of T in that component then colour all vertices along the $w \to v$ path to white. We only have to follow the directed edges and colour the vertices we meet. Last colour v and put a link to w to know it is the other vertex in T. Then continue the search and if the component is fully examined then do the same for all other components too.

Figure 2.2 shows two components of F (thick edges with arrows) after directing the edges towards v_1 and v_2 and vertices coloured to white. Thin and dotted edges are not part of F but of G.

When finished with the directing of the edges we can easily decide from a non-matched edge to which type of the four below it belongs.

1. The two endpoints are in the same component. Then adding this edge produces a circuit in this component.

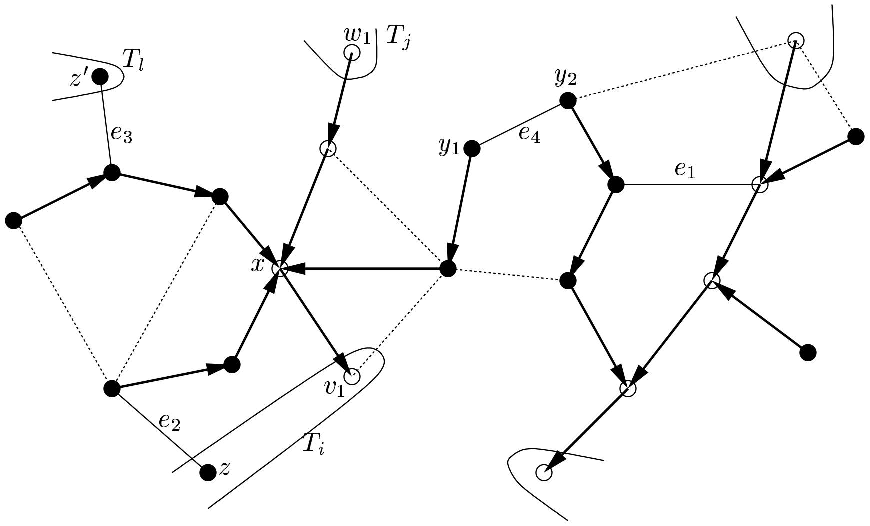

Figure 2.2: Directing the forest and colouring vertices to white

2. The edge connects a trivial component (a vertex z) to a bigger component. The corresponding upper vector is already in B.
   (a) One of the component's two vertices that are in T is in the same set T_i as z.
   (b) The three vertices are in three different sets T_i, T_j, T_l.
3. The edge connects two bigger components.

On figure 2.2 there are edges $e_{i_1}, e_{i_2}, e_{i_3}, e_{i_4}$ of the type 1, 2.(a), 2.(b) and 3, respectively. Now let's see how the coefficients can be calculated for different types of edges. First assume the whole matrix (of size $(2m - 2n + t) \times (2n - t)$ since $|B| = 2n - t$) is zero.

Assume the vectors of M have positive coordinates for the vertex to which the corresponding directed edge points and negative coordinates for the other vertex (recall tabular 2.1). If it is not so then we have to multiply that pair of vectors' column by -1 in the constructed coefficient matrix to get the real coefficients of the linear combination.

1. If the two endpoints are in the same component then follow the directed edges from one endpoint and add $(1, 0)$ and $(0, 1)$ coefficients to $\overline{e}_{i_1}$'s and $\underline{e}_{i_1}$'s row at the columns of the current edge, i.e. if the current edge in the path is e_j then

$$c_{i_1,\overline{e}_j} = 1, \qquad d_{i_1,\overline{e}_j} = 0,$$
$$c_{i_1,\underline{e}_j} = 0, \qquad d_{i_1,\underline{e}_j} = 1.$$

   On the path from the other endpoint add $(-1, 0)$ and $(0, -1)$ to coefficients. This guarantees that only the coefficients for vectors belonging to the circle will be non-zero since at the common edges of the two paths the coefficients add up to zero. The other parts which are really part of the circle remain unaffected by the other path, and coordinates will add up to zero except at the endpoints of e_{i_1}.

2. (a) First assume that v_1 and z (one of the endpoints of e_{i_2}) belong to the same T_i. Then $\overline{e}_{i_2}$ is a basic vector, $\underline{e}_{i_2}$ isn't. The lower vector's coefficient for the upper one will be $d_{i_2,\overline{e}_{i_2}} = i$ of course and the other coefficients we will get in the following way. Walk along the directed edges and set $\underline{e}_{i_2}$'s coefficients for these vectors to $(-i, 1)$ (or $(i, -1)$ if $\underline{e}_{i_2}$ has coordinates with different sign as the example has in table 2.1). This eliminates the sum at inner vertices and also at v_1 since it is also in T_i.

   Second, if w_1 and z are in the same set T_j then do the same we did just before from z to v_1 but with coefficients $(-j, 1)$ (or $(j, -1)$) of course and then do it for w_1 too but with different sign. As in the first case for the circuit this will eliminate the coefficients on the common part, too. The remaining parts (the path from z to w_1) give the correct coordinates since again everything add up to zero except the endpoint of e_{i_2}.

   (b) If $v_1 \in T_i$, $w_1 \in T_j$, $z' \in T_l$ are in different sets then first again $d_{i_3,\overline{e}_{i_3}} = l$ and then walk along the directed edges to the first white vertex x and set coefficients to $(l, -1)$ (or $(-l, 1)$) as in 2.(a) case. Then at x we have to eliminate the first coordinates too. We have to set the coefficients to some $(\alpha j, -\alpha)$ along the path from w_1 to x otherwise coordinates don't add up to zero. Also the coordinates have to be some $(\beta i, -\beta)$ for the path from x to v_1 to get zeros on that part of coordinates. So at the vertex x we have the following two equations (assuming the coordinates are positive at x for vectors corresponding to edges from z' and w_1 and they are negative for vectors leaving x in the direction v_1 and coefficients $(l, -1)$ were used):

$$\begin{aligned} l + \alpha j - \beta i &= 0 \\ -1 - \alpha + \beta &= 0 \end{aligned}$$

   We get $\alpha = \frac{i-l}{j-i}$ and $\beta = \frac{j-l}{j-i}$ so the coefficients will be $(\frac{j(i-l)}{j-i}, \frac{l-i}{j-i})$ and $(\frac{i(j-l)}{j-i}, \frac{l-j}{j-i})$.

3. Finally in the worst case we have to gain the correct coordinates from two components and we have two nonbasic vectors corresponding to e_{i_4}. Let y_1 and y_2 be the endpoints of e_{i_4}. First we would like to get the coordinates of $\overline{e}_{i_4}$ and $\underline{e}_{i_4}$ at y_1, which are $(1, 0)$ and $(0, 1)$. From y_1 to the first white vertex coefficients will be again simply $(-1, 0)$ for the upper one and $(0, -1)$ for the lower.

   At the white point we can set up two similar equations and we get the coefficients, if f and g are edges on the paths from w_1 and to v_1 then

$$\begin{aligned} c_{i_4,\overline{f}} &= -\frac{j}{i-j}, & d_{i_4,\overline{f}} &= -\frac{ij}{i-j}, \\ c_{i_4,\underline{f}} &= \frac{1}{i-j}, & d_{i_4,\underline{f}} &= \frac{i}{i-j}, \\ c_{i_4,\overline{g}} &= -\frac{i}{i-j}, & d_{i_4,\overline{g}} &= -\frac{ij}{i-j}, \\ c_{i_4,\underline{g}} &= \frac{1}{i-j}, & d_{i_4,\underline{g}} &= \frac{j}{i-j}. \end{aligned}$$

   On the other hand, to get coordinates at y_2 we have to do the same in y_2's component with opposite sign and of course the value of i and j can change.

It is easy to check that these coefficients really give the vectors we wanted since we always chose them so as to eliminate everything except the non-zero coordinates of the vector.

Let's see an example. Consider the graph on figure 2.3. The coordinates of the vectors one can check in table 2.2.

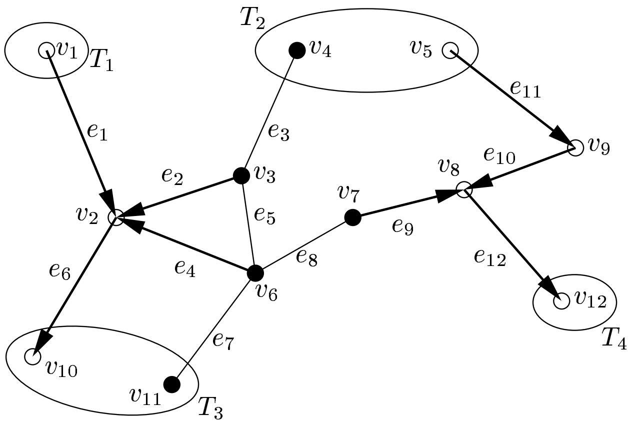

Figure 2.3: An example graph for calculating the coefficient matrix

And then the corresponding coefficient matrix is in table 2.3.

As we mentioned before, if a vector looks different (opposite sign) then in the second matrix we have to multiply the corresponding row or column by -1.

Note that basic vectors $\overline{e}_3$ and $\overline{e}_7$ was needed only once as we mentioned before. The construction shows that it is true in general since these basic vectors only used for its pair's coefficients other cases use only vectors of M.

2.3 Time needed for constructing the matrix

The forest F has at most $n-1$ edges so directing and labelling (to which component the vertex belongs) can be done in $O(n)$ and even the colouring of vertices too, since every vertex becomes white at most once so every edge of F was passed at most once more.

Then we have $O(n)$ basic vectors ($2n - s < 2n$) and $O(m)$ nonbasic vectors ($2m - 2n + s < 2m$). For all nonbasic vectors we have to do $O(n)$ step since they also uses just the edges of F plus maybe one simple step (the coefficient for the basic pair of a nonbasic vector). Maybe an edge was passed twice (e.g. in the circuit case) but never more. And calculating the formulas in case 2.(b) and 3 takes constant time for all vectors since it is three times the running time of at most two substraction, a multiplication, a division and a negation. So totally the construction of the coefficient matrix takes $O(mn)$ time.

	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}	v_{11}	v_{12}
$\overline{e}_1 =$	-1	1										
$\underline{e}_1 =$	-1	1										
$\overline{e}_2 =$		1	-1									
$\underline{e}_2 =$		1	-1									
$\overline{e}_3 =$			1	-1								
$\underline{e}_3 =$			1	-2								
$\overline{e}_4 =$		1				-1						
$\underline{e}_4 =$		1				-1						
$\overline{e}_5 =$			1			-1						
$\underline{e}_5 =$			1			-1						
$\overline{e}_6 =$		-1								1		
$\underline{e}_6 =$		-1								3		
$\overline{e}_7 =$						1					-1	
$\underline{e}_7 =$						1					-3	
$\overline{e}_8 =$						1	-1					
$\underline{e}_8 =$						1	-1					
$\overline{e}_9 =$							-1	1				
$\underline{e}_9 =$							-1	1				
$\overline{e}_{10} =$								1	-1			
$\underline{e}_{10} =$								1	-1			
$\overline{e}_{11} =$					-1				1			
$\underline{e}_{11} =$					-2				1			
$\overline{e}_{12} =$								-1				1
$\underline{e}_{12} =$								-1				4

Table 2.2: Coordinates of the vectors

	$\overline{e}_1$	$\underline{e}_1$	$\overline{e}_2$	$\underline{e}_2$	$\overline{e}_3$	$\overline{e}_4$	$\underline{e}_4$	$\overline{e}_6$	$\underline{e}_6$	$\overline{e}_7$	$\overline{e}_9$	$\underline{e}_9$	$\overline{e}_{10}$	$\underline{e}_{10}$	$\overline{e}_{11}$	$\underline{e}_{11}$	$\overline{e}_{12}$	$\underline{e}_{12}$
$\underline{e}_3$	$-\frac{1}{2}$	$\frac{1}{2}$	2	-1	2			$\frac{3}{2}$	$-\frac{1}{2}$									
$\overline{e}_5$			-1			1												
$\underline{e}_5$				-1			1											
$\underline{e}_7$						3	-1	3	-1	3								
$\overline{e}_8$	$-\frac{1}{2}$	$\frac{1}{2}$				-1		$-\frac{3}{2}$	$\frac{1}{2}$		1		1	$-\frac{1}{2}$	1	$-\frac{1}{2}$	2	$-\frac{1}{2}$
$\underline{e}_8$	$-\frac{3}{2}$	$\frac{3}{2}$					-1	$-\frac{3}{2}$	$\frac{1}{2}$			1	4	-2	4	-2	4	-1

Table 2.3: The coefficient matrix

Chapter 3

Second part of the algorithm

As already mentioned, the second part of the algorithm will augment the matching by one pair, if possible using the coefficient matrix constructed in the first part. In this part it will be easier to denote the coefficients with a single letter indexed with the vectors, no matter whether they are upper or lower vectors, and even some new vectors will have no meaning in the original graph. So denote the coefficient of b basic vector for the a non-basic vector simply by $c_{a,b}$.

3.1 Dependence graph and search paths

The algorithm will create a *dependence graph* for the base B (where B consists of the matching M and some added upper vectors as in chapter 2), denoted by $D(B)$ which is a bipartite graph with one colour class of the basic vectors B and the other colour class consisting of the remaining vectors. Edges will be the swaps for this base, i.e., if $a \in E \setminus B, b \in B$ then $\{a, b\} \in E(D(B))$ if and only if $B \cup \{a\} \setminus \{b\}$ is again a base. This is equivalent to the condition that the corresponding coefficient is not zero, so the coefficient matrix is very similar to the adjacency matrix of $D(B)$. The reason why this auxiliary graph is also needed is that an order of a vector's neighbours can be efficiently maintained with it.

Constructing the dependence graph (the lists of neighbours for all vectors) from the coefficient matrix takes $O(mn)$ time since we have to go over the rows and columns of the matrix. It is also possible to construct this graph when calculating the coefficients, too.

Then the algorithm creates a system of *search paths.* These search paths will be simple (not self-intersecting) paths consisting of pairs of vectors and they always end in one of the vectors of $B \setminus M$. Furthermore the vectors of different pairs can be connected only if they form a swap. For example consider a path $P = aa'bb'cc'dd'$. Here a, b, c and d must be different, d' have to be an added upper vector of the base, so $d' \in B \setminus M$ and the pairs $a'b, b'c, c'd$ are swaps of the base B.

Only the last pair will be "mixed" (except in the final search path), other pairs will be matched pairs or nonbasic pairs and they will be alternatingly matched and non-matched because of swaps. The final search path will contain two mixed pairs, it will also start with a vector of $B \setminus M$, and therefore this will be called *augmenting path* since it contains one more matched pair than non-matched and two additional such mixed pair, so replacing the matched pairs in M with non-matched and mixed pairs along the path, we get a matching M' with one more pair in it. See [2] for a proof that M' is really a matching. If the algorithm can't find such an augmenting path then M is a maximum size matching.

The search paths will be stored recursively. The first vector in a vector's search path is the vector itself, say a, followed by its pair a' and then a vector b of an other pair. The label of a will point to this third vector b, but sometimes the search path will consist of two parts. In this case a second pointer will be defined pointing to a vector g, which has a search path containing the vector a. The first part will be the part of g's search path ending in a, but in reverse order, and the second part will be an other vector's whole search path, the first pointer of the label will point to this vector. Precisely the search paths $P(e)$ and labels of a vector e defined in the following way:

$P(e)$	label	type of e
ee'	$(\emptyset, \emptyset)$	only if e is a nonbasic vector of a mixed pair
$ee'P(f)$	$(f, \emptyset)$	usual case
$eP(g,e')^r P(f)$	(f, g)	if the path consists of two parts

where $P(g,e)$ denotes the part of $P(g)$ starting in g and ending in e and P^r denotes the reverse path of P.

During the algorithm a vector gets a label only when it has a well defined search path, so there will be no pointers referring to an unlabelled vector.

The next section introduces *transforms*, for these new vectors the search path will be very similar to the above cases, for example if T is a transform replacing the vector e, then $P(T)$ is defined as above except that e is replaced by T at the beginning of the search path, but the second vector will be again e', the mate of the replaced vector e.

3.2 Transforms, blossoms

Changing the vectors in a matching along a search path can produce a dependent set since changing a base, the swaps of the previous base are not necesseraly swaps in the new base. Therefore some correction needed, the algorithm will add new vectors to the dependence graph, called *transforms* and to the coefficient matrix, too. This transforms will replace a vector, and if an augmenting path contains transforms, they will considered to be the replaced vector, when augmenting the matching.

Each transform will have three parameters, and depending on them there are two ways of creating the transform $T_{e,f,g}$. First if e and f are non-basic vectors, and g is basic then $T_{e,f,g} := c_{e,g}f - c_{f,g}e$. This linear combination will be added to the coefficient matrix, so if $e = \sum_{b \in B} c_{e,b}b$ and $f = \sum_{b \in B} c_{f,b}b$ then the new coefficients are $\sum_{b \in B}(c_{e,g}c_{f,b} - c_{f,g}c_{e,b})b$. A new vertex $T_{e,f,g}$ will be added to the dependence graph, to the colour class of e and f, and it will be connected by an edge to those basic vectors, which has non-zero coefficient in the last linear combination.

This 2 by 2 determinant, $c_{e,g}c_{f,b} - c_{f,g}c_{e,b}$ we will be denoted simply by $c_{e,f,g,b}$.

In the second case, when e and f are basic vectors and g is non-basic, a new column will be added to the coefficient matrix representing $T_{e,f,g}$ with coefficients $-c_{g,a,e,f}/c_{g,f}$ in a's row (other formulas can also work). Also a vertex $T_{e,f,g}$ will be added to the dependence graph on the basic vectors' colour class and it will be connected by an edge to vectors a such that $c_{g,a,e,f}$ is not zero.

When the algorithm will create a transform, neither $c_{e,g}, c_{f,g}, c_{g,e}, c_{g,f}$ will be zero, so the transforms are well defined and they will be adjacent exactly to those vectors, for which $c_{e,f,g,b}$ or $c_{g,a,e,f}$ is non-zero. It's trivial that $T_{e,f,g}$ won't be adjecent to g. The mate of $T_{e,f,g}$ will be e', as we used it in search paths, but these new vectors will not be in the original matroid neither in the graph.

Of course if one of the parameters is a previously created transform then the corresponding elements of the coefficient matrix used.

Now we introduce *blossoms*, which are similar to the blossoms of graph matching algorithms. The algorithm will store an identifier $A(e)$ for all vectors e representing the set (the blossom) which e belongs to. In the beginning all $A(e) = \{e\}$, later these sets will be also disjoint and $e \in A(e)$, therefore if $f \in A(e)$ then $A(f) = A(e)$. When two vectors, a and b are adjacent in the dependence graph and their search paths intersect, i.e., they end in the same mixed pair, then a blossom found, denote it with simply A. Now we define the vectors in A. Let a_0 be the first vector of $P(a)$ starting from a such that $A(a_0)$ contains a vector of $P(b)$. Let b_0 be this vector of $P(b)$ (if there is more than one, choose the closest to b on $P(b)$), which is also the first vector of $P(b)$ such that $A(b_0)$ contains a vector of $P(a)$ since $A(a_0) = A(b_0)$.

If $a_0 = b_0 =: g$ (for example when the first blossom found) then let e and f be the vectors immediately before g in paths $P(a)$ and $P(b)$, the last not common vectors. In this case a transform $T_{e,f,g}$ will be created. Then the blossom will be

$$A := \{T_{e,f,g}\} \cup \bigcup_{\substack{h \in P(a,e),\\ h \in P(b,f)}} A(h).$$

e and f will be *tips* of A and g is the *bud* of A. Moreover if e or f is in a previous blossom, then the tips of $A(e)$ or $A(f)$ will be also tips of A.

If $a_0 \neq b_0$ then a_0 and b_0 is already in a previous blossom, and now

$$A := \bigcup_{\substack{h \in P(a,a_0),\\ h \in P(b,b_0)}} A(h).$$

In this case the algorithm doesn't create any transform. The bud and tips of A will be the bud and tips of $A(a_0) = A(b_0)$.

In both cases for all vectors $x \in A$, $A(x)$ will be updated to be A and later this blossom will be considered (more or less) as one unit. An *old tip* of A is a tip of a previous blossom in A which is not a tip of A.

For example consider figure 3.1, where thick edges are pairs, thin edges connects adjacent vectors in the dependence graph and the set A is the blossom. The mixed pair is the leftmost. g is the first common vector of search paths $P(a)$ and $P(b)$, it is also the bud of A, and the blossom A consists of the vectors of paths $P(a,e)$ and $P(b,f)$, finally the tips of A are e and f.

A special case of blossoms is when a blossom is equal to a pair of vectors only. This "degenerate blossom" case occurs when a vector (the bud of this blossom) is adjacent to both vectors of a pair.

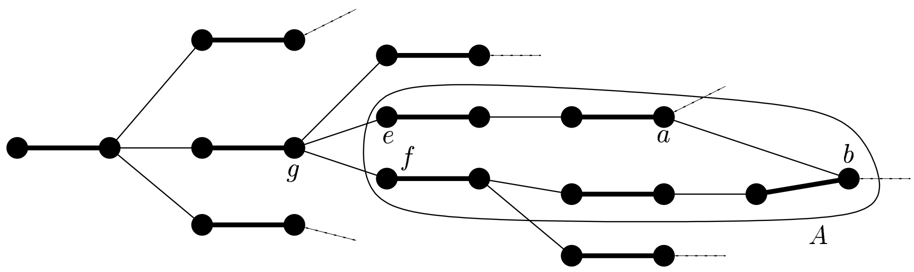

Figure 3.1: An example for blossom, tips and bud

3.3 The algorithm

Now we show how the second part of the algorithm works. It does a breadth-first search starting from the mixed pairs, enlarging the system of search paths by giving labels to unlabelled vectors. For keeping the right order of scanning the vectors, the algorithm stores a *serial number*, $s(e)$ for all vectors e, and a queue Q, to put into it the labelled vectors and read the next vector to be scanned from it. Serial numbers will be initially infinity (or a number big enough), the first labelled vector gets the serial number 1 and then, other vectors always get the last serial number plus 1. When labelling a vector, it not only gets two pointers, defining its search path as in section 3.1, but the algorithm adds this vector to the end of queue, and the vector gets a serial number (the next integer), too. Therefore exactly the labelled vectors have finite serial numbers. Furthermore, to speed up the algorithm, the edgelist of the dependence graph should be updated, to contain vectors in order of increasing serial numbers.

The following text can be read parallel with the pseudo code starting on page 24, which illustrates the structure of the algorithm. Some functions are not described here because it is the task of implementing the data structures. First(Q) removes the first element from the queue Q and returns this vector. Last(Q, e) adds the vector e to the end of queue Q. The algorithm creates transforms as in section 3.2. Stop ends the algorithm. In the procedure Augment we can also update the forest F of chapter 2, it's easier to do it here than later. Label with capital "L" is the procedure with three parameters, label(e) in lower case is the label itself, defining the search path for the vector e.

First the algorithm makes some initialization, some of them were also mentioned before. This includes initializing serial numbers, blossoms, the queue, the dependence graph. Then it labels the non-basic vectors of mixed pairs. They will be the ending pairs of all search paths, so it's clear that they should form the initial search path system. The labelling procedure (its code is after the main routine) does exactly what we've just discussed in the beginning of this section.

Then the algorithm starts scanning the vectors. Takes the next vector a to scan from the queue and for all vector b, adjacent to a and isn't in the same blossom does one of the following:

- If b is already labelled and it is also scanned, then an *augmenting step* or *blossom step* is done depending on whether $P(a)$ and $P(b)$ are disjoint or not. The augmenting step is done by the procedure Augment, it produces the matching M' from M by changing its pairs along the two search paths. Then it stops, since a bigger matching has been found.

The blossom step is made by procedure Blossom, it follows section 3.2. e and f are the new tips. The algorithm labels unlabelled elements of the blossom, which are not in a blossom, or is a tip of a previous blossom. New tips are not labelled. The order of labelling is just like they were labelled by growing steps continuing the search paths on the other side of the blossom. Tranform if necessary, created, too, and it also gets a label. Finally the blossoms, tips are updated.

- If b is not labelled but b', the pair of b is labelled, then a search path is already defined for b', there is no need for other, so the algorithm does nothing.
- If neither b, nor b' is labelled, and b' is not adjacent to a, then a search path can be defined for b', so the algorithm labels it by $(a, \emptyset)$. This is a *grow step.*
- Finally, if neither b, nor b' is labelled but b' is also adjacent to a then the *degenerate blossom step* occurs. Nothing is labelled, except the transform, which is also created. Blossoms and tips again updated.

If finished, the algorithm takes the next vector and so on. When the queue is empty, it stops, and the system of the search paths cannot be enlarged, which means that the matching M is maximal.

Main $\{$*the main routine*$\}$
for $e \in E$ **do** $s(e) := \infty$; $A(e) := \{e\}$ **end for**
$Q := \emptyset$
$s := 0$
initizalize D to $D(B)$
for e is a non-basic (lower) vector of a mixed pair **do** Label$(e, \emptyset, \emptyset)$ **end for**
while $Q \neq \emptyset$ **do**
    $a :=$ First(Q)
    for b adjacent to a in D **and** $A(a) \neq A(b)$ **and** in order of increasing $s(b)$ **do**
        if $s(b) < s(a)$ **then**
            let a_0 be the first vector of $P(a)$ such that $A(a_0) \cap P(b) \neq \emptyset$
            if $a_0 = \emptyset$ **then** Augment(a, b) **else** Blossom(a, b, a_0) **end if**
        elseif $s(b) = \infty$ **and** $s(b') = \infty$ **and** $A(b) = \{b\}$ **then**
            if b' is adjacent to a in D **then**
                $\{$*degenerate blossom step*$\}$
                create $T_{b,b',a}$ and add it to D
                Label$(T_{b,b',a}, a, \emptyset)$
                $A(b) := \{b, b', T_{b,b',a}\}$
                $A(b') := A(b)$
                $A(T_{b,b',a}) := A(b)$
                mark b, b' as tips
            else Label$(b', a, \emptyset)$
            end if

end if
end for
end while
Output $M \subset B$ *is maximum*
end Main

procedure Label(e, f, g)
$s := s + 1$
$s(e) := s$
label$(e) := (f, g)$
Last(Q, e)
for a adjacent to e **do**
sort the edgelist of a in D
end for
end Label

procedure Blossom(a, b, a_0)
let b_0 be the first vector of $P(b)$ in $A(a_0)$
if $a_0 = b_0$ **then** let e and f be the vectors immediately before a_0 in $P(a)$ and $P(b)$
else $e := \emptyset$; $f := \emptyset$ **end if**
for $g \in P(a, a_0) \cup P(b, b_0) \setminus \{e, f\}$ **and** $s(g) = \infty$ **and** $(A(g) = \{g\}$ **or** g is a tip$)$
and in order of decreasing $s(g')$ **do**
if $g \in P(a, a_0)$ **then** Label(g, b, a)
else Label(g, a, b) **end if**
end for
if $e \neq \emptyset$ **then**
create T_{e,f,a_0} and add it to D
Label(T_{e,f,a_0}, b, a)
end if
update $A(e)$ for e in this new blossom and mark any new tips
end Blossom

procedure Augment(a, b)
$M' :=$ the matching $M \subset B$
Path$(a, \emptyset)$
Path$(b, \emptyset)$
Output M'
Stop
end Augment

procedure Path(*start*, *finish*)
if *start* $\neq$ *finish* **then**
if *start* is a transform $T_{x,y,z}$ **then** $g := x$ **else** $g :=$ *start* **end if**
$M' := M' \triangle \{g, g'\}$

(*back*, *reverse*) := label(*start*)
if *reverse* $\neq \emptyset$ **then** Path(*reverse*, g') **end if**
Path(*back*, *finish*)
end if
end Path

Now here are some words about how to maintain the order of increasing serial numbers in the dependence graph. First every vector has infinity as serial number, so the order is not relevant. We can add a pointer to every vectors' edgelist pointing to the last labelled vector in the list. When a vector e gets labelled, in its neighbours' edgelist we can insert e after the last labelled vector, before the unlabelled vectors and set the pointer to point e. This takes constant time, if we don't delete e from the unlabelled part of the list. So the labelling takes $O(n)$ time for non-basic vectors (the size of the base with basic transforms), and $O(m)$ time for basic vectors. When reading the list (scanning that vector) it doesn't take extra time to "remember" the labelled vectors which are at the beginning of the list, in order to avoid taking twice them when they appear among the unlabelled vectors, too. The serial numbers are increasing, so inserting the labelled vector to the end of the labelled vectors always gives the right order of serial numbers.

3.4 Efficiency

The efficiency of the second part of the algorithm will be $O(mn)$, too.

First we show that there are $O(n)$ transforms on the basic side created by the algorithm and $O(m)$ on the non-basic side. This fact we already used a few lines above. The number of basic transforms is bounded by the number of blossoms containing matched vectors, since all blossom step, creating a basic transform decreases them by at least one. Initially there was $|M|$ such blossom, so there are $O(n)$ basic transforms. Similar reason shows that there are $O(m)$ non-basic transforms. As a consequence, there are $O(m)$ non-basic, and $O(n)$ basic vectors in the dependence graph, so it has $O(mn)$ edges.

Next we show that the labelling steps take $O(mn)$ in total. We already mentioned that this takes $O(n)$ time for non-basic vectors and $O(m)$ for basic vectors. Since there are (together with transforms) $O(m)$ non-basic and $O(n)$ basic vectors and all of them gets labelled at most once the total time required for labelling is $O(mn)$.

The main routine without blossom steps takes $O(mn)$ time. The initializations take trivially $O(mn)$, the time of scanning the vectors is proportional to the number of edges in the dependence graph, which is also $O(mn)$.

The blossom steps without creating the transforms and updating the blossoms take time proportional to the length of $P(a)$ and $P(b)$. A search path contains alternatingly matched and non-matched pairs, hence it is $O(n)$ long. There are $O(m)$ blossom steps since each one merges together at least two blossoms and initially there are also $O(m)$ of them.

Creating a transform takes $O(n)$ time for non-basic transforms, and $O(m)$ for basic transforms. So in total they can be created in $O(mn)$ time.

Finally, updating the blossoms is a disjoint set merging problem. There is $O(mn)$ find operations (accessing the value $A(e)$ for any e) and $O(m+n)$ unions (in the blossom steps).

Using the algorithm of [3], this can be done in $O(mn)$.

So in total everything can be done in $O(mn)$.

Chapter 4

Comments, future research

4.1 The forest F after augmenting

The algorithm starts with a forest containing a tree with two vertices in T and all other components being single vertices of T. An augmenting path found by the second part starts and ends in a mixed pair of vectors, but it doesn't meet any more since they are all labelled at the beginning and therefore also scanned in the beginning, furthermore there are dependence graph edges only from the non-basic vectors, as we mentioned this at end of section 2.2, basic vectors of mixed pairs are adjecent only to their pair.

So only two trivial components will be affected, the edge connecting them to the rest of G will be added to F (this corresponds to the addition of the two mixed pairs to the matching). Changing the matching along the augmenting path also means that the matching will have one more pair, and the forest F can have one more edge only if it has less components. Since the new M is a matching, too, again all components of the corresponding forest F has at most two vertices in T. This shows that the two affected components will be in one tree or they can be in different ones, but all components, not equal to a (non-affected) single vertex of T again will have two vertices in T. This is the structure we need for the first part of the algorithm to work.

Since the basic vectors of mixed pairs is adjacent only to their pairs, we can consider mixed pairs being non-basic pairs, and delete the last columns of the coefficient matrix corresponding to non-matched basic vectors. The role of them in the second part of the algorithm is only being the pair of non-basic vectors, which are labelled first.

4.2 Improvement

I don't think the running time of the algorithm can be improved easily. One reason for this is that the dependence graph can have cmn edges with some c constant, so even reading it takes $O(mn)$ time. The only part where I can imagine some improvement is between steps. When augmenting the matching, with smart handling of data, maybe there will be no need to recalculate every search paths from the beginning, and later augmentations can be achieved faster than we did. But I'm afraid it is not really a possibility since a longer augmenting path can modify such a big part of the matching that most of the calculations have to be calculated again.

Certainly this doesn't mean that there is no better algorithm. It can have a totally different idea behind it, and it can use also different structures, of course.

However some simplifications may arise. For example if taking some special tree or forest at the beginning of the whole algorithm can maybe avoid blossom steps. Degenerate blossom steps seem to be very usual cases, but large blossoms are relatively rare. If so, the second part of the algorithm will be much simpler and the proof of correctness also becomes much shorter. For testing this and also in general it would be good to implement the algorithm on a computer. I'm planning to do it in the future.

4.3 Other cases of the problem

- The edge-disjoint case follows from the vertex-disjoint case. Again we can assume that all T_i set is merged to a single point, i.e., we have a set T and we're interested in the maximum number of edge-disjoint T-paths in a graph G.

  Construct G' as follows. Take a vertex for all edges of G, and connect two vertices in G' if and only if the corresponding edges in G have a common endpoint. Then let $T_i \subseteq V(G')$ be the set of vertices corresponding to edges ending in $t_i \in T \subseteq V(G)$. So a path between t_i and t_j in graph G is a path between T_i and T_j in graph G'. If two such path have no common edges in G then the corresponding paths have no common vertices in G' and vice versa, so the maximum number of edge-disjoint T-paths in G corresponds to the maximum number of disjoint $\mathcal{T}$-paths in G'.

- One can study a possible generalization of $\mathcal{T}$-paths' problem, i.e., if transitivity doesn't hold in subsection 1.1.3, so only given pairs are not allowed to be connected by a T-path. In this case the problem seems to be much harder.

- An other way to generalize the problem is to define capacities on vertices or on edges of the graph, and ask for the maximum level of "traffic" that can be transferred not exceeding the capacities from any set T_i to any other set T_j. This kind of generalization looks like a network flow problem.

Bibliography

[1] D. Coppersmith, S. Winograd, *Matrix multiplication via arithmetic progressions,* Journal of Symbolic Computation 9 (1990) 251–280; Proceedings of the 19th Annual ACM Symposium on Theory of Computing (1987) 1–6.

[2] H. N. Gabow, M. Stallmann, *An augmenting path algorithm for linear matroid parity,* Combinatorica 6 (1986) 123–150.

[3] H. N. Gabow, R. E. Tarjan, *A linear-time algorithm for a special case of disjoint set union,* Journal of Computer and System Sciences 30 (1985) 209–221.

[4] P. Jensen, B. Korte, *Complexity of matroid property algorithms,* SIAM Journal on Computing 11 (1982) 184–190.

[5] L. Lovász, *The matroid matching problem,* Algebraic Methods in Graph Theory, Colloquia Mathematica Societatis János Bolyai, Szeged, Hungary (1978)

[6] J. B. Orlin, J. H. Vande Vate, *Solving the linear matroid parity problem as a sequence of matroid intersection problems,* Mathematical Programming 47 (1990) 81–106.

[7] A. Schrijver, *Combinatorial Optimization – Polyhedra and Efficiency*, (forthcoming book, contents: http://www.cwi.nl/~lex/files/begin.ps) Chapter 43.

[8] A. Schrijver, *Combinatorial Optimization – Polyhedra and Efficiency*, Chapter 73.

[9] A. Schrijver, *A short proof of Mader's $\mathcal{S}$-paths theorem,* Journal of Combinatiorial Theory, Series B 82 (2001) 319–321.