Finding one of many Disjoint Perfect Matchings in a Bipartite Graph

Martin Löhnertz

Institute for Computer Science V University of Bonn

Abstract. We demonstrate how to find a perfect matching in a bipartite graph containing $\sqrt{n}\sigma^3$ disjoint perfect matchings in time $\mathcal{O}(\sqrt{n}m/\sigma)$.

1 Introduction

1.1 Overview

The problem of finding a perfect (or a maximum) matching in a bipartite graph is one of the best known "classical" problems in graph theory. We identify a new special case of this problem which can be solved faster than the general case. Similar to the fact that it is easier to find a needle in a haystack containing many needles we show that a perfect matching can be found faster if there are many disjoint perfect matchings in the graph.
Several algorithms have been developed for the general bipartite perfect matching problem. The best known ones are the algorithm of Hopcraft and Karp [6] which has a complexity of $\mathcal{O}(\sqrt{n}m)$ and the much more involved algorithm of Feder and Motwani [5] with complexity $\mathcal{O}(\sqrt{n}m\frac{\log 2n^2/m}{\log n})$.

For the regular case there exist faster algorithms: The algorithm of Cole and Hopcroft [1] with complexity $\mathcal{O}(m + n\log^3 n)$ and the algorithm of Cole, Ost and Schirra [2] with complexity $\mathcal{O}(m)$.
The central idea of our method is to find a regular subgraph of a general graph in order to apply the latter methods to this subgraph.

The main result of this paper is

Theorem 1. *Let G be a bipartite graph with $2n$ vertices and m edges containing $\sqrt{n}\sigma^3$ pairwise disjoint perfect matchings. Then there is an algorithm that finds a single perfect matching in time $\mathcal{O}(\frac{\sqrt{n}m}{\sigma})$.*

As σ may depend on n e.g. in graphs containing kn disjoint perfect matchings for a constant k a perfect matching can be found in time $\mathcal{O}(\sqrt[3]{n}m)$.

The algorithm consists of the following steps which will be explained in more detail in the following sections:

1. transform the graph into a network
2. find a large flow in this network
3. construct an almost regular subgraph from the edges carrying flow
4. embed this graph in a slightly larger regular graph by adding some vertices and edges
5. find a matching in this regular graph
6. remove matching edges not belonging to the original graph
7. embed the found matching into the original graph
8. augment the matching to a perfect matching

We will describe steps 1 to 3 in section 2 and steps 4 to 8 in section 3.

1.2 Notation

A **graph** $G = (V, E)$ consists of a set of vertices V and a set of edges $E \subset V \times V$. An edge $\{a, b\}$ is said to be **incident** to the vertices a and b. a and b are **adjacent** if there is an edge $\{a, b\}$. The **degree** of a vertex $deg(a)$ is the number of incident edges. A **subgraph** H of G is a graph $H = (V', E')$ with $V' \subset V$ and $E' \subset E$. The degree of v in H, $deg_H(v)$ is the number of edges incident to v contained in H.

Two edges e_1, e_2 are said to be **independent**, if $e_1 \cap e_2 = \emptyset$. A **matching** in G is a set M of independent edges and a maximum matching is a matching M with $|M| \geq |M'|$ for all matchings M' in G.

A graph is said to be **bipartite** if V can be split in two disjoint sets $V = A \dot{\cup} B$ and all edges contain vertices from both sets, i.e. $E \subset A \times B$.
A graph is **regular** if all vertices have the same degree. We will only consider bipartite graphs with $|A| = |B|$ and will denote $|A|$ by n, implicating that the number of vertices is actually $2n$ but this constant factor will not influence the asymptotic complexity. As usually we denote the number of edges by m.

2 Finding an Almost Regular Subgraph

Definition 1. *An almost (n, σ)-regular subgraph of a bipartite graph $G = (V, E)$ is a subgraph H with $\deg_H(v) \leq \sqrt{n}\sigma^3$ $\forall v \in V$ and $\sum_{v \in V}(\sqrt{n}\sigma^3 - deg_H(v)) \leq 2n\sigma^2$*

The critical property of an almost (n, σ)-regular graph is the relation between maximum degree and the number of edges missing to make the graph regular. We will model this by a flow in a network in which almost the same large flow passes through each vertex.

2.1 Transformation to a Unit Capacity Network

Let $G = (A \dot{\cup} B, E)$, $|A| = |B| = n$, $V = A \cup B$ be a bipartite graph with $\sqrt{n}\sigma^3$ disjoint perfect matchings. Our transformation is similar to the well known "direct" transformation of the matching problem into a flow problem [7]. The only difference will be a change in some edge capacities. Note that in our network we will assume the edges to be directed, i.e. an edge (a, b) has a "tail" (a) and a "head" (b) and we assume that flow flows from the tail to the head. A network is a directed graph with two special vertices, a source s and a sink t, together with a function $c : E \to \mathbb{R}$. The underlying intuition is than one wants to send flow from s to t without sending more than $c(e)$ flow across an edge e. A network is an unit capacity network if $c(e) = 1$ for all edges.

A network N with capacities is created from G by directing all edges from A to B, adding a source node s and a sink node t and edges (s, a) for all $a \in A$ and (b, t) for all $b \in B$. For each edge $e \in G$ we set the capacity $c(e)$ to 1 and to $c := \sqrt{n}\sigma^3$ for all new edges. The maximum flow through this network will have size $n\sqrt{n}\sigma^3$ as each set of edges forming a matching will allow to transfer n units of flow.
To transform this network to a unit capacity network we replace each edge with capacity c by c parallel edges, each with capacity 1. Note that $2nc < 2m$,

as there have to be at least cn edges in G that form the matchings. Therefore $m+2nc \in \mathcal{O}(m)$ and we will continue to speak of m indifferent whether meaning m or $m+2nc$. Each set of parallel edges will be called a "multi-edge". The other edges will be called "simple" edges.

2.2 Finding a Large Flow

We start with a short repetition of blocking flow algorithms. A flow in our unit capacity network is a function $f : E \to \{0,1\}$ which associates a flow value with each directed edge. A flow f must fulfill the flow conservation constraints $\sum_{(u,v)\in E} f((u,v)) = \sum_{(v,w)\in E} f((v,w)) \; \forall v \in V\backslash\{s,t\}$. Our aim is to find a flow for which $\sum_{(s,v)\in E} f((s,v)) = \sum_{(u,t)\in E} f((u,t))$ is large.
For the unit capacity case we can define the so called residual network $N_f = (V, E_f)$ the following way: The vertices are the same as in N and for $(u,v) \in E$ we have $(u,v) \in E_f$ iff $f((u,v)) = 0$ and $(v,u) \in E_f$ iff $f((u,v)) = 1$. A directed path from s to t in the residual network corresponds to one possible augmentation of the current flow by sending one more unit of flow along this path.

Hopcroft and Karp [6] have observed, that one should augment along shortest paths in this network. This was modified by Dinic [3] to the "method of blocking flows".

By labeling the vertices of N_f by their distance from s in N_f, which can by done by breath-first-search in $\mathcal{O}(m)$, one can identify these shortest paths. The algorithm of Dinic [3] uses these in a different way. The labels partition the graph in several levels. Every edge in the residual network starting at level l must lead to a level $\leq l+1$ by construction. As every augmenting path of shortest length must cross each level exactly once one can reduce the graph to those edges leading to subsequent levels. One then calculates a maximal set of paths in this reduced graph (a blocking flow) and augments along them. As each subsequent augmenting path must still pass all levels and take at least one edge not leading to a subsequent level, the distance between s and t increases with each iteration of the whole procedure.

The above description contains an additional insight: As each augmenting path has to cross an edge leading from level i to level $i+1$ for all i, these edges form a directed cut in the residual network, i.e. a set of edges intersecting each directed path from s to t in N_f. For each i the capacity of the level i to level $i+1$ cut is an upper bound for the additional flow which still can be pushed from s to t. As noted by Even and Tarjan [4] the blocking flow method of Dinic also works for multi-graphs and has a complexity of $\mathcal{O}(m)$ per phase as each edge will be traversed at most twice.

2.3 Existence of a Small Cut

Even and Tarjan [4] observed that one can find a flow and a cut in the corresponding residual network of capacity $\left(\frac{n}{d}\right)^2$ in a simple unit-edge-capacity network in

$\mathcal{O}(dm)$ steps. We extend their argument to networks which have multi-edges incident to the source or the sink by observing that the cut constructed by their method does not contain any multi-edges. The network constructed in the previous section has this property.

After $2+4\sqrt{n}/\sigma$ phases of Dinic's algorithm[1] the $s-t$ distance will be greater or equal $2+4\sqrt{n}/\sigma$. Let N_f be the residual network. s defines level 0 and t will be at a level larger or equal $2+4\sqrt{n}/\sigma$. So there are $4\sqrt{n}/\sigma$ subsequent levels not containing s or t. Assume they are partitioned in $2\sqrt{n}/\sigma$ pairs of subsequent levels (Fig. 1).

Proposition 1. *There is one pair containing at most $\sigma\sqrt{n}$ vertices.*

Proof: Assume every pair contains more than $\sigma\sqrt{n}$ vertices. Then there are more than $\sigma\sqrt{n}\; 2\sqrt{n}/\sigma = 2n$ vertices in the graph. Contradiction.
Especially each of the two levels will have at most $\sigma\sqrt{n}$ vertices. Let these levels be l_i and l_{i+1}

Proposition 2. *All edges leading from level l_i to level l_{i+1} are single edges*

Proof: Neither s nor t belong to l_i or l_{i+1} while all multi-edges are incident to s or t.

Proposition 3. *The edges directed from l_i to l_{i+1} form a directed $s-t$ cut in the residual network N_f with capacity $n\sigma^2$.*

Proof: There are at most $\sqrt{n}\sigma * \sqrt{n}\sigma$ edges leading from l_i to l_{i+1} and each of these edges has unit capacity. So the capacity of the set of these edges is at most $n\sigma^2$.

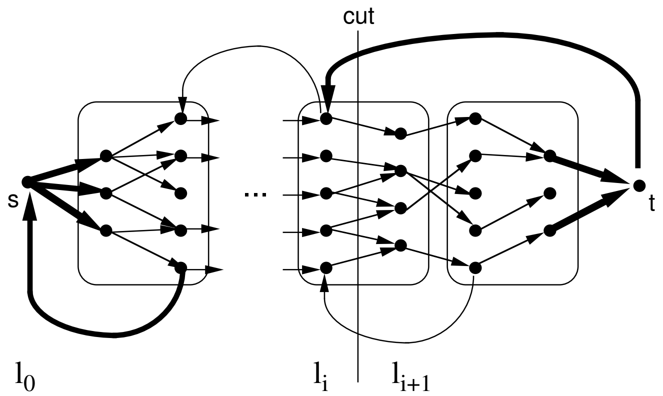

Fig. 1. Residual network with cut. Thick arcs represent multi-edges

[1] If the algorithm of Dinic stops before, we immediately continue with the next step of our algorithm.

Proposition 4. *The edges carrying flow in N_f which are not connected to s or t form an almost (n, σ)-regular subgraph H of G*

Proof: The capacity of the $\sqrt{n}\sigma^3$ edges entering a vertex $v \in A$ is 1, so for each vertex we have $\sum_{(s,v)\in E} f((s,v)) \leq \sqrt{n}\sigma^3$. Due to the flow conservation constraints we have $\sum_{(v,b)\in E} f((v,b)) = \sum_{(s,v)\in E} f((s,v))$. As each flow carrying edge leaving v carries exactly one unit of flow this also equals $\sum_{(v,b)\in E, f(v,b)=1} 1$ which equals the degree of v in the subgraph created by these edges.

On the other hand the total flow f_t is at least $n\sqrt{n}\sigma^3 - n\sigma^2$, as there could be a flow of $n\sqrt{n}\sigma^3$ through the edges of the disjoint perfect matchings and the current flow can at most be augmented by $n\sigma^2$ units of flow. The flow leaving s (resp. entering t) is t_f and as each unit of flow crosses exactly one edge from A to B there must be the same number of edges leaving the vertices of A implying

$$n\sqrt{n}\sigma^3 - n\sigma^2 \leq f_t = \sum_{v\in A} \sum_{(v,b)\in E, f((v,b))=1} 1 = \sum_{v\in A} deg_H(v)$$

The same argument applies to the vertices in B by exchanging incoming and outgoing edges in the proof. Therefore $\sum_{v\in G} deg_H(v) \geq 2n\sqrt{n}\sigma^3 - 2n\sigma^2 \Leftrightarrow \sum_{v\in G}(\sqrt{n}\sigma^3 - deg_H(v)) \leq 2n\sigma^2$.
Using that $2 + 4\sqrt{n}/\sigma$ phases of the blocking flow algorithm can be performed in $\mathcal{O}(\sqrt{n}m/\sigma)$ we get

Lemma 1. *An almost (n, σ)-regular subgraph can be found in a bipartite graph containing $\sigma^3\sqrt{n}$ disjoint perfect matchings in $\mathcal{O}(\sqrt{n}m/\sigma)$.*

3 Regularization and Construction of the Matching

Let H be the almost (n, σ)-regular subgraph constructed in the section 2. Let $d := 2n\sigma^3\sqrt{n} - \sum_{v\in H} \deg(v)$ be the missing vertex-degree. We now want to change H into a $\sqrt{n}\sigma^3$ regular multigraph. We do this by adding some additional vertices. Each of these can be incident to $\sigma^3\sqrt{n}$ additional edges. The other ends of these edges can be connected to the "original" vertices, increasing their degree. The sum of the increases required is d. So we need at most $\alpha := \lceil \frac{d}{\sigma^3\sqrt{n}} \rceil \leq \frac{2n\sigma^2}{\sigma^3\sqrt{n}} + 1 = \frac{2\sqrt{n}}{\sigma} + 1$ additional vertices. We insert $\lceil \alpha/2 \rceil$ vertices to A and the same number of vertices to B. Let $A'' \subset A$ and $B'' \subset B$ denote the sets of additional vertices. We connect vertices from $A \backslash A''$ with degree smaller than $\sqrt{n}\sigma^3$ with multi-edges to the first vertex from B'' until it has reached that degree and then continue to do so with the other vertices from B'' until all vertices of $A \backslash A''$ have degree $\sqrt{n}\sigma^3$. We do the same for B and A''. Due to the number of additional vertices chosen in the end there will be at most one vertex in A'' and one vertex in B'' having a degree smaller than $\sqrt{n}\sigma^3$. But as all other vertices have degree $\sqrt{n}\sigma^3$ and the sum of degrees in A must equal the sum of degrees in B these two will have the same degree γ and are connected to each other by $\sqrt{n}\sigma^3 - \gamma$ multi-edges.

3.1 Finding a Matching in the Regular Supergraph

Every regular bipartite graph contains a perfect matching. It can be found in time $\mathcal{O}(m)$ using the algorithm of Cole, Ost and Schirra [2] or the algorithm of Cole and Hopcroft [1] (as the additional summand of the complexity of this algorithm is dominated by m for the graphs considered here). In fact it suffices to apply the first part of these algorithms which is identical and reduces the number of edges in the graph to $\mathcal{O}(n \log n)$. One can use any efficient matching algorithm to find a matching in this reduced graph as the complexity of this is dominated by the complexity of the other steps of our algorithm. We will give a short sketch of this first part in order to explain why this also works for multigraphs.

The algorithm is based on the fact that a r-regular bipartite multigraph $G = (A\dot{\cup}B, E)$ always contains a perfect matching, and the underlying simple graph, i.e. the graph created by replacing multi-edges by simple edges, contains a perfect matching, too. This can be seen by an application of the pigeonhole principle: Let X be a subset of A then $r|X|$ edges are incident to X and at least $|X|$ vertices in B are needed to form the other ends of these edges. So the number of vertices adjacent to X denoted by $|\Gamma(X)|$ is at least as large as $|X|$. This is the Hall condition [7] guaranteeing the existence of a perfect matching covering $|A|$.
The idea of the algorithm of Cole and Hopcroft therefore is to modify the r-regular graph in a way that keeps it r-regular while making the underlying simple graph smaller. The base operation is to take any even length cycle in the graph and to replace each second edge with two edges and to delete every first edge, keeping the degree constant while deleting half the edges of that cycle. This is done for all cycles in a maximal cycle partition, then repeated for all cycles in a maximum cycle partition of the underlying simple graph of the double edges inserting quadruple edges and so on. As every maximum cycle partition has at least $m - (n - 1)$ edges and this procedure is applied at most $\log r$ times the underlying simple graph will have $\mathcal{O}(n \log r)$ edges in the end. In each step the number of edges under consideration is halved, and each iteration takes $\mathcal{O}(m)$ steps. Therefore the total complexity of this is $\mathcal{O}(\sum_{i=0}^{\log r} \frac{1}{2}^i m) = \mathcal{O}(m)$.
So all we have to show is than one can find a maximum cycle partition in a multigraph in $\mathcal{O}(m)$, as [1] only apply this to simple graphs. In fact one can apply their method without modification. This simplest method to see this is to replace each edge by a path of length 2 (two edges and a vertex incident to both) transforming the graph into a simple graph with $2m$ edges. A cycle partition of this graph will correspond to a cycle partition of the original graph and can be found in $\mathcal{O}(2m) = \mathcal{O}(m)$ by depth-first-search (DFS) as described by Cole and Hopcroft [1].

3.2 Embedding the Matching into the Original Graph

At most $2\sqrt{n}/\sigma + 1$ edges of the matching found in the previous section can be incident to the additional vertices. When removing them one deletes at most the same number of matching edges, resulting in a matching with at least $n - 2\sqrt{n}/\sigma - 1$ edges. These edges all belong to the original graph and therefore form a matching of the same size in G. Now we have to enlarge this matching by at most $\mathcal{O}(\sqrt{n}/\sigma)$.

It is known from the theorem of Berge [7] , that if a matching is non-maximal there is a so called "augmenting alternating" path, i.e. a path starting in a vertex not covered by the matching, then using alternatively matching and non-matching edges and ending in a different vertex not incident to a matching edge. Inserting all non-matched edges of this path into the matching and removing the matching edges contained in the path from the matching gives a new matching which is larger than the original. This can be repeated until a maximum matching, in our case a perfect matching, has been constructed. An augmenting path can be found in a bipartite graph by slightly modified DFS (see e.g. [6]) in time $\mathcal{O}(m)$. So by doing $\mathcal{O}(\sqrt{n}/\sigma)$ searches for an alternating path each of complexity $\mathcal{O}(m)$ the found matching can be changed into a perfect matching in G in time $\mathcal{O}(\sqrt{n}m/\sigma)$.

4 Remarks

1. We give a short remark on the "optimal" case. How large can σ as a function of n be ? A bipartite graph with $2n$ vertices can contain at most n disjoint perfect matchings. So in the best case σ^3 can be of size $k\sqrt{n}$ for some constant $k \leq 1$. This yields $\sigma = \sqrt[3]{k}\sqrt[6]{n}$, which results in an $\mathcal{O}(\sqrt{n}m/\sqrt[6]{n}) = \mathcal{O}(\sqrt[3]{n}m)$ algorithm.
2. If one does not know σ beforehand a σ' with $\frac{1}{2}\sigma \leq \sigma'$ can be found by binary search. The search starts with $\sigma' := \sqrt[3]{\frac{\min_{v \in G} deg(v)}{\sqrt{n}}}$. This value is an upper bound for σ as the minimum degree of each vertex is larger than the number of disjoint perfect matchings in the graph. Then the algorithm described above is run with this parameter until the final flow has been constructed. If the graph induced by the flow carrying edges is almost (n, σ)-regular we perform the other steps of the algorithm and are done. Otherwise σ' is set to $\sigma'/2$ and the procedure is repeated. Each of the intermediate executions of the algorithm has a lower complexity than the last iteration, as σ' is decreasing monotonously. There can be at most $\log \sqrt[3]{n} \leq \log n$ iterations.
3. Throughout this paper we have used $\sqrt{n}\sigma$ and $\sqrt{n}\sigma^3$ like integral numbers for simplicity. In fact these have to be rounded - depending on the context - to the nearest greater or smaller integral number. This does not affect the correctness of the proofs.

References

[1] R. Cole and J. Hopcroft. On edge coloring bipartite graphs. *Siam J. Comput.*, 11, 1982.

[2] R. Cole, K. Ost, and S. Schirra. Edge-coloring bipartite multigraphs in $O(E \log D)$ time. Technical Report TR1999-792, New York University, 21, 1999.

[3] E. A.: Dinic. Algorithm for solution of a problem of maximum flow. *Soviet Mathematics Doklady*, 11, 1970.

[4] S. Even and R.E. Tarjan. Network flow and testing graph connectivity. *Siam J. Comput.*, 4, 1975.

[5] Tomás Feder and Rajeev Motwani. Clique partitions, graph compression and speeding-up algorithms. In *STOC proceedings*, pages 123–133, 1991.

[6] J. E. Hopcroft and R. M. Karp. A $n^{5/2}$ algorithm for maximum matching in bipartite graphs. *Siam J. Comput.*, 2, 1973.

[7] László Lovász and Michael D. Plummer. *Matching Theory*. Akadémiai Kiadó, North-Holland Publishing, 1986.