Fast and Simple Algorithms for Weighted Perfect Matching

Mirjam Wattenhofer, Roger Wattenhofer

{*mirjam.wattenhofer,wattenhofer*}*@inf.ethz.ch, Department of Computer Science, ETH Zurich, Switzerland*

Abstract

We present two fast and simple combinatorial approximation algorithms for constructing a minimum-weighted perfect matching on complete graphs whose cost functions satisfy the triangle inequality. The first algorithm runs in $O(n^2 \log n)$ time and is at most a factor $\log n$ worse than an optimal solution. In the second algorithm, the average time until a node is matched is $O(n^2)$ and the approximation ratio is $\log^2 n$.

Key words: weighted matching, approximation algorithms

1 Introduction and Motivation

Let $G = K(V)$ be a complete graph of an even number of $|V| = n$ nodes and $w : E \to \mathbb{R}_+$ a non-negative cost function satisfying the triangle inequality. A *perfect matching* of G is a subset $M \subset E$ such that no two edges in M are adjacent and each node is incident to one edge in M. The *weight* $w(M)$ of a matching M is the sum of the weights of its edges. The problem is to find a perfect matching of minimum weight *(MM)*.

This problem can be solved in polynomial time by the algorithm of Edmonds [Edm65]. Though its running time was improved from $O(n^4)$ to $O(n(m + n \log n))$ by [Gab90] it is still too time-consuming for many applications. Hence, much effort was done to find good approximation algorithms for MM which are faster than the exact algorithm (e.g. [GW92], [VA99]). Besides its running time, Edmonds' algorithm has another disadvantage: By making use of the linear programming dual of the MM problem viewed as a linear program, the algorithm and its analysis are difficult to understand, implement, or teach. Unfortunately, most of the other exact or approximation algorithms for MM are based on Edmonds' algorithm or a linear programming approach. This almost inevitably leads to a less intuitive understanding of the algorithms and/or their analysis.

In this paper we will in a first step introduce a simple, purely combinatorial algorithm for MM, which yields a $2 \log n$ approximation in $O(n^2 \log n)$ time. In a next step, we are able to reduce the average time until a node is matched to $O(n^2)$ by losing a factor of $\log n$ in the approximation. Given that the only other simple MM algorithm we are aware of, the greedy algorithm, has an approximation ratio of $\Omega(n^{0.58})$ [RT81] and that the fastest approximation algorithms need time at least $O(n^2\sqrt{n\alpha(m,n)\log n})$[GT91], respectively $O(n^2 \log n)$ [GW92], also on average, we believe that our results are of two-fold interest. On the one hand the algorithms and their analysis are intuitive, and could be used as a stepping stone to explain more intricate algorithms. On the other hand, the approximation ratio and running

time of our algorithms slightly improve the currently best known results for polylogarithmic approximations. Altogether, due to their simplicity and efficiency, we hope that our algorithms or derivations thereof will be applicable in the distributed, online, and/or mobile computing setting.

2 The Algorithms

2.1 Description

The main algorithm, Algorithm 1 (Perfect Matching), takes as input an undirected graph $K(V)$ with edge costs $w_e \geq 0$ satisfying the triangle inequality and outputs a perfect matching M. The basic structure of the algorithm involves maintaining a forest F of edges which is initially empty. In each iteration ϕ of the algorithm's while-loop a set of edges is selected connecting distinct connected components of F. Define an *active component* to be any connected component C of F with an odd number of nodes, else call the component *inactive*. The while-loop terminates when all connected components C of F are inactive. The approximation properties of the algorithm will follow from the way we choose the edges in each iteration ϕ and which algorithm we call as a subroutine to compute the matching on the forest F. First to the edge-selection step: In each iteration the algorithm connects every active component to the *nearest* other active component, where nearest is meant in the sense of minimizing the weight of the edges to be added (see Figure A.1). Now to the matching-subroutines: Algorithm 2 (Idle Match) computes the matching on the forest F after the termination of the while-loop of Algorithm 1, whereas Algorithm 3 (Instant Match) matches the nodes in an ongoing fashion in each iteration of the while-loop. Thus, decreasing the average time until a node is matched significantly but naturally losing some accuracy.

Algorithm 1 Perfect Matching

Input: an undirected graph $G = K(V)$ with edge costs $w_e \geq 0$
Output: a matching M

1: (∗ depending on preferences either call line 11 or line 14 but not both to compute matching ∗)
2: $F \leftarrow \emptyset$, $M \leftarrow \emptyset$, $\phi = 0$,
3: $\mathcal{C}_\phi \leftarrow \{\{v\} : v \in V\}$
4: **while** ($\exists$ active component $C \in \mathcal{C}_\phi$) **do**
5: $\quad E_{\mathcal{C}_\phi} \leftarrow \emptyset$
6: $\quad$ **for** each active component $C \in \mathcal{C}_\phi$ **do**
7: $\quad\quad$ find a path P from C to an active component C', $C \neq C'$, that minimizes $w(E_C)$, where $E_C = P - F$
8: $\quad\quad E_{\mathcal{C}_\phi} \leftarrow E_{\mathcal{C}_\phi} \dot{\cup} E_C$
9: $\quad$ **end for**
10: $\quad F \leftarrow F \cup E_{\mathcal{C}_\phi}$
11: $\quad \boldsymbol{M} \leftarrow$ Algorithm 3 with Input $(\boldsymbol{F}, \boldsymbol{M}, \boldsymbol{\phi})$
12: $\quad \phi \leftarrow \phi + 1$
13: $\quad$ update $\mathcal{C}_\phi$
14: **end while**;
15: $\boldsymbol{M} \leftarrow$ Algorithm 2 with Input $\boldsymbol{F}$

Algorithm 2 Idle Match

Input: a forest F

Output: a perfect matching M

1: duplicate every edge of each component of F and shortcut to obtain a collection of cycles (see Figure A.2) (∗ the cycles have even length since all components are inactive ∗)
2: keep the best matching M out of the two matchings defined by every cycle

Algorithm 3 Instant Match

Input: a forest F, a partial matching M, ϕ

Output: a partial matching M

1: **if** $(\phi = 0)$ **then**
2: duplicate every edge of each component of F and shortcut to obtain a collection of (maybe odd-length) cycles
3: for every odd cycle fix an arbitrary node to remain unmatched
4: keep the best matching out of the two (partial) matchings defined by every cycle (and fixed node)
5: **else**
6: match free nodes in every component, s.t. the paths between matched nodes are disjoint (∗ See Lemma 5 for how to construct such paths. ∗)
7: **end if**
8: update M

2.2 Analysis

In this section we prove the approximation properties of the Perfect Matching Algorithm (Algorithm 1) in connection with the Idle Match Algorithm (Algorithm 2), respectively the Instant Match Algorithm (Algorithm 3). By using the Idle Matching Algorithm the matching we compute comes within a factor of $\log n$ of optimal. Whereas, for Algorithm 3 we prove an approximation ratio of $\log^2 n$. In the next section we show that the running time of the algorithms is $n^2 \log n$, but that for the Instant Match Algorithm the average time until a node is matched is $O(n^2)$.

Lemma 1 *The while-loop of the Perfect Matching Algorithm is executed at most* $\log n$ *times.*

PROOF. At least three active components must be connected to each other such that the new component is active again. Therefore, in each iteration the number of active components decreases by a factor of at least three and the logarithmic number of executions follows.

The incidence vector of the forest F produced by the Perfect Matching Algorithm is a feasible solution for following integer linear programm *(IP)*:

$$
\begin{array}{lllll}
(IP) & \text{Min} & \sum_{e \in E} w_e x_e & & \\
 & \text{subject to:} & x(\delta(S)) \geq |S| \mod 2 & & \emptyset \neq S \subset V \\
 & & x_e \in \{0,1\} & & e \in E,
\end{array}
$$

where $\delta(S)$ denotes the set of edges having exactly one endpoint in S and $x(F) = \sum_{e \in F} x_e$. The optimal solution F^* to *(IP)* is obviously of less weight than is the optimal solution M^* of the minimum-weighted perfect matching problem.

Lemma 2 *The weight of the set of edges added in each execution of the while-loop of the Perfect Matching Algorithm is at most the weight of the optimal forest F^*: $w(E_{\mathcal{C}_\phi}) \leq w(F^*)$.*

PROOF. We will argument about an arbitrary iteration ϕ. Construct a graph H by considering the inactive and active components of this iteration as nodes in H. We conceptually divide the edges of H into *red* and *blue* edges. The edges $e \in F^* \cap \delta(C)$ for all $C \in \mathcal{C}_\phi$ are the *red* edges of H. The edges $e \in E_{\mathcal{C}_\phi}$ added in this iteration to F are the *blue* edges[1]. Keeping in mind that the incidence vector of edges of F^* must be a feasible solution for (IP) we know that for each node v in H associated with an active component (a so called *active node*) there is a path P_v^r, consisting of red edges only, which connects this node to another active node in H. By the way we chose $E_{\mathcal{C}_\phi}$ we know that there is also a blue path P_v^b connecting v to another active node in H which is at most as heavy as the corresponding red path: $w(P_v^b) \leq w(P_v^r)$. Consequently, $\sum_{v \in V_a} w(P_v^b) \leq \sum_{v \in V_a} w(P_v^r)$, where V_a is the set of nodes in H associated with active components. Since, $\sum_{v \in V_a} w(P_v^r) \leq 2w(F^*)$ and $2w(E_{\mathcal{C}_\phi}) = \sum_{v \in V_a} w(P_v^b)$ the lemma follows. (The factors of two arise because each path is counted twice, once for each endpoint.)

Corollary 3 *The weight of the forest F at the termination of the while-loop of the Perfect Matching Algorithm is at most $\log n \cdot w(F^*)$.*

PROOF. The Corollary follows immediately from Lemma 1 and Lemma 2.

Theorem 4 *The matching computed by the Idle Matching Algorithm is a $\log n$-approximation of the minimum-weighted perfect matching M^*.*

PROOF. We compute an Euler tour with shortcuts on each component and choose the better out of the two possible matchings for each tour. Since the weight of an Euler tour is at most twice the weight of the component, the weight of the better matching is at most the weight of the component. All together the matching has weight at most $w(F)$, which is by Corollary 3 at most $\log n \cdot w(F^*)$. Since $w(F^*) \leq w(M^*)$ we can deduce the theorem.

Before we analyze the Instant Match Algorithm (Algorithm 3) we need following helper lemma.

Lemma 5 *Given a tree $T = (V, E)$ and a set of marked nodes $S = \{v_1, v_2, \ldots\} \subseteq V$. Let $S_p = \{(v_i, v_j), \ldots\} \subset \binom{S}{2}$ be a disjoint pairing of the nodes in S and for each pair (v_i, v_j), p_{ij} the path connecting v_i to v_j. Then it is always possible to construct S_p in such a way that the paths p_{ij} are mutually edge-disjoint.*

PROOF. We prove the lemma by giving a construction for S_p. Beginning from the leaves, pair v_i with v_j if they have a common ancestor v_a and there is no un-paired node v_k which has a closer common ancestor v_b with either v_i or v_j, where closer is meant in the sense of the length of the path between the nodes. This construction leads to all desired properties of the set S_p.

Lemma 6 *The weight of the edges added to the matching M at the end of the Instant Match Algorithm is at most the weight of the input forest F.*

PROOF. If $\phi = 0$ we basically do the same as in the Idle Match Algorithm, except for some cycles being of odd length, which does not have any implications on the weight of the matching. Hence, the weight of the edges added to the matching is at most the weight of the components of F, which is $w(F)$. For $\phi > 0$ we know by Lemma 5 that we can match the free nodes (except for one in each active component) in a component of F such that

[1] Note that some edges may be red as well as blue.

the paths between matched nodes are disjoint. Furthermore, by the triangle inequality we know that connecting the nodes directly is at most as expensive as connecting them via other nodes. Consequently, for each component of F the weight of the edges added to the matching is at most the weight of the component itself and the lemma follows.

Theorem 7 *The matching computed by the Instant Match Algorithm is a* $\log^2 n$*-approximation of the minimum-weighted perfect matching* M^**.*

PROOF. By Lemma 2 the weight of the forest in iteration ϕ is at most

$$w(F) = w(\mathcal{C}_\phi) = w(\mathcal{C}_{\phi-1}) + w(E_{\mathcal{C}_{\phi-1}}) \leq w(\mathcal{C}_{\phi-1}) + w(F^*) \leq \phi \cdot w(F^*).$$

By Lemma 6 we have:

$$\begin{aligned} w(M) &\leq \sum_{\phi=1}^{\log n} w(\mathcal{C}_\phi) \leq \sum_{\phi=1}^{\log n} \phi \cdot w(F^*) \\ &= \frac{1}{2}(\log^2 n + \log n) \cdot w(F^*) \leq \log^2 n \cdot w(M^*) \end{aligned} \quad .$$

2.3 Implementing the Algorithms

We now turn to the problem of implementing the algorithms efficiently. Both, the Idle Match Algorithm and the Instant Match Algorithm, have a running time of $O(n)$. Thus, the critical step of the implementation is the while-loop of the Perfect Matching Algorithm. More specific, in each iteration the crucial part is to find the paths P and to merge the components in the update of $\mathcal{C}_\phi$. If we maintain the components C as a union-find structure of nodes, merging will take at most $O(n\alpha(n,n))$ time, α being the inverse Ackermann function [Tar75]. By using a generalized Voronoi diagram the time to find the shortest path for all active components in an iteration is $O(n^2)$ [SPR80]. Since the while-loop is executed at most $\log n$ times, after $O(n^2 \log n)$ time steps, the Perfect Matching Algorithm terminates, independent of whether we use the Idle Match or the Instant Match Algorithm as a subroutine.

In the following we will prove that the average time until a node is matched decreases by a factor of magnitude $\log n$ if we use the Instant Match Algorithm.

Theorem 8 *After on average at most two iterations of the while-loop of the Perfect Matching Algorithm and using the Instant Match Algorithm as a subroutine a node is matched.*

PROOF. After the first iteration at least $\frac{2}{3}$ of all nodes are matched, since at most one node remains unmatched in each active component. By the same reasoning, after iteration ϕ at least $(1 - (\frac{1}{3})^\phi) \cdot n$ nodes are matched. For the average matching time we get:
$E[\#\text{ iterations until a node is matched}] = \sum_{\phi=1}^{\log n} (1 - (\frac{1}{3})^\phi)(1 - (\frac{1}{3})^{\phi-1}) \cdot \phi = \sum_{\phi=1}^{\log n} 2(\frac{1}{3})^\phi \cdot i \leq \frac{3}{2}$.

References

[Edm65] J. Edmonds. Paths, trees and flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965.

[Gab90] H.N. Gabow. Data structures for weighted matching and nearest common ancestors with linking. *in Proceedings of the 1st Annual ACM-SIAM Symposium on Discrete Algorithms*, 1990.

[GT91] H. N. Gabow and R. E. Tarjan. Faster scaling algorithms for general graph matching problems. *Journal of the ACM*, 38(4):815–853, 1991.

[GW92] M.X. Goemans and D.P. Williamson. A general approximation technique for constrained forest problems. In *SODA: ACM-SIAM Symposium on Discrete Algorithms (A Conference on Theoretical and Experimental Analysis of Discrete Algorithms)*, 1992.

[RT81] E.M. Reingold and R.E. Tarjan. On a greedy heuristic for complete matching. *SIAM Journal on Computing*, 10:676–681, 1981.

[SPR80] K.J. Suppowit, D.A. Plaisted, and E.M. Reingold. Heuristics for weighted perfect matching. *in Proceedings of the 12th Annual ACM Symposium on Theory of Computing*, pages 398–419, 1980.

[Tar75] R.E. Tarjan. Efficiency of a good but not linear set union algorithm. *Journal of the ACM*, 22:215–225, 1975.

[VA99] K.R. Varadarajan and P.K. Agarwal. Approximation algorithms for bipartite and non-bipartite matching in the plane. In *SODA: ACM-SIAM Symposium on Discrete Algorithms (A Conference on Theoretical and Experimental Analysis of Discrete Algorithms)*, 1999.

A APPENDIX

A.1 Visualization of Matching Algorithms

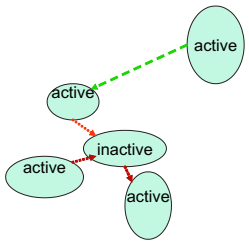

Fig. A.1. Connecting Active Components

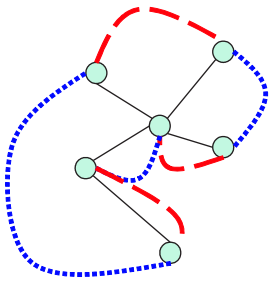

Fig. A.2. Duplicating and Shortcutting