Rank-Maximal Matchings

Robert W. Irving* Telikepalli Kavitha† Kurt Mehlhorn†
Dimitrios Michail† Katarzyna Paluch‡

Abstract

Suppose that each member of a set $\mathcal{A}$ of applicants ranks a subset of a set $\mathcal{P}$ of posts in an order of preference, possibly involving ties. A *matching* is a set of (applicant, post) pairs such that each applicant and each post appears in at most one pair. A *greedy* matching is one in which the maximum possible number of applicants are matched to their first choice post, and subject to that condition, the maximum possible number are matched to their second choice post, and so on. This is a relevant concept in any practical matching situation and it was first studied by Irving [8].

We define the bipartite graph $G = (\mathcal{A} \cup \mathcal{P}, \mathcal{E})$, where $\mathcal{E}$ consists of all pairs (a, p) such that post p appears in the preference list of applicant a. Each edge (a, p) has a rank i, which means that post p is an ith choice for applicant a. The traditional solution of computing a greedy matching in G would be to use the Hungarian algorithm to compute a maximum weight matching by assigning a suitably steeply decreasing sequence of weights to the edges. This would result in an algorithm with worst case running time $rn(m + n \log n)$ and the space requirement $\Theta(rm)$, where n is the number of vertices, m is the number of edges and r is the largest rank of an edge.

Here, we describe two algorithms to compute a greedy matching that improve upon this algorithm. We give a combinatorial algorithm with running time $O(\min(n + C, C\sqrt{n})m)$, where $C \leq r$ is the maximal rank of an edge used in a greedy matching. This algorithm works in phases and uses the maximum cardinality matching algorithm. We also give an $O(Cnm)$ algorithm that tackles the problem of large edge weights introduced by the Hungarian algorithm. This algorithm uses scaling and works in phases. The space requirement of both these algorithms is $O(m)$.

1 Introduction

1.1 The problem

Let $\mathcal{A}$ be a set of *applicants* and $\mathcal{P}$ be a set of *posts*, and suppose that, associated with each member of $\mathcal{A}$ is a partially ordered preference list comprising a subset of the elements of $\mathcal{P}$. A *matching* of $\mathcal{A}$ to $\mathcal{P}$ is an allocation of each applicant to at most one post so that each post is filled by at most one applicant; in other words it is a matching in the bipartite graph $G = (\mathcal{A} \cup \mathcal{P}, \mathcal{E})$, where $\mathcal{E}$ consists of all pairs (a, p) such that post p appears in the preference list of applicant a.

*Department of Computing Science, University of Glasgow, UK. rwi@dcs.gla.ac.uk

†Max-Planck-Institut für Informatik, Saarbrücken, Germany. {kavitha, mehlhorn, michail}@mpi-sb.mpg.de

‡Institute of Computer Science, University of Wroclaw, Poland. abraka@ii.uni.wroc.pl. Work done while the author was at MPII supported by Marie Curie Doctoral Fellowship

Each edge (a, p) has a rank i, which means that post p is an ith choice for applicant a. In any applicant a's list, there may be any number of ith choice posts, even zero. We believe that this is the natural way of formulating the problem of allocation of projects to students and the allocation of probationary posts to trainee teachers, for instance.

In the case where preferences are expressed on both sides, we have the notion of various kinds of *stability* to describe optimal matchings. This is the domain of *stable matching* problems, which have been studied extensively [2, 4, 5, 7, 9, 13]. When preferences are expressed on one side only (only applicants have preferences over posts), a number of different kinds of optimality can be defined. Here we study the notion of *greedy* or *rank-maximal* matchings, introduced in [8].

Let r be the largest rank that an applicant uses to rank any post.

Definition 1 *The* signature $\rho(M)$ *of a matching M is defined to be the r-tuple $(x_1, ..., x_r)$ where for each $1 \leq i \leq r$, x_i is the number of applicants who are matched in M with one of their ith choice posts.*

As a matter of convenience, we abbreviate a signature $(x_1, ..., x_r)$ by $(x_1, ..., x_d)$ if $x_d > 0$ and $x_i = 0$ for $i = d + 1, ..., r$.

We define a total order $\prec$, similar to lexicographic order, on signatures as follows: $(x_1, ..., x_r) \prec (y_1, ..., y_r)$ if $x_i = y_i$ for $1 \leq i < k$ and $x_k < y_k$, for some k. Denote by $\mathcal{M}$ the set of all matchings of $\mathcal{A}$ to $\mathcal{P}$.

Definition 2 *A matching that has the maximum signature under this ordering is a* greedy *or* rank-maximal *matching. Alternately, or equivalently, define $\mathcal{M}_1$ to be the subset of $\mathcal{M}$, in which the maximum possible number of applicants are matched to their first choice post. For $i = 2, 3, ..., r$ define $\mathcal{M}_i$ to be the subset of $\mathcal{M}_{i-1}$ in which the maximum possible number of applicants are matched to their ith choice post. A matching that belongs to $\mathcal{M}_r$ is a* greedy *or* rank-maximal *matching.*

For a given problem instance, there might be more than one greedy matching, but it is a consequence of the definition that all greedy matchings must have the same size.

It is easy to see that a simple greedy algorithm, in which we assign the maximum number of applicants to their first choice post, then the maximum number to their second choice post, and so on, is by no means guaranteed to lead to a greedy matching.

1.2 Previous Results and New Results

As mentioned earlier, the case where preferences are expressed by both sides has been extensively studied in the area of stable matching problems. When preferences are expressed by only one side, Irving in [8] considers the problem of computing a greedy matching in instances where the preference list for any applicant $a \in \mathcal{A}$ is strictly ordered i.e., there are no ties in a's list. The running time of the greedy matching algorithm in [8] is $O(\sum_{k=1}^{C} k^2(x_k + 1)(n + s_k))$, where $(x_1, ..., x_C)$ is the signature of a greedy matching and $s_k = x_1 + \cdots + x_k$. The worst case complexity of this algorithm in the case where all preference lists are of length bounded by d, is $O(d^3n^2)$.

We give a simple combinatorial algorithm with running time $O(\min(n + C, C\sqrt{n})m)$ for constructing a greedy matching.

We also give an $O(Cnm)$ algorithm for constructing a greedy matching by transforming this problem to a maximum weight matching problem. The algorithms given here illustrate two different approaches to compute a greedy matching.

1.3 Techniques

Our combinatorial algorithm runs in phases. Let the edge set of G be $\mathcal{E} = \mathcal{E}_1 \cup \mathcal{E}_2 \cup ... \cup \mathcal{E}_r$, where $\mathcal{E}_i$ is the set of edges of rank i. In phase i, our algorithm constructs a greedy matching M_i in the graph $G_i = (\mathcal{A} \cup \mathcal{P}, \mathcal{E}_1 \cup \mathcal{E}_2 \cup ... \cup \mathcal{E}_i)$. The algorithm computes M_{i+1} by computing a maximum matching in a suitable subgraph of G_{i+1} and by augmenting M_i.

Note that a greedy matching is very different from a maximum cardinality matching but we modify the graph G_{i+1} so as to transform the greedy matching problem to that of computing a maximum matching.

A greedy matching can also be found by transforming the input to an instance of the classical maximum weight bipartite matching problem. This involves allocating a suitably steeply decreasing sequence of weights to the edges. For instance, giving a weight of n^{r-1} to the edge (a, p) if p is a first choice of a, a weight of n^{r-2} if p is a second choice of a, and so on. But the use of such large integers as edge weights implies that an arithmetic operation might cost up to $\Theta(r)$ time and the space requirement becomes $\Theta(rm)$. We present a scaling algorithm, which works in phases, to tackle the problem of large edge weights. This algorithm has a running time of $O(Cnm)$ and it uses $O(m)$ space.

Organization of the Paper: In Section 2 we describe the combinatorial algorithm and analyse it. In Section 3 we describe the scaling algorithm and its analysis. Section 4 contains some concluding remarks and open problems.

2 A Combinatorial Algorithm

In this section we present a combinatorial algorithm for computing a greedy matching in a bipartite graph $G = (\mathcal{A} \cup \mathcal{P}, \mathcal{E})$. Before presenting the algorithm, let us examine the structure of the problem and build some intuition.

For the time being, let us assume that, for every i, $\mathcal{E}_i$ contains exactly one edge incident to any $a \in \mathcal{A}$. First, we can notice that in this case we can tell at once how many edges from $\mathcal{E}_1$ belong to a greedy matching as well as which nodes from $\mathcal{P}$ are matched by them. Let us denote them by $\mathcal{P}_1$. If some vertex $p \in \mathcal{P}_1$ is connected through $\mathcal{E}_1$ edges to more than one vertex $a \in A$, then we only know that in a greedy matching, one of them must be matched with p through $\mathcal{E}_1$. We do not know, however, which of them gets assigned to p and which cannot. Nevertheless, without any harm, we can delete from the graph all the edges of rank higher than 1 incident to vertices belonging to $\mathcal{P}_1$.

We can observe that if we match all the nodes in $\mathcal{P}_1$ through $\mathcal{E}_1$ edges arbitrarily, delete edges of rank higher than one incident to nodes in $\mathcal{P}_1$ and then extend this matching along augmenting paths, then the number of $\mathcal{E}_1$ edges in the matching will not change.

What can we say about $\mathcal{E}_2$ edges? Here we may sometimes be uncertain which vertices belonging to $\mathcal{P}$ should be matched through $\mathcal{E}_2$ edges in a greedy matching. It is so, for example, in Figure 1. We know that p_2 is matched through an $\mathcal{E}_1$ edge but we do not know a priori which of p_1, p_3 gets matched with an $\mathcal{E}_2$ edge. For a_1 and a_2 we can notice that they must be matched by edges of rank at most two in a greedy matching. Hence, we can delete all edges ranked higher than 2 incident upon a_1, a_2. But do deletions of this kind suffice?

We would like to delete all edges that we know will never belong to any greedy matching in order to reduce the greedy matching problem to a maximum matching problem in the reduced graph. We will show that we can determine which edges should be deleted using some well-known notions and facts from matching theory. We also drop the assumption that for every i, $\mathcal{E}_i$ contains exactly one edge incident to any $a \in A$.

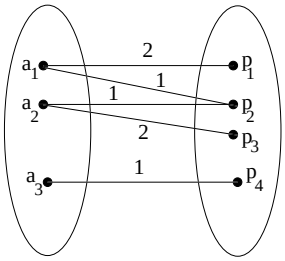

Figure 1: One of p_1, p_3 can be matched with a rank 2 edge.

Let M be a maximum matching in a bipartite graph G'. We will show that the vertex set of G' can be partitioned into three disjoint sets: E, O, and U. Nodes in E, O, and U are called *even, odd,* and *unreachable*, respectively [3]. E (O) are the nodes that can be reached in G' from a free node by an even (odd) length alternating path (with respect to M), and U are the nodes that cannot be reached from a free node by any alternating path. In Figure 1, $\{(a_1, p_1), (a_2, p_2), (a_3, p_4)\}$ is a maximum matching. Then the even nodes are $\{p_1, p_2, p_3\}$. The odd nodes are $\{a_1, a_2\}$ and the unreachable nodes are $\{a_3, p_4\}$.

The following lemma is well-known in matching theory. We include its proof for completeness.

Lemma 1 *The sets E, O and U are pairwise disjoint. Every maximum matching in G' pairs the nodes in U, matches all nodes in O, matches each node in O with a node in E, and has cardinality equal to $|O| + |U|/2$. There is no edge in G' connecting a node in E with a node in U, or between two nodes of E. No maximum matching in G' uses an edge connecting two nodes in O or a node in O with a node in U.*

Proof: Assume a node v is reachable by an even length alternating path from the free node a and by an odd length alternating path from the free node b. Then v is on the same side as a and the composition of the paths is an augmenting path from a to b. Thus M is not maximum, a contradiction.

Every node not reachable by an alternating path must be matched (otherwise it would be reachable by a path of length zero) and hence must be matched with a node which is also unreachable. M matches the nodes in O with nodes in E. Thus the cardinality of M is $|O| + |U|/2$.

Consider any maximal matching N. Then $M \oplus N$ consists of a set of alternating cycles and paths. Augmenting any such paths and cycles to M leaves the odd and the unreachable nodes matched and also does not change the status of any node.

Nodes in E are reachable by even length alternating paths. Such paths end in a matching edge. An edge connecting a node in E to a node in U is non-matching and hence could be used to extend the alternating path, a contradiction to the definition of U.

Since nodes in E are reachable by alternating paths ending in a matching edge, if there is an edge between two nodes of E, then it is a non-matching edge and we can use it to construct an augmenting path. This contradicts the maximality of M.

Since any maximum matching pairs the nodes in U and matches nodes in O with nodes in E, no maximum matching uses an edge connecting two odd nodes or an odd node with an unreachable node. ∎

The above facts formalise the ideas that we developed at the beginning of this section. Consider the edge set $\mathcal{E} = \mathcal{E}_1 \cup \mathcal{E}_2$. We first determine a maximum matching M_1 in $G_1 = (\mathcal{A} \cup \mathcal{P}, \mathcal{E}_1)$ and identify the sets of odd, even and unreachable vertices: O_1, E_1, and U_1. Suppose we remove all rank 2 edges incident upon nodes in $O_1 \cup U_1$, remove rank 1 edges between two nodes of O_1 or a node of O_1 and a node of U_1 and then augment M_1 to obtain a matching M_2. Since M_2 is obtained by augmenting M_1, nodes matched in M_1 are still matched in M_2. So, vertices in O_1 and U_1 are still matched. By virtue of the edges that we removed, we know that each node in O_1 has to be matched by a rank 1 edge to a vertex in E_1 and each node in U_1 is matched to another node in U_1 by a rank 1 edge. Hence, M_2 has at least $|O_1| + |U_1|/2$ nodes matched by rank 1 edges. So, this way we preserve the number of nodes that are matched by rank 1 edges. If we prove that every edge that we removed can never occur in a greedy matching and since M_2 is a maximum matching in the remaining graph, then we can see that M_2 is indeed a greedy matching in $(\mathcal{A} \cup \mathcal{P}, \mathcal{E}_1 \cup \mathcal{E}_2)$.

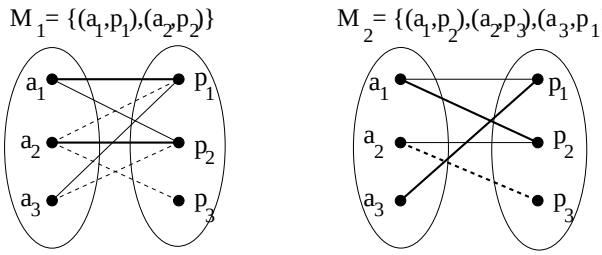

Figure 2: A simple example.

Let us consider the simple example shown in Figure 2, where the bold lines indicate rank 1 edges and the dashed lines indicate rank 2 edges. $M_1 = \{(a_1, p_1), (a_2, p_2)\}$ is a maximum matching in the graph G_1 consisting of only rank 1 edges. $O_1 = \{p_1, p_2\}, E_1 = \{a_1, a_2, a_3, p_3\}, U_1 = \emptyset$. Now we remove the edges (a_3, p_2) and (a_2, p_1). Then we augment M_1 to obtain M_2 which consists of two edges of $\mathcal{E}_1$ and one edge of $\mathcal{E}_2$. If we had augmented M_1 without removing the rank 2 edge $\{a_3, p_2\}$, we would have ended up in a maximum matching with one edge of $\mathcal{E}_1$ and two edges of $\mathcal{E}_2$ - not a greedy matching.

We now present the following iterative algorithm for constructing a greedy matching M. In the i-th iteration the algorithm constructs a greedy matching M_i in $G_i = (\mathcal{A} \cup \mathcal{P}, \mathcal{E}_1 \cup \ldots \cup \mathcal{E}_i)$. If $(s_1, s_2, \ldots, s_i, \ldots)$ is the signature of a greedy matching, then M_i has signature $(s_1, s_2, \ldots, s_i)$. We start with $G'_1 = G_1$, and M_1 any maximum matching in G'_1.

For $i = 1$ to $r - 1$ do the following steps, and output M_r.

1. Partition the nodes of $A \cup B$ into three disjoint sets: E_i, O_i, and U_i. E_i and O_i, as defined before, are the nodes that can be reached in G'_i from a free node by an even or odd length alternating path (with respect to M_i), respectively, and U_i are the nodes that cannot be reached from a free node by an alternating path.

2. Delete all edges incident to a node in $O_i \cup U_i$ from $\mathcal{E}_j, \forall j > i$. $O_i \cup U_i$ are the nodes that are matched by every maximum matching of G'_i. Delete all edges in G'_i connecting two nodes in O_i or a node in O_i with a node in U_i. These are the edges that are not used by any maximum matching of G'_i. Add the edges in $\mathcal{E}_{i+1}$ to G'_i. Call the resulting graph G'_{i+1}.

3. Determine a maximum matching M_{i+1} in G'_{i+1} by augmenting M_i. (Note that M_i is still contained in G'_{i+1}.)

Figure 3: A combinatorial algorithm for computing a rank maximal matching

Observe that the algorithm maintains the invariant that M_i is a maximum matching in G'_i. We will show in the proof of correctness that the following invariants are also maintained:

- every greedy matching in G_i has all its edges in G'_i
- M_i is a greedy matching in G_i.

The proof of correctness

We start with the following lemma, which proves that the edges that are deleted during phase $i+1$ do not belong to any greedy matching of G_{i+1}, provided that we maintain the invariants until the end of phase i.

Lemma 2 *Suppose that every greedy matching of G_i is a maximum matching of G'_i. Then every greedy matching of G_{i+1} is contained in G'_{i+1}.*

Proof: We need to show that the edges that we removed in the $(i+1)$th phase of the algorithm do not belong to any greedy matching of G_{i+1}.

Let N_{i+1} be any greedy matching of G_{i+1}. Then its signature is $(s_1, s_2, ..., s_i, s_{i+1})$. $N_i = N_{i+1} \cap \mathcal{E}_{\leq i}$ is a matching with signature $(s_1, s_2, ..., s_i)$ and is therefore, a greedy matching of G_i. So, N_i is a maximum matching in G'_i by the assumption. By Lemma 1, it has to pair the nodes in U_i and match all nodes in O_i with nodes in E_i.

Thus, N_i does not use any edge of G_i connecting two nodes in O_i or a node in O_i with a node in U_i. Since N_{i+1} is a matching, it cannot use any such edge either, and moreover, it cannot use any edge of rank higher than i incident to some node in $O_i \cup U_i$. So, N_{i+1} is contained in G'_{i+1}. ∎

In addition, the deletions guarantee that the number of edges of each smaller rank is preserved throughout the algorithm:

Lemma 3 *For every i, j such that $j > i$, the number of edges of rank at most i is the same in M_i and M_j.*

Proof: Since M_j is obtained from M_i by successive augmentations, every vertex matched by M_i is also matched by M_j. Hence, all nodes in U_i and O_i are matched in M_j.

Since G'_j has no edges of rank greater than i incident to nodes in O_i and U_i and no edges of rank at most i connecting two nodes in O_i or a node in O_i with a node in U_i, M_j must pair the nodes in U_i and must match nodes in O_i with nodes in E_i. And these edges have rank at most i. So, M_j has at least as many edges of rank $\leq i$ as M_i and M_j cannot have more since all the edges of rank $\leq i$ in G'_j belong to G'_i and M_i is a maximum matching in G'_i. ∎

Now, we are ready to prove the correctness of our algorithm.

Lemma 4 *For every k, the following statements hold:*
(i) *Every greedy matching in G_k is a maximum matching in G'_k;*
(ii) *M_k is a greedy matching in G_k.*

Proof: We prove this by induction on k. Since all edges in G_1 have the same rank, a greedy matching in G_1 is the same as a maximum matching. Since M_1 is a maximum matching in G_1, both statements hold for $k = 1$.

Let us now prove these statements for $i + 1$ assuming them to be true for i. Since M_i is a greedy matching in G_i, its signature is $(s_1, \ldots, s_i)$. Suppose the signature of M_{i+1} is $(r_1, \ldots, r_i, r_{i+1})$. By Lemma 3, we know that for every k between 1 and i, $\sum_{j=1}^{k} s_i = \sum_{j=1}^{k} r_i$.

It follows that the signature of M_{i+1} is $(s_1, \ldots, s_i, r_{i+1})$ for some $r_{i+1} \leq s_{i+1}$.

By the induction hypothesis, every greedy matching of G_i is a maximum matching of G'_i. Hence, by Lemma 2, any greedy matching of G_{i+1} is contained in G'_{i+1}. Thus there is a matching of cardinality $s_1 + \ldots + s_i + s_{i+1}$ in G'_{i+1}. Since M_{i+1} is a maximum matching in G'_{i+1}, its cardinality is at least $s_1 + \ldots s_i + s_{i+1}$. Thus $r_{i+1} = s_{i+1}$.

Hence, M_{i+1} is a greedy matching in G_{i+1}. It is now easy to show that every greedy matching of G_{i+1} is a maximum matching of G'_{i+1}. Let N_{i+1} be any greedy matching of G_{i+1}. By the induction hypothesis and Lemma 2, we know that N_{i+1} is contained in G'_{i+1}. N_{i+1} has cardinality $s_1 + s_2 + \ldots + s_{i+1}$, which is equal to the cardinality of M_{i+1}, which is a maximum matching of G'_{i+1}. Hence, N_{i+1} is also a maximum matching of G'_{i+1}. This completes the proof of the lemma. ∎

The running time of the algorithm

Theorem 1 *A rank-maximal matching can be computed in time $O(\min(n + C, C \cdot \sqrt{n}) \cdot m)$, where C is the maximal rank of an edge in an optimal solution*

Proof: Consider a fixed phase. We first determine the partition of the node set and then reduce the edge sets. This takes time $O(m)$. We next compute M_{i+1} from M_i by augmenting along augmenting paths. Using the algorithm of Hopcroft and Karp [6], this takes time $O(\min(\sqrt{n}, |M_{i+1}| - |M_i| + 1) \cdot m)$. The total number of phases is r and hence the overall running time is $O(\min(n + r, r \cdot \sqrt{n}) \cdot m)$.

We next show how to replace r by C. At the beginning of each phase, say phase i, we first check whether M_{i-1} is already a maximum matching in G'_{ir}, where G'_{ir} denotes the graph consisting of all edges, of all ranks, that are still present at the beginning of phase i. This takes time $O(m)$. If so, we stop. If not, we continue as described above. In this way, only C phases are executed. ∎

3 A Reduction to Weighted Matchings

In this section, we present another algorithm to compute a greedy matching in the graph $G = (\mathcal{A} \cup \mathcal{P}, \mathcal{E})$. This algorithm transforms the greedy matching problem into the maximum weight matching problem. In order to do so, as mentioned in Section 1, one can give each edge of rank i the weight n^{r-i}, where r is the maximum rank of any edge in G. It is then easy to see that a maximum weight matching is a greedy matching.

The best strongly polynomial algorithm for solving the maximum weighted matching (as described in [10]) uses the primal-dual schema and has running time $O(n(m + n \log n))$. Moreover, when the edge weights are integers there are weakly polynomial algorithms [1, 11]

with running time $O(\sqrt{n}m \log(nD))$ where D is the largest edge weight. These algorithms are based on scaling. All the above running times assume that arithmetic operations can be performed in constant time. In our case, since any edge weight may be as large as n^{r-1}, each operation might take time up to r (this assumes that numbers up to n are handled in constant time). Together with the fact that the algorithms use numbers which are bounded by a constant times the maximal edge weight, we get a running time of $rn(m + n \log n)$ and $r\sqrt{n}m \log n$ respectively. The space requirement also increases since for each edge we have to keep a vector of size r for its edge weight. This limits the feasible problem size.

Our algorithm is based on the strongly polynomial primal-dual algorithm. However, as we will show there is no need for shortest path computations and moreover, we have tackled the problem of the arithmetic operations due to the large edge weights. Its running time is $O(Cnm)$, where $C \leq r$ is the maximal rank used in an optimal solution. The space requirement is $O(m)$. We leave it as an open question whether similar techniques as used in [1, 11] can be used in order to get a better running time of $(C\sqrt{n}m)$ using weighted matching algorithms.

Let us give some background. The maximum weighted matching problem can be formulated as a linear programming problem. We associate a variable $x(e)$ with every edge e and constrain it to $0 \leq x(e) \leq 1$. In the dual linear program we use the variable $\pi(v)$ for the variable corresponding to node v. The function π is called the *potential function*. Moreover, let the *reduced cost* of an edge e with endpoints a and b be $\bar{c}(e) = \pi(a) + \pi(b) - w_e$, where w_e is the weight of the edge. An edge is called *tight* if it has reduced cost equal to zero. A tight non-negative potential function proves the optimality of a maximum weight matching. For more details see [10] and [12].

Our algorithm uses scaling and works in phases. Recall that $\mathcal{M}_1$ is the set of matchings in which a maximal number of nodes in A is matched through their highest ranked edge. For $i \geq 2$, $\mathcal{M}_i$ is the subset of $\mathcal{M}_{i-1}$ in which a maximal number of nodes in A is paired through their rank i edges. Note that a greedy matching belongs to all $\mathcal{M}_i$'s. At the end of phase i, our algorithm constructs a matching $M_i \in \mathcal{M}_i$ and a potential function $\pi_i : \mathcal{A} \cup \mathcal{P} \mapsto \mathbb{N}$ proving its optimality. In phase i we use cost n^{i-j} for the edges of rank j.

An overview of the algorithm

All the edges in phase 1 are rank 1 edges. So, M_1 is a maximum cardinality matching, which can be computed by the Hopcroft-Karp algorithm [6] which takes time $O(\sqrt{n}m)$. Moreover, we set the potential π_1 of all nodes in $\mathcal{A}$ and all free nodes in $\mathcal{P}$ to zero and the potential of all matched nodes in $\mathcal{P}$ to one. In this way π_1 proves the optimality of M_1.

Assume now we have completed phase $i-1$. We have a matching $M_{i-1} \in \mathcal{M}_{i-1}$ and a potential function π_{i-1} proving its optimality, i.e., for every edge $e = (a, b) \in \mathcal{E}_{\leq i-1}$ we have $\bar{c}_{i-1}(e) \geq 0$ and with equality for the edges in M_{i-1} and we have $\pi_{i-1}(v) \geq 0$ for all nodes and with equality for free nodes. These are the usual optimality conditions.

We now scale up. We multiply all node potentials by n, add one to the potentials of the nodes in $\mathcal{A}$, for every edge in $\mathcal{E}_{\leq i-1}$ we multiply the weight by n, we add the edges in $\mathcal{E}_i$ and set their weight to one. Call the new potential function π_i.

It is easy to see that by this scaling up, the resulting potential function π_i is non-negative and also that $\bar{c}_i(e) \geq 0$ is true for all edges. We ensure this condition holds for the edges in $\mathcal{E}_i$ by adding one to the potentials of the nodes in $\mathcal{A}$. For the edges in $\mathcal{E}_{\leq i-1}$ it holds trivially. Finally observe that edges in M_{i-1} have reduced cost exactly one.

We now find M_i by any of the standard algorithms, e.g., the one described in [10]. The algorithm starts with the empty matching, and the potential function π_i obtained from the scaling step. The only initially possible violated optimality condition, is that a free node v might have potential $\pi_i(v) > 0$. The algorithm iteratively reduces the number of such free nodes, by either augmenting along a suitable alternating path of tight edges, or by reducing the total potential. In both situations, the other optimality conditions are always preserved. When all remaining free nodes have potential zero, all optimality conditions are satisfied and thus the obtained matching M_i is optimal, and the potential function π_i proves its optimality.

Lemma 5 *M_i has the same number of edges of all ranks $j, j < i$ as M_{i-1}.*

Proof: Because of the steeply decreasing sequence of weights, for any value of $j < i$, if M_i has a smaller number of edges of rank j than M_{i-1}, then the weight of M_i is smaller than the current weight of M_{i-1}, independent of the number of edges of rank $> j$ matched by M_i. This contradicts that M_i is the maximum weight matching. ∎

The following lemma is valuable for our algorithm's running time.

Lemma 6 *Let Π_i be the sum of all π_i potentials. Then*

$$n\Pi_{i-1} = nw(M_{i-1}) \leq w(M_i) \leq \Pi_i \leq n\Pi_{i-1} + n.$$

Proof: The first equality holds since π_{i-1} proves the optimality of M_{i-1}. The second inequality holds since M_i contains the same number of edges of all ranks j, $j < i$ as M_{i-1} and maybe some edges of rank i and the weights of all edges in $\mathcal{E}_{\leq i-1}$ have been multiplied by n. The third inequality holds since π_i is non-negative and covers all edges, and the final inequality follows from the definition of π_i. ∎

We next analyze the running time of a phase.

Lemma 7 *Assuming that all arithmetic operations can be performed in constant time, a phase takes $O(nm)$ running time.*

Proof: The lemma follows from the fact that each augmentation either increases the size of the matching or increases the number of free nodes of potential zero, thus there can be at most $2n$ augmentations. Moreover, each potential change decreases the total potential by at least one, and therefore by Lemma 6 there can be at most n global potential changes. Each such operation takes time $O(m)$. Finally, the scaling part of the phase takes time $O(n + m)$. ∎

Cost of arithmetic operations

Let us now address the question of arithmetic. Node potentials, edge costs, and reduced costs of edges are bounded by n^n and hence the cost of an arithmetic operation might be as large as n (this assumes that numbers up to n are handled in constant time).

We want to argue that edges of large reduced cost can be ignored. The following two lemmas allow us to do so.

Lemma 8 *If the reduced cost of an edge, $\bar{c}(e)$ is more than n at the beginning of a phase, then this edge cannot become tight during the execution of this phase.*

Proof: In each phase there are at most n global potential changes. We may assume that each such change increments and decrements node potentials by one. Thus the reduced cost of an edge can be decreased by at most n in a phase and hence any edge which has reduced cost more than n at the beginning of a phase cannot become tight during the phase. ∎

Lemma 9 *Any edge which is non-tight at the end of a phase, will never become tight later in the execution and can therefore be deleted.*

Proof: Assume that an edge has reduced cost $\bar{c}(e) \geq n+1$ at the beginning of a phase. Then it has reduced cost at least 1 at the end of the phase. At the beginning of the next phase, this edge has reduced cost $\bar{c}'(e) = n * \bar{c}(e) + 1 \geq n+1$. Thus this edge will never become tight later in the execution and can therefore be deleted. ∎

From the above, we can conclude that all old edges will have reduced cost one at the beginning of a phase and also that the reduced costs are bounded by $O(n)$ and hence can be handled in constant time. It only remains to show that we can compute the initial reduced costs of newly added edges in constant time.

Their weight is one and hence their reduced cost will be larger than $n+1$ if either endpoint has potential two or more at the end of the previous phase. So let us assume that we know at the end of a phase, which nodes have potential zero, one or at least two. The former nodes have potential zero or one at the beginning of the next phase, the middle nodes have potential n or $n+1$ and the latter nodes have potential at least $2n$ at the beginning of the next phase and hence at least n at the end of the next phase. Thus we only need to distinguish potential values 0 to $n+1$ and $+\infty$. The ∞ category contains all nodes whose potential is guaranteed to be at least 2 at the end of the phase. Potentials up to $n+1$ are incremented and decremented during a phase. Once a node potential reaches $n+2$, the node is moved to the ∞-category.

Now we present the full scaling algorithm.

The scaling algorithm

We start with M_1, which is a maximum cardinality matching in the graph defined by $\mathcal{E}_1$, the edges of rank 1, and potential π_1 such that $\pi_1(a) = 0\ \forall a \in A$; $\pi_1(b) = 0\ \forall$ free nodes $b \in B$ and $\pi_1(b) = 1\ \forall$ matched nodes $b \in B$. All edges of rank 1 have weight 1.

For $i = 2$ to r do the following steps, and output M_r.

1. If M_{i-1} is a perfect matching, then output M_{i-1} and exit, else define
   $\pi_i(v) = \pi_i(v) \cdot n + 1 \quad \forall v \in A$
   $\pi_i(v) = \pi_i(v) \cdot n \quad \forall v \in B$

2. Add the edges in $\mathcal{E}_i$ to the graph. For every new edge e, if either endpoint of e is in the ∞ category, delete e. For every remaining new edge e, compute the reduced cost of e by setting $w(e) = 1$.

3. Determine M_i by using the primal-dual algorithm for maximum weighted matching in the above graph. With every edge, store its reduced cost. Update the reduced cost whenever the potential of one of its endpoints changes. Whenever a node potential reaches $n+2$, move the node to the ∞ category.

4. Delete all non-tight edges.
   If for any v, $\pi_i(v) \geq 2$, then move v to the ∞ category.

The following theorem summarizes our algorithm's running time.

Theorem 2 *The scaling maximum weighted matching algorithm computes a rank maximal matching in $O(Cnm)$ time, where C is the largest rank used in an optimal solution. It uses space $O(m)$.*

Proof: The running time follows directly from Lemma 7 and the fact that every arithmetic operation can be performed in constant time. The space requirement follows from the fact that we need to store only the reduced costs of the edges, and the node potentials. Both are bounded by $O(n)$, thus use constant space. ∎

4 Conclusion and open problems

We present two algorithms to compute a greedy matching in a bipartite graph where each edge has a rank. Both the algorithms are improvements over the traditional solution of this problem using the Hungarian algorithm. We describe a combinatorial algorithm with running time $O(C\sqrt{n}m)$ and a scaling algorithm with running time $O(Cnm)$, where C is the highest rank used in a greedy matching. Our scaling algorithm uses the strongly polynomial primal-dual algorithm to compute a maximum weight matching. We leave it as an open question whether the weakly polynomial algorithms for maximum weight matchings can be adapted to give better running times for computing a greedy matching.

References

[1] H. N. Gabow and R.E. Tarjan. Faster scaling algorithms for network problems. *SIAM Journal of Computing*, 18:1013–1036, 1989.

[2] D. Gale and L. S. Shapley. College admissions and the stability of marriage. *American Mathematical Monthly*, 69:9–15, 1962.

[3] R. L. Graham, M. Grötschel, and L. Lovasz, editors. *The Handbook of Combinatorics*, chapter 3, Matchings and Extensions, pages 179–232. North Holland, 1995.

[4] D. Gusfield and R. W. Irving. *The Stable Marriage Problem: Structure and Algorithms.* MIT Press, 1989.

[5] M. M. Halldórsson, K. Iwama, S. Miyasaki, and H. Yanagisawa. Improved approximation of the stable marriage problem. to appear in ESA 2003.

[6] J. E. Hopcroft and R. M. Karp. An $n^{5/2}$ algorithm for maximum matching in bipartite graphs. *SIAM Journal on Computing*, 2:225–231, 1973.

[7] R. W. Irving. Matching medical students to pairs of hospitals: a new variation on a well-known theme. In *Proceedings of the Sixth European Symposium on Algorithms*, pages 381–392, 1998.

[8] R. W. Irving. Greedy matchings. Technical Report TR-2003-136, University of Glasgow, April 2003.

[9] R. W. Irving, D. F. Manlove, and S. Scott. Strong stability in the hospitals/residents problem. In *Proceedings of the STACS*, pages 439–450, 2003.

[10] K. Mehlhorn and S. Naher. *LEDA: A Platform for Combinatorial and Geometric Computing*. Cambridge University Press, 1999.

[11] J. B. Orlin and R. K. Ahuja. New scaling algorithms for the assignment and minimum cycle problems. *Mathematical Programming*, 54:41–56, 1992.

[12] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization Algorithms and Complexity*. Prentice Hall, 1982.

[13] A. E. Roth. The evaluation of the labor market for medical interns and residents: a case study in game theory. *Journal of Political Economy*, 92(6), 1984.