Linear Time Local Improvements for Weighted Matchings in Graphs*

Doratha E. Drake and Stefan Hougardy

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

Abstract. Recently two different linear time approximation algorithms for the weighted matching problem in graphs have been suggested [5][17]. Both these algorithms have a performance ratio of 1/2. In this paper we present a set of local improvement operations and prove that it guarantees a performance ratio of 2/3. We show that a maximal set of these local improvements can be found in linear time.
To see how these local improvements behave in practice we conduct an experimental comparison of four different approximation algorithms for calculating maximum weight matchings in weighted graphs. One of these algorithms is the commonly used Greedy algorithm which achieves a performance ratio of 1/2 but has $O(m \log n)$ runtime. The other three algorithms all have linear runtime. Two of them are the above mentioned 1/2 approximation algorithms. The third algorithm may have an arbitrarily bad performance ratio but in practice produces reasonably good results. We compare the quality of the algorithms on a test set of weighted graphs and study the improvement achieved by our local improvement operations. We also do a comparison of the runtimes of all algorithms.

1 Introduction

A *matching* M in a graph $G = (V, E)$ is defined to be any subset of the edges of G such that no two edges in M are adjacent. If $G = (V, E)$ is a weighted graph with edge weights given by a function $w : E \to \mathbb{R}_+$ the *weight of a matching* is defined as to be $w(M) := \sum_{e \in M} w(e)$. The weighted matching problem is to find a matching M in G that has maximum weight. Calculating a matching of maximum weight is an important problem with many applications. The fastest known algorithm to date for solving the weighted matching problem in general graphs is due to Gabow [7] and has a runtime of $O(|V||E| + |V|^2 \log |V|)$.

Many real world problems require graphs of such large size that the runtime of Gabow's algorithm is too costly. Examples of such problems are the refinement of FEM nets [15], the partitioning problem in VLSI-Design [16], and the gossiping problem in telecommunications [2]. There also exist applications were the weighted matching problem has to be solved extremely often on only moderately large graphs. An example of such an application is the virtual screening of

* supported by DFG research grant 296/6-3

protein databases containing the three dimensional structure of the proteins [6]. The graphs appearing in such applications only have about 10,000 edges. But the weighted matching problem has to be solved more than 100,000,000 times for a complete database scan.

Therefore, there is considerable interest in approximation algorithms for the weighted matching problem that are very fast, having ideally linear runtime, and that nevertheless produce very good results even if these results are not optimal.

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. Recently two different linear time approximation algorithms for the weighted matching problem in graphs have been suggested [5][17]. Both these algorithms have a performance ratio of $1/2$. In this paper we present a set of local improvement operations and prove that it guarantees a performance ratio of $2/3$. We show that a maximal set of these local improvements can be found in linear time.

The performance ratio only gives information about the worst case behaviour of an algorithm. In this paper we will also study how several algorithms for the weighted matching problem behave in practice. We make an experimental comparison of the three known approximation algorithms for the weighted matching problem that have a performance ratio of $\frac{1}{2}$. We also include a simple, extremely fast heuristic that cannot guarantee any performance ratio but behaves reasonably well in practice. In addition we apply our local improvement operations to all these algorithms and test how much improvement they yield in practice.

2 Local Improvements

The idea of local improvements has been used in several cases to improve the performance ratio of approximation algorithms. See [10, 3] for such examples.

In the case of the unweighted matching problem which is usually called the maximum matching problem it is well known that by local improvements a given matching can be enlarged. In this case the local improvements are augmenting paths, i.e. paths that alternately consist of edges contained in a matching M and not contained in M such that the first and the last vertex of the path are not contained in an edge of M. From a result of Hopcroft and Karp [11] it follows that if M is a matching such that a shortest augmenting path has length at least l then M is an $\frac{l-1}{l+1}$ approximation of a maximum matching.

We extend the notion of augmenting paths to weighted matchings in a natural way. Let $G = (V, E)$ be a weighted graph with weight function $w : E \to \mathbb{R}_+$ and $M \subseteq E$ be an arbitrary matching in G. A path or cycle is called *M-alternating* if it uses alternately edges from M and $E \setminus M$. Note that alternating cycles must contain an even number of edges. Let P be an alternating path such that if it ends in an edge not belonging to M then the endpoint of P is not covered by an edge of M. The path P is called *M-weight-augmenting* if

$$w\left(E(P) \cap M\right) \;<\; w\left(E(P) \setminus M\right) \quad .$$

If P is an M-weight-augmenting path then $M \triangle P$ (the symmetric difference between M and P) is again a matching with strictly larger weight than M. The notion of M-weight-augmenting cycles is defined similarly.

We will consider in the following M-weight-augmenting cycles of length 4, M-weight-augmenting paths of length at most 4 and M-weight-augmenting paths of length 5 that start and end with an edge of M. See Fig. 1 for all possibilities of such weight augmenting paths and cycles.

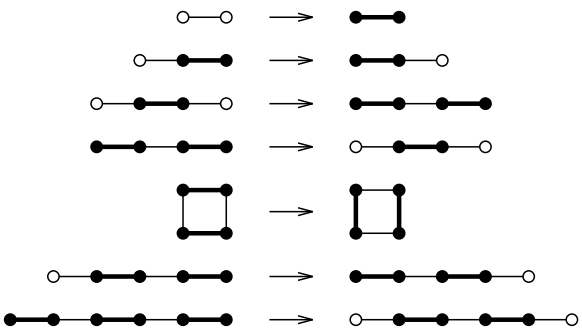

Fig. 1. The seven local improvements. Edges belonging to the matching are shown in bold. Hollow vertices are vertices not contained in any matching edge.

The following result shows that the non-existence of short M-weight-augmenting paths or cycles guarantees that the weight of M is at least $2/3$ of the maximum possible weight.

Theorem 1. *Let M_{opt} be a maximum weight matching in a weighted graph $G = (V, E)$ with weight function $w : E \to \mathbb{R}_+$. If M is a matching in G such that none of the seven operations shown in Fig. 1 increases the weight of M then*

$$w(M) \;\geq\; \frac{2}{3} \cdot w(M_{opt}) \;\; .$$

Proof. Consider the graph induced by the symmetric difference $M \triangle M_{opt}$. It consists of even alternating cycles C_i and alternating paths P_i. We will show that

$$w(C_i \cap M) \;\geq\; \frac{2}{3} \cdot w(C_i \cap M_{opt}) \qquad \forall i \tag{1}$$

and

$$w(P_i \cap M) \;\geq\; \frac{2}{3} \cdot w(P_i \cap M_{opt}) \qquad \forall i \;\; . \tag{2}$$

Equation (1) and (2) imply

$$\begin{aligned} w(M) &= w(M \cap M_{opt}) + \sum w(C_i \cap M) + \sum w(P_i \cap M) \\ &\geq \frac{2}{3} \cdot w(M \cap M_{opt}) + \frac{2}{3} \cdot \sum w(C_i \cap M_{opt}) + \frac{2}{3} \cdot \sum w(P_i \cap M_{opt}) \\ &= \frac{2}{3} \cdot w(M_{opt}) \;\; . \end{aligned}$$

We start proving (1). If C_i is a cycle of length 4 then by the assumptions of the theorem

$$w(C_i \cap M) \;\geq\; w(C_i \cap M_{opt}) \;\geq\; \frac{2}{3} \cdot w(C_i \cap M_{opt}) \;.$$

Now assume C_i is a cycle of length at least 6. Let $e_1, e_2, e_3, \ldots$ be the edges of C_i in a consecutive order such that $e_j \in M$ for j odd. Consider a subpath of type $e_{2k+1}, e_{2k+2}, e_{2k+3}, e_{2k+4}, e_{2k+5}$ in C_i. By the assumptions of the theorem we have

$$w(e_{2k+1}) + w(e_{2k+3}) + w(e_{2k+5}) \;\geq\; w(e_{2k+2}) + w(e_{2k+4}) \;.$$

By summing this inequality over all possible values for k we see that each edge of $C_i \cap M$ appears 3 times on the left side and each edge of $C_i \cap M_{opt}$ appears 2 times on the right side of the inequality. Therefore

$$3 \cdot w(C_i \cap M) \;\geq\; 2 \cdot w(C_i \cap M_{opt}) \;.$$

This proves (1).

We use a similar idea to prove (2). Let e_j be the edges of a path P_i such that $e_j \in M$ if and only if j is odd. The path may start with e_1 or e_0 as we do not know whether it starts with an edge of M or an edge of $E \setminus M$. Consider subpaths of P_i of type $e_{2k+1}, e_{2k+2}, e_{2k+3}, e_{2k+4}, e_{2k+5}$. We have

$$w(e_{2k+1}) + w(e_{2k+3}) + w(e_{2k+5}) \;\geq\; w(e_{2k+2}) + w(e_{2k+4}) \;.$$

Now extend the path P_i artificially to both sides by four edges of weight 0. So we have edges $e_{-1}, e_{-2}, \ldots$ on the left side of P_i. Now add the above inequality for all k where k starts at -2. Then each edge of $P_i \cap M$ appears in three inequalities on the left and each edge of $P_i \cap M_{opt}$ appears in 2 inequalities on the right. The artificial edges may appear in arbitrary number. As they have weight 0, it does not matter. Therefore

$$3 \cdot w(P_i \cap M) \;\geq\; 2 \cdot w(P_i \cap M_{opt}) \;.$$

This proves (2). □

Theorem 2. . *Let $G = (V, E)$ be a weighted graph with weight function $w : E \to \mathbb{R}_+$ and let $M \subseteq E$ be a matching. A maximal set of any of the seven operations shown in Fig. 1 that are pairwise node disjoint and such that each operation increases the weight of M can be found in linear time.*

Proof. To achieve linear runtime in a preprocessing step each vertex that is covered by M gets a pointer to the edge of M it belongs to. Consider an arbitary edge $e \in M$. To decide whether e belongs to an M-weight-augmenting C_4 run over all edges incident to one endpoint of e and mark all edges of M that are adjacent to such edges. Now run over the edges incident to the other endpoint of e and see whether they are incident to the other endpoint of a marked edge. If yes, an alternating C_4 has been found. Check whether it is M-weight-augmenting. If yes remove it from the graph.

Similarly paths of length at most 4 and paths of length 5 which have an edge of M at both ends can be found in linear time. □

3 The Algorithms

In this section we briefly describe four different approximation algorithms for the weighted matching problem in graphs. For all these algorithms we have tested their performance and runtime with and without our local improvements. The results of these tests are given in Section 5.

The first of the approximation algorithms is Greedy Matching [1] shown in Fig. 2. Greedy Matching repeatedly removes the currently heaviest edge e and all of its adjacent edges from the input graph $G = (V, E)$ until E is empty. In each iteration e is added to the matching M which is returned as the soltution. It is easy to see that Greedy Matching has a performance ratio of $\frac{1}{2}$ [1]. If the edges of G are sorted in a preprocessing step then the runtime is $O(|E| \log |V|)$.

```
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 adjacent to e from E
6 end
7 return M
```

Fig. 2. The greedy algorithm for finding maximum weight matchings.

The second algorithm is the LAM algorithm by Preis [17]. A sketch of this algorithm is shown in Fig. 3. Preis improved upon the simple greedy approach by making use of the concept of a so called *locally heaviest edge*. This is defined as an edge for which no other edge currently adjacent to it has larger weight. Preis proved that the performance ratio of the algorithm based on this idea is $\frac{1}{2}$. He also showed how a locally heaviest edge in a graph can be found in amortized constant time. This results in a total runtime of $O(|E|)$ for the LAM algorithm. See [17] for more details.

```
LAM (G = (V, E), w : E → ℝ+)
1 M := ∅
2 while E ≠ ∅ do begin
3     find a locally heaviest edge e ∈ G
4     remove e and all edges adjacent to e from G
5     add e to M
6 end
7 return M
```

Fig. 3. The LAM approximation algorithm for finding maximum weight matchings.

The third algorithm is the Path Growing Algorithm (PGA) by Drake and Hougardy [5] shown in Fig. 4. PGA constructs node disjoint paths in the input graph G along heaviest edges by lengthening the path one node at a time. Each time a node is added to a path it and all of its incident edges are removed from the graph. This is repeated until the graph is empty. By alternately labelling the edges of the paths 1 and 2 one obtains two matchings M_1 and M_2, the larger of which is returned as the solution. The PGA algorithm has a performance ratio of $\frac{1}{2}$ and a runtime of $O(|E|)$ [5]. There are two simple improvements that are proposed in [5] which can be applied to the PGA algorithm without changing its runtime. The first is to compute a maximum weight matching along the paths constructed by the algorithm and return this as the solution. The second is to add any remaining edges in the graph to the solution until the solution becomes a maximal matching. Neither of these improvements can guarantee a better worst case behaviour for the algorithm, but in practice these improvements can make a considerable difference. Therefore we test both versions of this algorithm. We call the second version where both improvements are applied PGA$'$.

PathGrowingAlgorithm $(G = (V, E), w : E \to \mathbb{R}_+)$

```
1 M_1 := ∅, M_2 := ∅
2 while E ≠ ∅ do begin
3     choose x ∈ V of degree at least 1 arbitrarily
4     grow a path from x along heaviest edges added alternately to M_1 and M_2
5     remove the path from G
6 end
7 return max(w(M_1), w(M_2))
```

Fig. 4. The Path Growing Algorithm for finding maximum weight matchings.

Finally we include a trivial heuristic called MM (for Maximal Matching) shown in Fig. 5. This heuristic computes a maximal matching in a graph in a greedy manner. For each vertex x a heaviest edge e which is incident to x is added to the solution. All edges adjacent to e are removed. The runtime of algorithm MM is $O(|E|)$. It is easy to construct examples where the weighted matching returned by MM can be arbitrarily bad. Yet it is interesting to see how this heuristic behaves in practice. Therefore we have included it in this study.

4 Test Instances

We test our implementations of the above algorithms and local improvements against four classes of data sets: random graphs, two dimensional grids, complete graphs, and randomly twisted three dimensional grids. We do not include any geometric instances in this study as none of the algorithms considered here was designed to take advantage of any geometric properties.

```
MM (G = (V, E), w : E → ℝ+)
1 M := ∅
2 while E ≠ ∅ do begin
3     choose x ∈ V arbitrarily
4     add to M the heaviest edge e incident to x
5     delete all edges adjacent to e from E
6 end
7 return M
```

Fig. 5. The MM heuristic for finding maximum weight matchings.

Within each class instances with different parameters such as number of vertices, edge densities etc. have been generated. For each specific parameter set ten instances were randomly generated. Each of the algorithms was run on all ten instances and then the average value of the runtime and percentage of the deviation from an optimal solution was computed. We calculated the optimal solutions to these instances using LEDA [13]. The size of the test instances was restricted to graphs with at most 100,000 vertices and at most 500,000 edges because for these instances the runtime of the exact algorithm was already several hours.

The random graphs are based on the $G_{n,p}$ model. This means that the graphs have n vertices and the possibe $\binom{n}{2}$ edges are chosen independently with probability p. We have chosen $n = 10,000$ and p in the range from 5/10000 to 100/10000 resulting in graphs from about 25,000 to 500,000 edges. The edge weights are integer values chosen randomly from the range of 1 to 1000. The graphs are labelled as "R10000.5" through "R10000.100".

The two dimensional grids are grids of dimension $h \times 1000$ with h chosen in the range 10 to 100. The edge weights for these grids have been assigned integer values chosen randomly from the range between 0 and 999. These graphs are labelled "G10" through "G100".

The complete graphs are graphs on n vertices containing all possible $\binom{n}{2}$ edges. We have generated these graphs for $n = 200$ to 2000. The integer edge weights have been randomly assigned from the range of 0 to 999. These graphs are labelled "K200" through "K2000".

For the randomly twisted three dimensional grid we have used the RMFGEN graph generator introduced in [9] which was used to generate flow problems. The graphs created consist of a square grid of dimension a called a frame. There are b such frames $F_1, \ldots, F_b$ which are all symmetric. There are a^2 edges connecting the nodes of F_i to a random permutation of the nodes of F_{i+1} for $1 \leq i < b$. The edges within a frame all have weight 500. Those between frames have weights randomly chosen between 1 and 1,000. The only changes we have made to the RMFGEN generator besides making the graph undirected is concerning the weights of the in-frame edges. We have assigned 500 to these edges instead of $a^2 * 1000$ assigned by RMFGEN as the latter value did not seem to produce instances that were as interesting for the weighted matching problem. We have

created three such tests on graphs of dimension $a = 4$, $b = 1250$; $a = 27$, $b = 27$; and $a = 70$, $b = 4$. These three instances are labelled "a", "b" and "c".

5 Experimental Results

The following two subsections contain the experimental results we obtained on the test instances described in Section 4. We have compared the five algorithms described in Section 3 with and without additionally performing our local improvement operations which where applied as follows: For each of the seven local improvement operations shown in Fig. 1 we have computed a maximal set of disjoint improvements in time $O(m)$ as indicated in Theorem 2 and then augmented along this set. We used the same ordering of the operations as shown in Fig. 1.

For each row in a table ten different test instances have been generated and the average value has been taken. The variance was in all cases below 1%, in many cases even below 0.5%. Due to space restrictions we list the results for some part of the instances only.

5.1 Performance

Tables 1, 2, and 3 show the performances of the five algorithms on the different classes of test sets. The first column of the tables contains the name of the test instance as described in Section 4. The next two columns "n" and "m" denote the number of vertices and edges of the graph. In case of the graphs "R10000.x" the number of edges is the average value of the ten test instances that were computed for each row of the table. In all other cases the ten test instances have the same number of edges, only the weight of the edges differs. The next five columns show the difference in % of the solution found by the algorithms to the optimum solution. The names of the algorithms are abbreviated as in Section 3. Each row contains one value in bold which is the best value. The worst value is given in gray.

Table 1. Performances of the five algorithms on weighted random graphs with different densities. The values denote the difference from the optimum in %.

graph	n	m	Greedy	MM	LAM	PGA	PGA′
R10000.5	10000	25009	8.36	14.66	8.38	14.70	**7.38**
R10000.10	10000	49960	9.18	12.59	9.18	12.56	**8.31**
R10000.20	10000	100046	7.73	9.40	7.74	9.52	**7.20**
R10000.30	10000	150075	6.28	7.55	6.29	7.55	**5.95**
R10000.40	10000	200011	5.46	6.33	5.50	6.30	**5.08**
R10000.60	10000	299933	4.19	4.83	4.23	4.82	**3.99**
R10000.80	10000	399994	3.52	3.99	3.54	4.02	**3.39**
R10000.100	10000	499882	3.05	3.39	3.08	3.45	**2.95**

As can be seen from Table 1 there is a great difference in the quality of the PGA and PGA′ algorithm. The simple heuristics added to the PGA algorithm drastically improve its performance in practice. This observation also holds for all other test instances. On all random graph instances the PGA′ algorithm performs best. The LAM and Greedy algorithms have almost the same quality which is slightly worse than that of PGA′. For all algorithms the performance improves as the random graphs get denser. The only exception are the extremely sparse graphs "R10000.5". Such an effect also has been observed in the unweighted case [12].

Table 2. Performances of the five algorithms on weighted grid like graphs. The values denote the difference from the optimum in %.

graph	n	m	Greedy	MM	LAM	PGA	PGA′
G10	10000	18990	5.87	12.38	5.87	12.76	**4.51**
G20	20000	38980	5.97	13.04	5.98	12.50	**4.53**
G40	40000	78960	5.99	13.45	5.99	12.52	**4.60**
G60	60000	118940	5.96	13.68	5.97	12.49	**4.66**
G80	80000	158920	5.99	13.79	5.99	12.61	**4.67**
G100	100000	198900	6.05	13.82	6.05	12.60	**4.66**
a	20000	49984	5.71	10.22	**5.50**	12.91	5.59
b	19683	56862	5.79	8.79	**5.25**	13.14	6.14
c	19600	53340	6.13	8.00	**4.93**	12.71	6.22

Table 2 shows that the performances of all algorithms are independent of the size of the test instances. For the two dimensional grids the PGA′ algorithm achieves the best solutions. Again the Gredy algorithm and the LAM algorithm have almost the same quality which is significantly worse than that of PGA′. This situation changes in the case of the randomly twisted three dimensional grids. Here the LAM algorithm achieves the best result and the Greedy algorithm is slightly better than PGA′.

Table 3. Performances of the five algorithms on weighted complete graphs. The values denote the difference from the optimum in %.

graph	n	m	Greedy	MM	LAM	PGA	PGA′
K200	200	19900	1.75	1.63	1.77	1.65	**1.55**
K600	600	179700	0.72	0.82	**0.69**	0.75	0.72
K1000	1000	499500	**0.46**	0.55	0.49	0.53	0.51
K1400	1400	979300	0.39	0.38	0.39	0.37	**0.35**
K2000	2000	1999000	**0.27**	0.29	0.29	0.29	0.29

On weighted complete graphs all five algorithms have almost the same quality. For large complete graphs the performances of all algorithms tends to one.

This is of course not surprising as in complete graphs an algorithm can barely choose a 'wrong' edge.

Tables 4, 5, and 6 show the performances of the five algorithms on the different classes of test sets with local improvements applied. This means that we have taken the solution returned by the algorithms and then computed a maximal set of pairwise disjoint local improvements for each of the seven local improvements shown in Fig. 1. The quality of all algorithms is drastically improved by the local improvements. The deviation from the optimum solution is reduced by a factor between 1.5 and 3. The performances of the Greedy algorithm and the LAM and PGA′ algorithms are very similar after performing the local improvements.

Table 4. Performances of the five algorithms on weighted random graphs with different densities with local improvements applied. The values denote the difference from the optimum in %.

graph	n	m	Greedy	MM	LAM	PGA	PGA′
R10000.5	10000	25009	3.15	4.57	3.15	5.21	**3.07**
R10000.10	10000	49960	4.62	5.52	4.62	6.14	**4.53**
R10000.20	10000	100046	4.39	4.90	**4.39**	5.42	4.44
R10000.30	10000	150075	**3.75**	4.18	3.76	4.56	3.87
R10000.40	10000	200011	**3.22**	3.60	3.26	3.84	3.37
R10000.60	10000	299933	**2.53**	2.84	2.55	3.00	2.75
R10000.80	10000	399994	**2.11**	2.38	2.15	2.52	2.28
R10000.100	10000	499882	**1.85**	2.08	1.87	2.15	2.00

Table 5. Performances of the five algorithms on weighted grid like graphs with local improvements applied. The values denote the difference from the optimum in %.

graph	n	m	Greedy	MM	LAM	PGA	PGA′
G10	10000	18990	2.02	5.58	2.02	4.86	**1.79**
G20	20000	38980	2.11	6.06	2.11	4.94	**1.87**
G40	40000	78960	2.11	6.31	2.11	5.01	**1.91**
G60	60000	118940	2.12	6.47	2.12	5.01	**1.94**
G80	80000	158920	2.11	6.57	2.11	5.05	**1.94**
G100	100000	198900	2.16	6.56	2.16	5.05	**1.93**
a	20000	49984	2.52	6.00	**2.49**	6.42	3.00
b	19683	56862	2.81	6.43	**2.61**	7.31	3.48
c	19600	53340	2.99	6.01	**2.62**	7.54	3.77

We also tested how much improvement can be achieved by applying the local improvement operations as long as they are possible. Usually computing three rounds of maximal sets of these local improvements led to a matching satisfying the conditions of Theorem 1. In some cases up to 5 iterations were necessary.

Table 6. Performances of the five algorithms on weighted complete graphs with local improvements applied. The values denote the difference from the optimum in %.

graph	n	m	Greedy	MM	LAM	PGA	PGA′
K200	200	19900	0.81	0.98	**0.78**	0.92	0.94
K600	600	179700	0.38	0.48	**0.35**	0.45	0.42
K1000	1000	499500	**0.23**	0.32	0.26	0.31	0.31
K1400	1400	979300	**0.19**	0.22	0.19	0.21	0.20
K2000	2000	1999000	**0.14**	0.17	0.15	0.18	0.18

Using this approach the deviation from an optimum solution can be reduced by additional 30% on average. The runtime increases by a factor of 2 to 3.

5.2 Runtimes

We compared the runtimes of all five algorithms on all instances against each other and also compared it to LEDA's exact algorithm. Table 7 shows the relative runtimes of the algorithms LAM, PGA and PGA′ compared to the MM algorithm which is clearly the fastest. As can be seen there is no big difference between the runtimes of algorith PGA and PGA′. The PGA′ algorithm is about a factor 2, the LAM algorithm about a factor 3 slower than the MM algorithm. The factor by which the local improvement operations are slower than the MM algorithm is about 4. This means that the LAM and PGA′ algorithm with local improvements applied are about a factor of 7 and 6 slower than MM.

Table 7. Relative runtimes expressed as a range of the factor within which the runtime of the algorithm is slower than the MM algorithm.

MM	LAM	PGA	PGA′	local improvements
1.00	2.68 - 3.55	1.58 - 2.06	1.88 - 2.10	3.45 - 5.12

Table 8 shows the relative runtimes of the Greedy algorithm and LEDA's exact algorithm compared to the MM algorithm. The Greedy algorithm has a worst case runtime of $O(m \log m)$. The algorithm implemented in LEDA has a runtime of $O(mn \log n)$. Therefore one should expect that the Greedy algorithm is within a factor of $\log m$ and LEDA's algorithm is within a factor of $n \log n$ of the runtime of the MM algorithm. This behaviour can be roughly confirmed by the data.

To give an impression of the absolute running times we mention these for algorithm MM on some large graphs. It took 0.16 seconds on R10000.100, 0.06 seconds on G100 and 0.62 seconds on K2000. All times were measured on a 1.3GHz PC.

Table 8. Relative runtimes expressed as a factor by which the runtime of the Greedy algorithm and LEDA's exact algorithm is slower than the MM algorithm.

graph	n	m	Greedy	MM	LEDA	graph	n	m	Greedy	MM	LEDA
a	20000	49984	7.28	1.0	26119	G10	10000	18990	6.64	1.0	14873
b	19683	56862	6.98	1.0	27181	G20	20000	38980	9.19	1.0	27667
c	19600	53340	5.21	1.0	28944	G40	40000	78960	11.64	1.0	53737
R10000.5	10000	25009	5.71	1.0	11156	G60	60000	118940	13.11	1.0	79321
R10000.10	10000	49960	8.15	1.0	8255	G80	80000	158920	14.10	1.0	104473
R10000.20	10000	100046	10.55	1.0	6004	G100	100000	198900	15.13	1.0	129857
R10000.30	10000	150075	12.11	1.0	5096	K200	200	19900	11.59	1.0	266
R10000.40	10000	200011	13.44	1.0	4660	K600	600	179700	20.67	1.0	514
R10000.60	10000	299933	15.05	1.0	4286	K1000	1000	499500	23.54	1.0	775
R10000.80	10000	399994	15.68	1.0	3949	K1400	1400	979300	22.05	1.0	1006
R10000.100	10000	499882	15.99	1.0	3832	K2000	2000	1999000	18.65	1.0	1402

6 Conclusion

We have suggested a set of seven local improvement operations for weighted matchings in graphs which guarantee a performance ratio of $\frac{2}{3}$. A maximal set of such operations can be found in linear time. We compared five different approximation algorithms for the weighted matching problem. The algorithms MM, LAM, PGA, and PGA′ have linear runtime while the Greedy algorithm has runtime $O(m \log n)$. The PGA′ algorithm is significantly better than PGA. The computation of a maximum weight matching on the paths generated by PGA can easily be incorporated in the generation of these paths. Therefore the PGA′ algorithm requires almost no additional expense in coding or runtime and is definitely the better choice.

Only in the case of complete graphs are there a few instances where the Greedy algorithm achieved the highest quality. But in these cases the LAM and PGA′ algorithm are very close to these results. Therefore the higher runtime required by the Greedy algorithm does not justify its application. The LAM or PGA′ algorithm is a better choice. They usually should guarantee a solution within 5% of the optimum.

The local improvement operations introduced in this paper yield better performances for all five algorithms. On average the deviation from the optimum solution is reduced by a factor of 2. This improvement is achieved by more than doubling the runtimes of the linear time algorithms. Still these runtimes are dramatically smaller than those required by exact algorithms. Therefore the linear time LAM and PGA′ algorithms are definitely the best choice for applications where runtimes are of crucial importance. If better quality is needed our local improvement operations should be applied which increase the runtime of these two algorithms by roughly a factor of 2-3 only. By applying our local improvement operations as long as they are possible the distance from an optimal solution can be decreased by additional 30% while the runtime grows by a factor of 4.

References

1. D. Avis, A Survey of Heuristics for the Weighted Matching Problem, Networks, Vol. 13 (1983), 475–493
2. R. Beier, J.F. Sibeyn, A Powerful Heuristic for Telephone Gossiping, Proc. 7th Colloquium on Structural Information and Communication Complexity, Carleton Scientific (2000), 17–35
3. B. Chandra, M.M. Halldórsson, Greedy local improvement and weighted set packing approximation, Journal of Algorithms, 39 (2000), 223–240
4. W. Cook, A. Rohe, Computing minimum-weight perfect matchings, INFORMS Journal on Computing 11 (1999), 138–148
5. D.E. Drake, S. Hougardy, A Simple Approximation Algorithm for the Weighted Matching Problem, Information Processing Letters 85 (2003), 211–213
6. C. Frömmel, A. Goede, C. Gröpl, S. Hougardy, T. Nierhoff, R. Preissner, M. Thimm, Accelerating screening of 3D protein data with a graph theoretical approach, Humboldt-Universität zu Berlin, September 2002
7. H.N. Gabow, Data Structures for Weighted Matching and Nearest Common Ancestors with Linking, SODA 1990, 434–443
8. H.N. Gabow, R.E. Tarjan, Faster Scaling Algorithms for General Graph-Matching Problems, JACM 38 (1991), 815–853
9. G. Goldfarb, M.D. Grigoriadis, A Computational Comparison of the Dinic and Network Simplex Methods for Maximum Flow, Annals of Operations Research 13 (1988), 83-123
10. M.M. Halldórsson, Approximating Discrete Collections via Local Improvements, In Proc. of Sixth SIAM/ACM Symposium on Discrete Algorithms, San Francisco (1995), 160–169
11. J.E. Hopcroft, R.M. Karp, An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs, SIAM Journal on Computing 2 (1973), 225–231
12. J. Magun, Greedy Matching Algorithms, an Experimental Study, ACM Journal of Experimental Algorithms, Volume 3, Article 6, 1998
13. K. Mehlhorn, S. Näher, LEDA: A Platform for Combinatorial and Geometric Computing, ACM Press (1995), New York, NY
14. 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
15. R.H. Möhring, M. Müller-Hannemann, Complexity and Modeling Aspects of Mesh Refinement into Quadrilaterals, Algorithmica 26 (2000), 148–171
16. B. Monien , R. Preis , R. Diekmann, Quality Matching and Local Improvement for Multilevel Graph-Partitioning, Parallel Computing, 26(12), 2000, 1609–1634
17. 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
18. 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