This paper appeared in: Information Processing Letters 85 (2003), 211-213

A Simple Approximation Algorithm for the Weighted Matching Problem

Doratha E. Drake Stefan Hougardy

Humboldt-Universität zu Berlin
Institut für Informatik
10099 Berlin, GERMANY
{drake,hougardy}@informatik.hu-berlin.de

revised Version
July 15th, 2002

Abstract. We present a linear time approximation algorithm with a performance ratio of 1/2 for finding a maximum weight matching in an arbitrary graph. Such a result is already known and is due to Preis [7]. Our algorithm uses a new approach which is much simpler than the one given in [7] and needs no amortized analysis for its running time.

Keywords. approximation algorithms, analysis of algorithms, graph algorithms, maximum weight matching

1 Introduction

A *matching* M in a graph $G = (V, E)$ is a subset of the edges E of G such that no two edges in M are incident. Finding a matching of maximum size in a graph is a fundamental problem in algorithmic graph theory which has many applications. The fastest algorithm known today for solving this problem is due to Micali and Vazirani [4][8] and has a running time of $O(\sqrt{|V|}|E|)$.

In this paper we are interested in algorithms for the weighted matching problem. The problem is defined as follows: Let $G = (V, E)$ be a graph and $w : E \to \mathbb{R}_+$ be a function which assigns a weight to each of the edges of G. Then the weight $w(F)$ of a subset $F \subseteq E$ of the edges of G is defined as $w(F) := \sum_{e \in F} w(e)$. Now the weighted matching problem is to find a matching M in G that has maximum weight.

The fastest algorithm known today for solving the weighted matching problem in general graphs is due to Gabow [2] and has a running time of $O(|V||E| + |V|^2 \log |V|)$. Under the assumption that all edge weights are integers in the range $[1 \dots N]$ Gabow and Tarjan [3] presented an algorithm with running time $O(\sqrt{|V| \log(|V|)\alpha(|E|, |V|)}|E| \log(N|V|))$, where α is the inverse of Ackermann's

This paper appeared in: Information Processing Letters 85 (2003), 211-213

```
Greedy Matching (G = (V, E), w : E → ℝ+)
1  M := ∅
2  while E ≠ ∅ do begin
3     let e be the heaviest edge in E
4     add e to M
5     remove e and all edges incident to e from E
6  end
```

Figure 1: The greedy algorithm for finding maximum weight matchings.

function. For many practical problems involving very large graphs such running times can be too costly. Examples are the refinement of FEM nets [5] and the partitioning problem in VLSI-Design [6]. Therefore one is interested in approximation algorithms for the weighted matching problem which ideally have linear running time.

One other reason why one is interested in *simple* approximation algorithms, is that implementing polynomial time exact algorithms for the weighted matching problem is a very tedious task [1].

The quality of an approximation algorithm for solving the weighted matching problem is measured by its so called *performance ratio*. An approximation algorithm has a performance ratio of c, if for all graphs it finds a matching with a weight of at least c times the weight of an optimal solution.

It is easily seen that the greedy algorithm shown in Figure 1 is an approximation algorithm for the weighted matching problem with a performance ratio of $1/2$. The greedy algorithm can be implemented with a running time as fast as $O(|E| \log |V|)$ if the edges of G are sorted in a preprocessing step by decreasing weight.

In [7] Preis improved upon the simple greedy approach by making use of the concept of a so called *locally heaviest edge*. This way he obtained an approximation algorithm for the weighted matching problem with a performance ratio of $1/2$ and a running time of $O(|E|)$. In this paper we use a new approach to get another such algorithm that has two advantages compared to the algorithm in [7]: Our algorithm is much simpler and needs no amortized analysis to prove its linear running time.

2 The Path Growing Algorithm

Our new algorithm with a performance ratio of $1/2$ for finding a maximum weight matching in a graph is shown in Figure 2. Because the main idea involves growing a set of vertex disjoint paths in the given graph, we call it the *Path Growing Algorithm*. The Path Growing Algorithm starts with a path of length 0, i.e. a single vertex, and tries to extend the current path in one direction for as long as possible. The path is always extended along the heaviest edge currently available. Once the heaviest edge has been chosen all other edges incident to the current vertex are deleted to ensure that the paths are vertex disjoint. If

This paper appeared in: Information Processing Letters 85 (2003), 211-213

```
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))
```

Figure 2: The Path Growing Algorithm for finding maximum weight matchings.

it is no longer possible to extend the current path by an edge, a new path is started in a new vertex. This process goes on until all vertices of the graph belong to some path (possibly of length 0).

The edges of all these paths are collected in the two sets M_1 and M_2. Due to line 7 of the algorithm the edges of the paths are alternately inserted into M_1 and M_2. Therefore both of these sets contain no incident edges and are matchings. Of these two matchings the one having larger weight is returned.

3 The running time is linear

Theorem 1 *The Path Growing Algorithm has running time* $O(|E|)$.

Proof. Every vertex of the graph is processed at most once in the while-loop in lines 4–10 because after being processed, it will be removed from G in line 8. Finding the heaviest edge incident to some vertex x and removing x from G can easily be done in time proportional to the degree of x. By always starting paths from among end vertices of an edge one can ensure that no vertex of degree zero will ever be processed. The number of vertices of degree at least 1 in G is $O(|E|)$. Therefore the total running time of the Path Growing Algorithm is bounded by the sum of the degrees of the vertices in V which is $O(|E|)$. □

4 The performance ratio is $1/2$

Theorem 2 *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

This paper appeared in: Information Processing Letters 85 (2003), 211-213

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. □

5 Concluding remarks

It is easy to construct examples of graphs showing that 1/2 is the worst case performance ratio of the Path Growing Algorithm. However, there are several ways to improve on the performance ratio of this algorithm in practice.

First note that a matching returned by the Path Growing Algorithm is not necessarily a maximal matching. Therefore, in a postprocessing phase one can extend the matching found by the Path Growing Algorithm to a maximal matching in $\mathcal{O}(|E|)$.

Secondly, instead of returning one of the matchings M_1 or M_2 as the solution one can return a maximum weight matching for each path computed by the Path Growing Algorithm. Such a maximum weight matching can be computed in linear time for trees and therefore also for paths by a dynamic programming approach.

References

[1] W. Cook, A. Rohe, *Computing minimum-weight perfect matchings,* INFORMS Journal on Computing 11 (1999), pp. 138–148

[2] H.N. Gabow, *Data Structures for Weighted Matching and Nearest Common Ancestors with Linking,* SODA 1990: 434-443

[3] H.N. Gabow, R.E. Tarjan, *Faster Scaling Algorithms for General Graph-Matching Problems,* JACM 38(4): 815-853 (1991)

[4] S. Micali and V.V. Vazirani, *An $O(\sqrt{V}E)$ Algorithm for Finding Maximum Matching in General Graphs,* Proc. 21st Annual IEEE Symposium on Foundations of Computer Science (1980) 17-27.

[5] R.H. Möhring, M. Müller-Hannemann, *Complexity and Modeling Aspects of Mesh Refinement into Quadrilaterals,* Algorithmica 26 (2000), pp. 148–171.

[6] B. Monien , R. Preis , R. Diekmann, *Quality Matching and Local Improvement for Multilevel Graph-Partitioning,* Parallel Computing, 26(12), 2000, 1609-1634.

This paper appeared in: Information Processing Letters 85 (2003), 211-213

[7] R. PREIS, *Linear Time 1/2-Approximation Algorithm for Maximum Weighted Matching in General Graphs,* Symposium on Theoretical Aspects of Computer Science, STACS 99, C. Meinel, S. Tison (eds.), Springer, LNCS 1563, 1999, 259-269

[8] V.V. VAZIRANI, *A Theory of Alternating Paths and Blossoms for Proving Correctness of the $O(\sqrt{V}E)$ Maximum Matching Algorithm,* Combinatorica 14:1 (1994) 71-109.