9

Linear Time Approximation Algorithms for Degree Constrained Subgraph Problems

Stefan Hougardy

Summary. Many real-world problems require graphs of such large size that polynomial time algorithms are too costly as soon as their runtime is superlinear. Examples include problems in VLSI-design or problems in bioinformatics. For such problems the question arises: What is the best solution that can be obtained in linear time? We survey linear time approximation algorithms for some classical problems from combinatorial optimization, e.g. matchings and branchings.

9.1 Introduction

For many combinatorial optimization problems arising from real-world applications, efficient, i.e., polynomial time algorithms are known for computing an optimum solution. However, there exist several applications for which the input size can easily exceed 10^9. In such cases polynomial time algorithms with a runtime that is quadratic or even higher are much too slow. It is therefore desirable to have faster algorithms that not necessarily find an optimum solution.

An *approximation algorithm* for a combinatorial optimization problem is an algorithm that for any possible input returns some feasible solution. An approximation algorithm has an *approximation ratio* of c if for any input it returns a solution with value at least c times the value of an optimum solution (in this paper we will consider maximization problems only).

For most reasonable problems the lowest possible runtime for a deterministic algorithm is linear, as at least the whole input must be read. In this paper we are interested in approximation algorithms that achieve this linear runtime. Moreover, we are interested here only in approximation algorithms that achieve a constant approximation ratio. The reason for the latter requirement is that in practice solutions that are far away from an optimum solution are quite useless. Thus, even an approximation ratio of $1/2$ may be too bad for a given application. However, as the approximation ratio is a guarantee for the worst case, in practice approximation algorithms with

constant approximation ratios usually deliver solutions that are very close to the optimum. For example the greedy algorithm for the MAXIMUM WEIGHT MATCHING PROBLEM has an approximation ratio of $1/2$ but its solutions are usually within 5% of the optimum solution (Drake and Hougardy 2003a).

Linear time approximation algorithms offer several benefits against exact algorithms.

1. The most obvious benefit is its runtime: Exact algorithms even if their runtime is polynomial may simply be too slow to be applicable.
2. Linear time approximation algorithms are usually simpler than their exact counterparts. This not only means that the algorithms are simpler, but also their proofs of correctness may be simpler.
3. The implementation of linear time approximation algorithms can be much simpler than for exact algorithms. This is the main reason why in many applications approximation algorithms are used, even though exact algorithms would be fast enough (see for example Avis 1978).
4. There is another major reason, why in many applications approximation algorithms are used for combinatorial optimization problems even though exact algorithms would be fast enough: If the algorithm is used as a subroutine in some heuristic that does not give any performance guarantee, it may be a waste of time to compute exact solutions. In some cases one might even observe the weird effect that an exact solution yields results that are inferior to approximate solutions.
5. Finally, approximation algorithms can avoid problems with floating point arithmetic. In Althaus and Mehlhorn (1998) such a problem is analyzed for the maximum flow problem using a preflow-push algorithm. These unexpected difficulties may always occur when using floating point arithmetic in exact algorithms.

Of course, an algorithm with linear runtime may turn out to be completely useless in practice. A famous such example is the algorithm of Bodlaender (1996) for determining the treewidth of a graph. More precisely, if k is fixed, then for a given graph G the algorithm of Bodlaender finds in linear time a tree decomposition of width at most k, or decides that the treewidth of G exceeds k. A huge constant is involved in the linear runtime of Bodlaenders algorithm. Therefore, this algorithm is not feasible in practice, even not for $k = 4$.

All linear time algorithms that we present in this paper have constants in their runtime that are small, which means that they are at least not larger than the constants involved in the runtime of exact algorithms for the problem.

In this survey we will cover linear time deterministic algorithms only. There are related subjects as for example nearly linear time algorithms, sublinear time algorithms, or linear time randomized algorithms which we will not discuss in this paper. In the following the notion linear time algorithm always means a deterministic linear time algorithm.

9.1.1 A Technique for Obtaining Linear Runtime

There exists one general approach for obtaining linear time approximation algorithms for combinatorial optimization problems. Several exact algorithms for these problems work in *phases*, where each phase has linear runtime. Examples are the matching algorithm of Micali and Vazirani (1980) which needs $O(\sqrt{n})$ phases, each phase can be accomplished in $O(m)$, or the algorithm of Ford and Fulkerson (1956) for computing a flow of maximum value, where in each phase one flow augmenting path is computed in $O(m)$.

A simple way to turn such algorithms into linear time approximation algorithms is to simply stop after executing a constant number of phases. We will call this approach the *phase bounding approach*. This approach clearly results in linear time approximation algorithms, however, it is not at all clear what approximation ratio it achieves. One can prove for example (see below) that the phase bounding approach applied to the matching algorithm of Micali and Vazirani (1980) achieves an approximation ratio, that can be arbitrarily close to 1. However, the phase bounding approach applied to Ford and Fulkerson's maximum flow algorithm cannot guarantee any constant approximation ratio larger than 0.

The phase bounding approach while it can produce linear time approximation algorithms with constant approximation ratio misses one of the advantages that we listed above, namely to be much simpler than exact algorithms. Nevertheless, in cases where no other approach is known to yield similar results we do at least know what approximation ratio can be achieved in linear time. Such a result should be considered as a stimulation to look for simpler algorithms that do not use the phase bounding approach.

9.2 Matchings

A *matching* M in a graph $G = (V, E)$ is a subset of the edges of G such that no two edges in M are incident to the same vertex. A *perfect matching* in G is a matching M such that each vertex of G is contained in an edge of M. Computing a (perfect) matching that is optimal with respect to certain side constraints is one of the fundamental problems in combinatorial optimization.

The first polynomial time matching algorithms date back to the papers of Kőnig and Egerváry from 1931 (Frank 2005) which resulted in the famous Hungarian method for bipartite graphs due to Kuhn (1955). A major breakthrough was Edmonds' polynomial time algorithm for the MAXIMUM WEIGHT MATCHING PROBLEM (Edmonds 1965). Edmonds introduced in his paper for the first time the notion of a 'good' algorithm which led to the definition of the class $\mathcal{P}$ of polynomially time solvable problems.

Given a matching M in a graph $G = (V, E)$, we say that a vertex x is *matched* if $\{x, y\} \in M$ for some $y \in V$ and *free* otherwise. An *M-augmenting path* is a simple path $P = v_0, v_1, \ldots, v_k$ such that the endpoints v_0 and v_k are free, $\{v_i, v_{i+1}\} \in E(G)$ and the edges of P are alternately in $E(G) \setminus M$ and M. The *length* of a path

is defined as the number of edges contained in it. As the endpoints of an augmenting path are free this implies that an augmenting path has odd length. Given an augmenting path, one can augment the matching M by deleting from M the edges on the path that are in M, and adding all of the other edges on the path to M. This results in a matching with one more edge. A well known result that is the basis of many matching algorithms says that the absence of an augmenting path implies optimality of the current matching. This result was proved by Petersen (1891) and first formulated in the language of modern graph theory by Berge (1957).

Theorem 2.1 (Petersen 1891; Berge 1957). *A matching M has maximum size if and only if there exists no M-augmenting path.*

9.2.1 Maximum Cardinality Matchings

The maximum cardinality matching problem simply asks for a matching of maximum size in an unweighted undirected graph. As a special case it contains the question whether a given graph has a perfect matching.

CARDINALITY MATCHING PROBLEM	
Input:	An undirected graph G.
Output:	A maximum cardinality matching in G.

The fastest known deterministic algorithm for the CARDINALITY MATCHING PROBLEM is due to Goldberg and Karzanov (2004) and has runtime $O(\sqrt{n}m \cdot \log(n^2/m)/\log n)$ (see also Fremuth-Paeger and Jungnickel 2003). It makes use of graph compression techniques that have been developed by Feder and Motwani (1995). For dense graphs Mucha and Sankowski (2004) presented a faster randomized algorithm which has been simplified by Harvey (2006).

We will show that the phase bounding approach can be used to get a linear time approximation algorithm for the CARDINALITY MATCHING PROBLEM with approximation ratio arbitrarily close to 1. The next lemma is the key to prove such a result. It says that if a matching does not admit short augmenting paths then its cardinality must be close to the cardinality of a maximum cardinality matching. This result is due to Hopcroft and Karp (1973) but has been rediscovered several times, e.g., (Fischer et al. 1993) or (Hassin and Lahav 1994).

Lemma 2.2 (Hopcroft and Karp 1973). *If M is a matching in a graph G such that every M-augmenting path has length at least $2k-1$ then*

$$|M| \geq \frac{k-1}{k} \cdot |M^*|,$$

where M^* denotes a maximum cardinality matching in G.

Proof. Suppose the matching M does not admit augmenting paths of length less than k. Consider the symmetric difference between M and M^*. It contains $|M^*|-|M|$

vertex disjoint M-augmenting paths. Since each of these paths contains at least $k-1$ edges of M, we have

$$|M| \geq \frac{k-1}{k} \cdot |M^*|. \qquad \square$$

For the phase bounding approach one can make use of the maximum cardinality matching algorithm due to Micali and Vazirani (1980), see also (Vazirani 1994; Blum 1990) and (Gabow and Tarjan 1991). Their algorithm constructs in each phase a maximal set of vertex disjoint augmenting paths. Each phase can be implemented in $O(m)$ time. The size of the shortest augmenting path strictly increases from a phase to the next. Thus, by applying Lemma 2.2 one gets the following result which was first observed by Gabow and Tarjan (1988).

Theorem 2.3 (Gabow and Tarjan 1988). *For every fixed $\epsilon > 0$ there exists an algorithm that computes a matching of size at least $(1-\epsilon) \cdot |M^*|$ in linear time.*

The number of phases needed to obtain a maximum cardinality matching is $O(\sqrt{n})$. One can easily construct instances where this number of phases is needed. In practice it turns out that usually a much fewer number of phases suffices to find a maximum cardinality matching in a graph. This observation can be proved for certain graph instances rigorously. A first such result is due to Motwani (1994). It has been improved by Bast et al. (2006) who proved the following statement.

Theorem 2.4 (Bast et al. 2006). *In $G_{n,\frac{33}{n}}$ the algorithm of Micali and Vazirani terminates with high probability after $O(\log n)$ phases.*

Here, $G_{n,\frac{33}{n}}$ denotes a random graph on n vertices where each edge is in the graph with probability $\frac{33}{n}$.

Theorem 2.3 shows that one can find in linear time a matching whose cardinality is arbitrarily close to the cardinality of a maximum cardinality matching. However, as we used the phase bounding approach to get this result, the algorithm is not simpler than the exact one. Therefore we will consider here alternative approaches to compute large matchings in linear time.

One such approach is to simply compute a maximal matching. A matching M is called *maximal* if for all $e \in E(G) \setminus M$ the set $M \cup \{e\}$ is not a matching. For maximal matchings we get the following simple result in the special case $k = 2$ of Lemma 2.2.

Lemma 2.5. *If M is a maximal matching and M^* a maximum cardinality matching then $|M| \geq \frac{1}{2} \cdot |M^*|$.*

Maximal matchings can easily be computed in linear time: start with $M = \emptyset$ and for each $e \in E(G)$ add e to M if both endpoints of e are free. Therefore, Lemma 2.5 immediately implies a very simple linear time approximation algorithm for the CARDINALITY MATCHING PROBLEM with approximation ratio $1/2$. It is easily seen that this approximation ratio is tight (see Fig. 9.1).

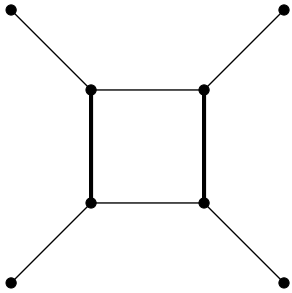

Fig. 9.1. An example where a maximum matching is twice as large as a maximal matching (*bold* edges)

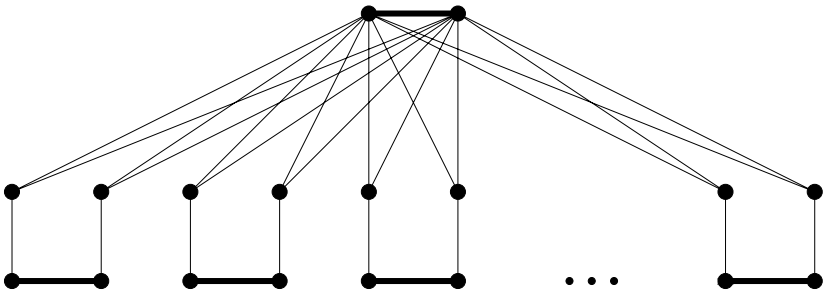

Fig. 9.2. An example where a maximum matching is asymptotically twice as large as a maximal matching (*bold* edges) that considers vertex degrees

Figure 9.1 suggest that it should be a good idea to consider the degrees when computing a maximal matching: First sort in linear time the vertices in increasing order by their degrees and then compute a maximal matching by choosing as the next edge one that is incident to a vertex of currently lowest degree. This approach is often used in practice (Karypis and Kumar 1998) and usually yields better results. But as Fig. 9.2 shows this approach does not improve the approximation ratio.

A better approximation ratio than $1/2$ can be achieved by a simple algorithm that results from an algorithm of Drake Vinkemeier and Hougardy (2005) for the MAXIMUM WEIGHT MATCHING PROBLEM by specializing it to the unweighted case. The slightly complicated computation of 'good' augmentations that is needed in their algorithm becomes completely trivial in the unweighted case. This way one obtains a simple linear time approximation algorithm for the CARDINALITY MATCHING PROBLEM with an approximation ratio arbitrarily close to $2/3$.

However, one can obtain even an approximation ratio of $4/5$ in the unweighted case. For this we will make use of the following result of Hopcroft and Karp (1973).

Lemma 2.6 (Hopcroft and Karp 1973). *Let M be a matching in a graph G such that every M-augmenting path has length at least k. If $\mathcal{P}$ is a maximal set of vertex*

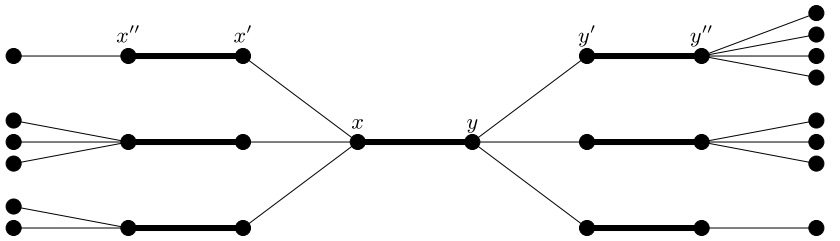

Fig. 9.3. How to find a maximal set of vertex disjoint augmenting paths of length 7 in linear time

disjoint M-augmenting paths of length k then after augmenting all paths in $\mathcal{P}$ the shortest augmenting path in the new matching has length at least $k + 2$.

A maximal set of vertex disjoint M-augmenting paths of length k can be found in linear time for small k.

Lemma 2.7. *Let M be a matching in a graph G. For $k \leq 7$ a maximal set of vertex disjoint M-augmenting paths of length k can be found in linear time.*

Proof. We prove the result here for $k = 7$ only. The proof shows that the statement also holds for smaller k. Let M be a matching in G such that each M-augmenting path has length at least 7. For each edge in M we check whether it is the middle edge of an M-augmenting path of length 7 in G.

Let $e = \{x, y\} \in M$ be such an edge. Then we scan all matched neighbors of y. For each such neighbor, say y', we scan its incident edge $\{y', y''\} \in M$. This can be done in time proportional to the degree of y. Similarly, we scan all possible neighbors x' and incident edges $\{x', x''\} \in M$ of x in time proportional to the degree of x (see Fig. 9.3).

Now we simply have to look for a vertex y'' that has a free neighbor y''' and a vertex x'' that has a free neighbor x'''. This clearly can be done in time proportional to the sum of the degrees of vertex x and y. However, we need to be a bit careful, as the vertices y', y'', y''' and x', x'', x''' need not to be distinct. Therefore we proceed as follows: If there is a vertex y'' that has at least two free neighbors and there is an edge $\{x', x''\}$ different from $\{y', y''\}$ such that x'' has a free neighbor then we have found an M-augmenting path of length 7 in time proportional to the sum of the degree of x and y. If all y'' have only one free neighbor, we check whether there are at least two different such neighbors. If so, we can find an M-augmenting path of length 7 as soon as there is a vertex x'' on the other side. If all vertices y'' are adjacent to the same free vertex we also can easily check whether an M-augmenting path of length seven can be found.

Therefore we can find an M-augmenting path of length 7 with the edge $\{x, y\}$ in the middle in time proportional to the sum of the degrees of x and y. Thus, a maximal vertex disjoint set of such paths can be found in linear time. □

As a consequence of Lemma 2.6 and Lemma 2.7 we get a linear time algorithm with approximation ratio 4/5 for the CARDINALITY MATCHING PROBLEM.

Theorem 2.8. *If M^* is a maximum cardinality matching then a matching M with $|M| \geq \frac{4}{5} \cdot |M^*|$ can be computed in linear time.*

Better approximation algorithms can be obtained in the special case of bounded degree graphs. In this case one easily can check for all augmenting paths up to a fixed length k containing a given vertex x in constant time. More precisely: if all vertex degrees are bounded by a constant B, then there can exist at most B^k paths of length k containing a given vertex. Therefore, one gets a simple linear time approximation algorithm for bounded degree graphs with approximation ratio arbitrarily close to 1. However, as this approach involves the constant B^k in the runtime it is not useful in practice.

9.2.2 Maximum Weight Matchings

Given a graph $G = (V, E)$ and a weight function $c : E(G) \to \mathbb{R}$, the weight of a matching $M \subseteq E$ is defined as $w(M) := \sum_{e \in M} w(e)$. The MAXIMUM WEIGHT MATCHING PROBLEM now asks for a matching of maximum weight. The CARDINALITY MATCHING PROBLEM is a special case of this problem where all weights are equal.

MAXIMUM WEIGHT MATCHING PROBLEM	
Input:	An undirected graph G and edge weights $c : E(G) \to \mathbb{R}$.
Output:	A maximum weight matching in G.

The fastest known algorithm for the MAXIMUM WEIGHT MATCHING PROBLEM is due to Gabow (1990) and has runtime $O(nm + n^2 \log n)$. However, this algorithm involves rather complicated data structures that prevent it from being useful in practice. The fastest implementations known today for solving the MAXIMUM WEIGHT MATCHING PROBLEM are due to Cook and Rohe (1999) respectively Mehlhorn and Schäfer (2002). These algorithms have a worst case runtime of $O(n^3)$ respectively $O(nm \log n)$. Under the assumption that all edge weights are integers in the range $[1..N]$ Gabow and Tarjan (1991) presented an algorithm with runtime $O(\sqrt{n \log n \alpha(m, n)} m \log(Nn))$, where α is the inverse of Ackermann's function.

The greedy algorithm for the MAXIMUM WEIGHT MATCHING PROBLEM is a very simple approximation algorithm that achieves an approximation ratio of $1/2$ (Jenkyns 1976; Korte and Hausmann 1978; Avis 1978). This algorithm simply sorts the edges by decreasing weight and then computes a maximal matching using this ordering. The runtime of the greedy algorithm is $O(m \log n)$ as sorting the edges requires this amount of time. Surprisingly, for long time no linear time approximation algorithm for the MAXIMUM WEIGHT MATCHING PROBLEM was known that achieves a constant approximation ratio strictly larger than zero. The first such algorithm was presented by Preis (1999). The idea of this algorithm is that instead of using the heaviest edge in each step it is enough to consider a locally heaviest edge, i.e., an edge that is heavier than all adjacent edges. Using this approach it is easy to

```
PathGrowingAlgorithm (G = (V, E), w : E → ℝ+)
 1  M1 := ∅, M2 := ∅, i := 1
 2  while E ≠ ∅ do begin
 3      choose x ∈ V of degree at least 1 arbitrarily
 4      while x has a neighbor do begin
 5          let {x, y} be the heaviest edge incident to x
 6          add {x, y} to Mi
 7          i := 3 − i
 8          remove x from G
 9          x := y
10      end
11  end
12  return max(w(M1), w(M2))
```

Fig. 9.4. The Path Growing Algorithm for finding maximum weight matchings

see that the algorithm achieves an approximation ratio of $1/2$. However, it is quite complicated to prove that its runtime is indeed linear.

Drake and Hougardy (2003a) presented a much simpler linear time approximation algorithm for the MAXIMUM WEIGHT MATCHING PROBLEM with approximation ratio $1/2$. This algorithm is called the Path Growing Algorithm and is shown in Fig. 9.4. Its idea is to simultaneously compute two matchings and for the heavier of these two matchings the algorithm guarantees that its weight is at least half the weight of a maximum weight matching.

It is easily seen that the runtime of the Path Growing Algorithm is linear (Drake and Hougardy 2003a). The next result shows the correctness of the algorithm.

Theorem 2.9 (Drake and Hougardy 2003a). *The Path Growing Algorithm has a performance ratio of* $1/2$.

Proof. For the analysis of the performance ratio we will assign each edge of the graph to some vertex of the graph in the following way. Whenever a vertex is removed in line 8 of the algorithm all edges which are currently incident to that vertex x are assigned to x. This way each edge of G is assigned to exactly one vertex of G. Note that there might be vertices in G that have no edges assigned to them.

Now consider a maximum weight matching M in G. As M must not contain two incident edges, all edges of M are assigned to different vertices of G. In each step of the algorithm the heaviest edge that was currently incident to vertex x is chosen in line 5 of the algorithm and added to M_1 or M_2. Therefore the weight of $M_1 \cup M_2$ is at least as large as the weight of M. As

$$\max(w(M_1), w(M_2)) \;\geq\; \frac{1}{2}\, w(M_1 \cup M_2) \;\geq\; \frac{1}{2}\, w(M)$$

the weight returned by the Path Growing Algorithm is at least half the weight of the optimal solution. □

It turns out that the greedy algorithm, the algorithm of Preis, and the Path Growing Algorithm of Drake and Hougardy are in practice much better than the approximation factor of $1/2$ suggests. In Drake and Hougardy (2003a) it is shown that the solution found by these algorithms is typically about 5% away from the weight of an optimum solution. For the Path Growing Algorithm one even can get a guarantee that the solution found by the algorithm is strictly larger than the approximation ratio of $1/2$. This is due to the fact that this algorithm computes two different matchings for which it guarantees that the sum of the weights of these two matchings is at least as large as the weight of an optimum solution. Therefore we have the following observation which allows to get a better guarantee how far the solution returned by the Path Growing Algorithm is away from an optimum solution.

Lemma 2.10. *The matching M returned by the Path Growing Algorithm has weight at least*

$$\frac{\max\{w(M_1), w(M_2)\}}{w(M_1) + w(M_2)} \cdot w(M^*),$$

where M^* is a maximum weight matching.

Note that we have $\frac{1}{2} \leq \frac{\max\{w(M_1),w(M_2)\}}{w(M_1)+w(M_2)} \leq 1$. This means that the Path Growing Algorithm might even return a matching M and a guarantee that this matching M is a maximum weight matching.

Drake Vinkemeier and Hougardy (2005) improved on the Path Growing Algorithm by presenting a linear time approximation algorithm for the MAXIMUM WEIGHT MATCHING PROBLEM that achieves an approximation ratio arbitrarily close to $2/3$. This approximation ratio is the best that is currently known to be achievable in linear time.

Theorem 2.11 (Drake Vinkemeier and Hougardy 2005). *For every fixed $\epsilon > 0$ there exists a linear time algorithm that computes a matching of weight at least $(2/3 - \epsilon) \cdot w(M^*)$ where M^* is a maximum weight matching.*

Pettie and Sanders (2004) improved on this result by presenting another linear time $2/3 - \epsilon$ approximation algorithm for the MAXIMUM WEIGHT MATCHING PROBLEM whose runtime has a better dependence on ϵ.

9.2.3 Minimum Weight Perfect Matchings

Computing a perfect matching of maximum or minimum weight is a problem that appears quite often in applications. The maximum and minimum perfect matching problem can easily be transformed into each other by just negating the edge weights.

MINIMUM WEIGHT PERFECT MATCHING PROBLEM	
Input:	An undirected graph G and edge weights $c : E(G) \to \mathbb{R}$.
Output:	A minimum weight perfect matching in G or a proof that G has no perfect matching.

There is a simple reduction that allows to formulate the MAXIMUM WEIGHT MATCHING PROBLEM as a MAXIMUM WEIGHT PERFECT MATCHING PROBLEM. Take a copy G' of the input graph G and connect each vertex in G by an edge of weight 0 with its copy in G'. Then a maximum weight perfect matching in the new graph corresponds to a maximum weight matching in G (the weights differ exactly by a factor of 2).

This is a simple reduction that can be performed in linear time, unfortunately it does not preserve approximation ratios. However, we cannot expect to find any such reduction as every algorithm that finds an approximate solution to the MINIMUM WEIGHT PERFECT MATCHING PROBLEM must at least be able to decide whether the graph has a perfect matching. The fastest algorithm for doing so has runtime $O(\sqrt{n}m \log(n^2/m)/\log n)$ (Goldberg and Karzanov 2004). Therefore linear time approximation algorithms with constant approximation ratios do not exist for the MINIMUM WEIGHT PERFECT MATCHING PROBLEM unless one can decide in linear time whether a given graph has a perfect matching.

9.3 Degree Constrained Subgraphs

Let $G = (V, E)$ be an undirected graph and $b : V(G) \to \mathbb{N}$ a degree constraint for every vertex. A subgraph H of G is a *degree constrained subgraph* for G with respect to b, if the degree of each vertex x in H is at most $b(x)$. We consider here the problems of finding a degree constrained subgraph with the largest possible number of edges and a weighted version of this problem. A degree constrained subgraph is also known as *b-matching*. A subset $M \subset E$ is called a *b-matching* if for all $v \in V(G)$ the number of edges in M incident to v is at most $b(v)$. For the special case that $b(v) = 1$ for all $v \in V(G)$ the b-matching is a matching. Therefore, the degree constrained subgraph problems are generalizations of matching problems. As the connections to matching problems are quite strong, we prefer here the notion of b-matchings instead of degree constrained subgraphs.

CARDINALITY b-MATCHING PROBLEM	
Input:	An undirected graph G and $b : V(G) \to \mathbb{N}$.
Output:	A maximum cardinality b-matching in G.

The CARDINALITY b-MATCHING PROBLEM can be reduced to the CARDINALITY MATCHING PROBLEM by the following reduction which is due to Shiloach (1981): Replace each vertex v of degree $d(v)$ by $d(v)$ copies, such that each edge incident to v uses another copy of v. Then add additional $b(v)$ vertices for each vertex v that are completely connected to the $d(v)$ copies of v. Figure 9.5 shows an example for this reduction.

Let G be an arbitrary graph with n vertices and m edges and let G' be the graph that results from this reduction. Then it is not difficult to prove that a matching in G' of size $\alpha + m$ corresponds to a b-matching in G of size α (Shiloach 1981). The graph

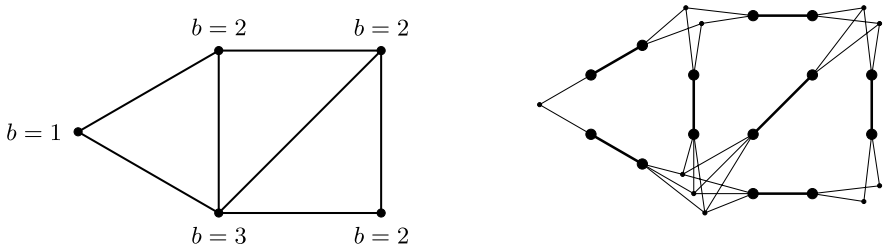

Fig. 9.5. An example illustrating the reduction of a b-matching problem to a matching problem

G' has $O(m)$ vertices and $O(Bm)$ edges, where $B = \max_{v \in V(G)}\{b(v)\}$. For constant B this reduction leads to a b-matching algorithm with runtime $O(m^{3/2})$ by using the algorithm of Micali and Vazirani (1980). For non-constant B the fastest known algorithm for the CARDINALITY b-MATCHING PROBLEM has runtime $O(nm \log n)$ and is due to Gabow (1983).

Unfortunately the above reduction does not preserve approximation ratios. Therefore, even for constant B we cannot use the constant factor approximation algorithms for the CARDINALITY MATCHING PROBLEM to obtain such algorithms for the CARDINALITY b-MATCHING PROBLEM. Instead, we directly have to adapt the approximation algorithms to the CARDINALITY b-MATCHING PROBLEM. A b-matching M in a graph $G = (V, E)$ is called *maximal* if $M \cup \{e\}$ is not a b-matching for all $e \in E \setminus M$. A maximal b-matching can easily be computed in linear time: Start with $M = \emptyset$ and for each $e \in E(G)$ add e to M if this does not violate the degree conditions. The following result shows that this already gives a linear time $1/2$-approximation algorithm for the CARDINALITY b-MATCHING PROBLEM.

Theorem 3.1. *The* CARDINALITY b-MATCHING PROBLEM *can be solved in linear time with approximation ratio* $\frac{1}{2}$.

Proof. As observed above a maximal b-matching can be computed in linear time. Thus we simply have to prove that if M^* is a maximum cardinality b-matching and M is a maximal b-matching then $|M| \geq \frac{1}{2} \cdot |M^*|$. By S we denote all vertices v such that there are exactly $b(v)$ edges of M incident to v. Then every edge in M^* must have at least one endpoint in S as otherwise M is not maximal. As for each vertex $v \in S$ there are at most $b(v)$ edges of M^* incident to v and each edge in M is incident to at most two vertices in S we have:

$$|M^*| \;\leq\; \sum_{v \in S} b(v) \;\leq\; 2 \cdot |M|$$

which proves the approximation ratio of $1/2$. □

We now also consider a weighted version of the b-matching problem.

MAXIMUM WEIGHT b-MATCHING PROBLEM	
Input:	An undirected graph G and edge weights $c : E(G) \to \mathbb{R}$.
Output:	A maximum weight b-matching in G.

For the weighted version of the b-matching problem one can use the above mentioned reduction of Shiloach (1981) to get a MAXIMUM WEIGHT MATCHING PROBLEM. Again, faster algorithms are possible by adapting algorithms for the MAXIMUM WEIGHT MATCHING PROBLEM directly to the MAXIMUM WEIGHT b-MATCHING PROBLEM. This has been done by Gabow (1983) who obtained the currently fastest algorithm for the MAXIMUM WEIGHT b-MATCHING PROBLEM. His algorithm has runtime $O(m^2 \log n)$. It is easily seen that the Greedy algorithm for the MAXIMUM WEIGHT b-MATCHING PROBLEM achieves an approximation ratio of $1/2$ and has runtime $O(m \log n)$ (Avis 1978).

Linear time approximation algorithms for the MAXIMUM WEIGHT b-MATCHING PROBLEM are known only for the case that $b(x)$ is bounded by some constant for all $x \in V(G)$. In this case Mestre (2006) obtained the following result by adapting an algorithm of Drake and Hougardy (2003b) for the MAXIMUM WEIGHT MATCHING PROBLEM to the MAXIMUM WEIGHT b-MATCHING PROBLEM.

Theorem 3.2 (Mestre 2006). *If $b(x)$ is bounded by some constant for all $x \in V(G)$ then the* MAXIMUM WEIGHT b-MATCHING PROBLEM *can be solved in linear time with approximation ratio $\frac{1}{2}$.*

9.4 Branchings

A *branching* is a directed graph without circuits such that each vertex has indegree at most one. Computing branchings of maximum weight is one of the classical problems in combinatorial optimization.

MAXIMUM WEIGHT BRANCHING PROBLEM	
Input:	A directed graph G and edge weights $c : E(G) \to \mathbb{R}$.
Output:	A maximum weight branching in G.

Edmonds (1967) and independently Chu and Liu (1965) and Bock (1971) gave the first polynomial time algorithms for computing a maximum weight branching. Later Gabow et al. (1986) showed that Edmond's algorithm can be implemented in $O(m + n \log n)$ time. This is already quite close to a linear time algorithm but from a practical point of view one is still interested in simpler linear time approximation algorithms for the MAXIMUM WEIGHT BRANCHING PROBLEM.

One such algorithm is given by the following greedy approach: Start with an empty branching B and sort the edges by decreasing weight. Now check for each edge one by one whether its addition violates the degree condition. If not then add it to B. Now one can destroy each circuit that might appear by removing the lightest edge from each circuit. This can be done in linear time. As each circuit has at least two edges, the algorithm has an approximation ratio of $1/2$. The total runtime is linear, as sorting of the edges is actually not required: simply take the heaviest incoming edge for each vertex.

This result can be improved by applying the phase bounding approach to Edmonds' algorithm. This has been observed by Ziegler (2008). He obtained the following result.

Theorem 4.1 (Ziegler 2008). *For every* $\epsilon > 0$ *there exists a linear time approximation algorithm for the* MAXIMUM WEIGHT BRANCHING PROBLEM *that has an approximation ratio of* $1 - \epsilon$.

Acknowledgement

I am grateful to an anonymous referee for several useful comments.

References

Althaus, E., Mehlhorn, K.: Maximum network flow with floating point arithmetic. Inf. Process. Lett. **66**(3), 109–113 (1998)

Avis, D.: Two greedy heuristics for the weighted matching problem. Congr. Numer. **XXI**, 65–76 (1978). Proceedings of the Ninth Southeastern Conference on Combinatorics, Graph Theory, and Computing (1978)

Bast, H., Mehlhorn, K., Schäfer, G., Tamaki, H.: Matching algorithms are fast in sparse random graphs. Theory Comput. Syst. **39**(1), 3–14 (2006)

Berge, C.: Two theorems in graph theory. Proc. Natl. Acad. Sci. U.S.A. **43**(9), 842–844 (1957)

Blum, N.: A new approach to maximum matching in general graphs. In: Proc. of 17th ICALP (1990). Lecture Notes in Computer Science, vol. 443, pp. 586–597. Springer, Berlin (1990)

Bock, F.: An algorithm to construct a minimum directed spanning tree in a directed network. In: Avi Itzhak, B. (ed.) Developments in Operations Research, Proceedings of the Third Annual Israel Conference on Operations Research, July 1969, vol. 1, pp. 29–44. Gordon and Breach, New York (1971). Paper 1-2

Bodlaender, H.L.: A linear-time algorithm for finding tree-decompositions of small treewidth. SIAM J. Comput. **25**(6), 1305–1317 (1996)

Chu, Y.-J., Liu, T.-H.: On the shortest arborescence of a directed graph. Sci. Sin. **14**(10), 1396–1400 (1965)

Cook, W., Rohe, A.: Computing minimum-weight perfect matchings. INFORMS J. Comput. **11**(2), 138–148 (1999)

Drake, D.E., Hougardy, S.: Linear time local improvements for weighted matchings in graphs. In: Jansen, K. et al. (eds.) International Workshop on Experimental and Efficient Algorithms (WEA) 2003. Lecture Notes in Computer Science, vol. 2647, pp. 107–119. Springer, Berlin (2003a)

Drake, D.E., Hougardy, S.: A simple approximation algorithm for the weighted matching problem. Inf. Process. Lett. **85**(4), 211–213 (2003b)

Drake Vinkemeier, D.E., Hougardy, S.: A linear-time approximation algorithm for weighted matchings in graphs. ACM Trans. Algorithms **1**(1), 107–122 (2005)

Edmonds, J.: Paths, trees, and flowers. Can. J. Math. **17**(3), 449–467 (1965)

Edmonds, J.: Optimum branchings. J. Res. Natl. Bur. Stand., B Math. Math. Phys. **71**(4), 233–240 (1967)

Feder, T., Motwani, R.: Clique partitions, graph compression and speeding-up algorithms. J. Comput. Syst. Sci. **51**(2), 261–272 (1995)

Fischer, T., Goldberg, A.V., Haglin, D.J., Plotkin, S.: Approximating matchings in parallel. Inf. Process. Lett. **46**(3), 115–118 (1993)

Ford, L.R., Fulkerson, D.R.: Maximal flow through a network. Can. J. Math. **8**, 399–404 (1956)

Frank, A.: On Kuhn's Hungarian method—a tribute from Hungary. Nav. Res. Logist. **52**(1), 2–5 (2005)

Fremuth-Paeger, C., Jungnickel, D.: Balanced network flows. VIII. A revised theory of phase-ordered algorithms and the $O(\sqrt{n}m\log(n^2/m)/\log n)$ bound for the nonbipartite cardinality matching problem. Networks **41**(3), 137–142 (2003)

Gabow, H.N.: An efficient reduction technique for degree-constrained subgraph and bidirected network flow problems. In: STOC '83: Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing, pp. 448–456. ACM Press, New York (1983)

Gabow, H.N.: Data structures for weighted matching and nearest common ancestors with linking. In: SODA '90: Proceedings of the First Annual ACM–SIAM Symposium on Discrete Algorithms, pp. 434–443. SIAM, Philadelphia (1990)

Gabow, H.N., Tarjan, R.E.: Algorithms for two bottleneck optimization problems. J. Algorithms **9**(3), 411–417 (1988)

Gabow, H.N., Tarjan, R.E.: Faster scaling algorithms for general graph-matching problems. J. Assoc. Comput. Mach. **38**(4), 815–853 (1991)

Gabow, H.N., Galil, Z., Spencer, T., Tarjan, R.E.: Efficient algorithms for finding minimum spanning trees in undirected and directed graphs. Combinatorica **6**(2), 109–122 (1986)

Goldberg, A.V., Karzanov, A.V.: Maximum skew-symmetric flows and matchings. Math. Program. **100**(3), 537–568 (2004)

Harvey, N.J.A.: Algebraic structures and algorithms for matching and matroid problems. In: Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06), pp. 531–542. IEEE Computer Society, Washington (2006)

Hassin, R., Lahav (Haddad), S.: Maximizing the number of unused colors in the vertex coloring problem. Inf. Process. Lett. **52**(2), 87–90 (1994)

Hopcroft, J.E., Karp, R.M.: An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. SIAM J. Comput. **2**(4), 225–231 (1973)

Jenkyns, T.A.: The efficacy of the greedy algorithm. Congr. Numer. **17**, 341–350 (1976)

Karypis, G., Kumar, V.: Multilevel k-way partitioning scheme for irregular graphs. J. Parallel Distrib. Comput. **48**(1), 96–129 (1998)

Korte, B., Hausmann, D.: An analysis of the greedy heuristic for independence systems. Ann. Discrete Math. **2**, 65–74 (1978)

Kuhn, H.W.: The Hungarian method for the assignment problem. Nav. Res. Logist. Q. **2**, 83–97 (1955)

Mehlhorn, K., Schäfer, G.: Implementation of $O(nm\log n)$ weighted matchings in general graphs: the power of data structures. ACM J. Exp. Algorithmics **7**, 4 (2002)

Mestre, J.: Greedy in approximation algorithms. In: Azar, Y., Erlebach, T. (eds.) ESA 2006. Lecture Notes in Computer Science, vol. 4168, pp. 528–539. Springer, Berlin (2006)

Micali, S., Vazirani, V.V.: An $O(\sqrt{|v|}\cdot|E|)$ algorithm for finding maximum matching in general graphs. In: Proc. of 21st Annual Symposium on Foundations of Computer Science (21st FOCS, Syracuse, New York, 1980), pp. 17–27 (1980)

Motwani, R.: Average-case analysis of algorithms for matchings and related problems. J. Assoc. Comput. Mach. **41**(6), 1329–1356 (1994)

Mucha, M., Sankowski, P.: Maximum matchings via Gaussian elimination. In: Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science (FOCS'04), pp. 248–255. IEEE Computer Society, Washington (2004)

Petersen, J.: Die Theorie der regulären Graphs. Acta Math. **15**(1), 193–220 (1891)

Pettie, S., Sanders, P.: A simpler linear time $2/3 - \varepsilon$ approximation for maximum weight matching. Inf. Process. Lett. **91**(6), 271–276 (2004)

Preis, R.: Linear time $\frac{1}{2}$-approximation algorithm for maximum weighted matching in general graphs. In: Meinel, C., Tison, S. (eds.) Symposium on Theoretical Aspects in Computer Science (STACS). Lecture Notes in Computer Science, vol. 1563, pp. 259–269. Springer, Berlin (1999)

Shiloach, Y.: Another look at the degree constrained subgraph problem. Inf. Process. Lett. **12**(2), 89–92 (1981)

Vazirani, V.V.: A theory of alternating paths and blossoms for proving correctness of the $O(\sqrt{V}E)$ general graph maximum matching algorithm. Combinatorica **14**(1), 71–109 (1994)

Ziegler, V.: Approximating optimum branchings in linear time. Technical report, Humboldt-Universität zu Berlin, Institut für Informatik (2008)