Greedy Matchings

Robert W. Irving

Department of Computing Science, University of Glasgow, Glasgow G12 8QQ, UK. rwi@dcs.gla.ac.uk.

Abstract

Suppose that each member of a set of n applicants ranks a subset of a set of m posts in strict order of preference. A *matching* is a set of (post, applicant) 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 where the preferences are on only one side of the market.

A greedy matching can be found by a transformation to the classical problem of *maximum weight bipartite matching*. However an exponentially decreasing sequence of weights must be assigned to the entries in each preference list, and this adversely affects the complexity of the algorithm (and its performance in practice). Here, we describe a more direct augmenting-path based algorithm to find a greedy matching, we analyse its worst-case complexity, and give empirical evidence on its performance relative to a highly tuned implementation of the Hungarian algorithm.

1 Description of the problem

Let $\mathcal{A}$ be a set of n *applicants* and $\mathcal{P}$ a set of m *posts*, and suppose that, associated with each member of $\mathcal{A}$ is a strictly 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 $G = (V, E)$ in which $V = \mathcal{A} \cup \mathcal{P}$ and $(a, p) \in E$ if and only if post p appears in the preference list of applicant a. The *size* of a matching is the number of applicants matched.

The question arises as to how we might compare matchings in this context, and how we might define a notion of optimality, and find efficiently a matching that is optimal in that sense. In the case where preferences are expressed on both sides of the market, a key property of a matching is

stability, and, subject to that property, matchings that are simultaneously optimal for all members on one side of the market can be found efficiently. This is the domain of *stable matching problems*, which have been studied extensively — see, for example, [2] — and variants of the so-called Gale-Shapley algorithm [1] are routinely used to solve a number of large-scale real-life matching problems of this kind [6, 3].

When preferences are expressed on one side only, a number of different forms of optimality can be defined. Such problems arise commonly in practice — for example in the allocation of projects to students, and in the allocation of trainee teachers to probationary posts in at least one national matching scheme. Here we investigate just one such possibility, the notion of what we call a *greedy matching*.

Denote by $\mathcal{M}$ the set of all matchings of $\mathcal{A}$ to $\mathcal{P}$. Define $\mathcal{M}_1$ to be the subset of $\mathcal{M}$ in which the maximum possible number of candidates are matched to their first choice post[1]. For $i = 2, 3, \ldots$, define $\mathcal{M}_i$ to be the subset of $\mathcal{M}_{i-1}$ in which the maximum possible number of candidates are matched to their ith choice post. A matching that belongs to $\mathcal{M}_i$ for all i will be called a *greedy* matching.

Alternatively and equivalently, define the *profile* $\rho(M)$ of a matching M to be the m-tuple $(x_1, \ldots, x_m)$ where, for each i, x_i is the number of applicants who are matched in M with their ith choice post. Define a total order $\prec$ on profiles as follows: $(x_1, \ldots, x_m) \prec (y_1, \ldots y_m)$ if, for some r, $x_i = y_i$ for $1 \leq i < r$ and $x_r < y_r$. Then a greedy matching is one that, among all matchings, has the maximum possible profile under this ordering. As a matter of convenience, we abbreviate a profile $(x_1, \ldots x_m)$ by $(x_1, \ldots, x_d)$ if $x_d > 0$ and $x_i = 0$ for $i = d+1, \ldots, m$.

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

We make the trivial observation that, in spite of the terminology, a simple greedy algorithm, in which we assign the maximum number of candidates to their first choice post, then the maximum number of the remainder to their second choice post, and so on, is by no means guaranteed to lead to a greedy matching. However, a greedy matching can be found by transforming to an instance of the classical maximum weight bipartite matching problem [4, 5]. This involves allocating a suitably steeply decreasing sequence of weights to the positions in each applicant's preference list to ensure that, for any value of i, an applicant who improves from his $(i+1)$th to his ith choice post would change the weight of the matching by more than any number of applicants who move down from their $(i+j)$th choice, for any value of $j \geq 1$. This can be achieved, for example, by assigning a weight of n^{m-i}

[1]This number is clearly the number of different posts that appear as the first choice of at least one applicant.

to each applicant's ith choice, and 0 to a post that does not appear in the applicant's preference list. Note, however, that the use of such large integers as edge weights in the graph may lead to implementation complications, and also results in a $\Omega(m \log n)$ time algorithm for integer operations (such as comparison), giving an additional factor of $m \log n$ in the complexity function of the algorithm. So, for example, in the special case where $m = n$ and every applicant's preference list contains all of the posts, the $O(n^3)$ Hungarian algorithm for the maximum weight bipartite matching problem yields an algorithm for the greedy matching problem that is no better than $O(n^4 \log n)$ in the worst case. If the preference lists are of bounded length, say at most c, then the resulting bipartite graph has at most cn edges, and the weights are bounded by n^c. In this case the Hungarian algorithm can be implemented to run in $O(cn^2 \log n(c + \log n))$ time in the worst case [4].

We note in passing that the cardinality of a greedy matching may differ considerably from that of a maximum cardinality matching. As an extreme case, consider an example with $n = m$ (n odd) in which a maximum cardinality matching is perfect, i.e., has size n, while a greedy matching has size $1 + \lfloor n/2 \rfloor$, giving a ratio tending to 0.5. The preference lists of the n applicants are as follows:

- applicant 1 has a preference list of length 1 containing only post 1
- for $2 \leq i \leq 1 + \lfloor n/2 \rfloor$, applicant i has a preference list of length 2 containing post 1 and post i in that order
- for $2 + \lfloor n/2 \rfloor \leq i \leq n$, applicant i has a preference list of length 2 containing post $i - \lfloor n/2 \rfloor$ and post i in that order.

In a greedy matching, applicants $2 + \lfloor n/2 \rfloor, \ldots, n$ and one of applicants $1, \ldots, 1 + \lfloor n/2 \rfloor$ obtain their first choice post and all remaining applicants are unmatched. However, a maximum cardinality matching assigns candidate 1 to his first choice post and all other candidates to their second choice post.

On the other hand, it is easy to see that a greedy matching can never be of size $< 1/2$ the size of a maximum cardinality matching.

In this paper, we address the problem of finding a direct algorithm for a greedy matching, and whether we can find such an algorithm that compares favourably, in terms of worst-case complexity and/or performance in practice, with solutions obtained by transforming to maximum weight bipartite matching. The algorithm, together with a correctness proof, is given in Section 2. A worst case analysis of the algorithm appears in Section 3. In Section 4 we give some empirical evidence to compare the performance, in practice, of the new algorithm relative to the highly tuned LEDA implementation of the Hungarian algorithm for maximum weight bipartite matching. Finally, Section 5 contains some concluding remarks and open problems.

2 A direct algorithm for the greedy matching problem

We define the *degree* $d(M)$ of a matching M to be the largest value of k such that some applicant is assigned to his kth choice post in M. For a particular instance of the greedy matching problem, suppose that a greedy matching has degree d. (Clearly every greedy matching has the same degree, so the value of d is a property of the instance itself.) Let the profile of a greedy matching M be $(r_1, r_2, \ldots, r_d)$, so that $r_d > 0$, and $r_1 + r_2 \cdots + r_d = g$, where g is the size of M.

A matching M' of size s with degree j and profile $(s_1, \ldots, s_j)$ will be called a *greedy s-matching* if $s_i = r_i$ for $1 \leq i < j$, and $s_j \leq r_j$.

Given an instance of the greedy matching problem we can construct a corresponding weighted bipartite graph $G = (V, E)$ in which $V = \mathcal{A} \cup \mathcal{P}$, $E = \{\{a, p\} : a \in \mathcal{A}, p \in \mathcal{P}$, p is one of a's chosen posts $\}$, and the weight, or *order*, of edge $\{a, p\}$ is the position of p in a's preference list.

Suppose we have the weighted bipartite graph G representing an instance of the greedy matching problem, and let M be a greedy s-matching in G. Then, clearly, M corresponds to a set of edges in G no two of which have a common end-vertex. Any edge of G that is in M will be called a *match edge*, and any edge of G that is not in M will be called a *non-match edge*. A vertex v is *exposed* relative to M if it is not a member of any edge of M. An *alternating path* relative to M is a path consisting alternately of match and non-match edges.

We seek an algorithm that finds a greedy matching by iterating a step that takes a greedy s-matching as input and generates a greedy $(s+1)$-matching as output. As is customary in matching, the algorithm will be based on an appropriate notion of augmenting path – i.e., an alternating path, with suitable properties, that starts and ends at an exposed vertex. The question is, what properties does such an augmenting path have, and how can we set about searching for one?

We begin with a standard type of lemma that justifies the concept of an augmenting path. By *augmenting a matching M* along such a path P, we mean adding to M the odd-numbered edges and removing from M the even-numbered edges, or, equivalently, forming the symmetric difference $M \oplus P$, thus obtaining a matching M' of size one greater than M.

Lemma 2.1 *For a given instance I of the Greedy Matching problem, if we have a greedy s-matching M, then we can obtain from it a greedy $(s+1)$-matching by augmenting along an appropriate augmenting path.*

Proof Let M' be an arbitrary greedy $(s+1)$-matching. Let $G = (V, E)$ be the weighted bipartite graph corresponding to I, and consider the

subgraph $G' = (V', E')$ of G with $V' = V$ and $E' = M \oplus M'$, where $\oplus$ denotes symmetric difference. Let the edges of E' be coloured red or blue according as the corresponding edge in E is in M or in M'. It is immediate that the connected components of G' are paths or even-length cycles with edges that are alternately red and blue.

In any component of G' that is an even-length path or a cycle, the multiset of orders of the red edges must be identical to the multiset of orders of the blue edges, otherwise the greedy property of M or of M' would be contradicted. Hence we can amend M' by replacing all of the blue edges that are in cycles or even-length paths of G' by the red edges that are in those cycles and paths. After this amendment, M' remains a greedy $(s+1)$-matching.

Similarly, if G' has more than one component that is an odd-length path, we can take any two such components that have, between them, the same number of red and blue edges. Again, for the same reason as before, the multisets of orders of red and blue edges must be identical, so we can amend M', retaining its greedy property, to the point where the corresponding graph G' has just a single odd-length path of alternate blue and red edges. This establishes the lemma. □

Suppose that we have a greedy s-matching M_s in G. A *greedy augmenting path* for M_s is an augmenting path that leads, by augmentation, to a greedy $(s+1)$-matching M_{s+1}. There are some severe restrictions on the nature of such a path because of the greedy status of M_s and M_{s+1}.

We shall call an alternating path from an exposed applicant vertex a to some other vertex v in G feasible if it could, in principle, be extended to a greedy augmenting path[2]. In order to be feasible, the sequence of orders of the edges on the path must satisfy a property that can most easily be described by a Boolean valued function, as given in Figure 1. When applied to the order sequence $< x_1, x_2, \ldots, x_r >$, this function must return the value `True`. We call any integer sequence that causes the value `True` to be returned by this function *stack monotonic*.

The sequence of values on the stack (enumerated top to bottom) is called the *signature* of this path from a to v. It is a strictly increasing sequence of positive integers, and is of odd length if v is a post vertex, and of even length if v is an applicant vertex.

Lemma 2.2 *Suppose that $a_1, p_1, a_2, p_2, \ldots$ is an alternating path from an exposed vertex a_1, and that this path can be extended to a greedy augmenting path. Then the corresponding sequence of edge orders is stack monotonic.*

[2]This is not the definition of feasible, merely intuitive justification for the concept; the definition is in terms of stack monotonicity introduced subsequently.

```
Create(S);    -- an empty integer stack
for i in 1...r loop
    if Is-Empty(S) or else x_i < Top(S) then
        Push(S, x_i);
    elsif x_i = Top(S) then
        Pop(S);
    else
        return False;
    end if;
end loop;
return True;
```

Figure 1: Algorithm to test for stack monotonicity

Proof Suppose first that the above function returns the value `False` on encountering a_i, for some i. Then it is not hard to show that replacing the edges $(p_1, a_2), \ldots, (p_{i-1}, a_i)$ by the edges $(a_1, p_1), \ldots, (a_{i-1}, p_{i-1})$ in the current matching will contradict the fact that the current matching is greedy.

Suppose now that the above function returns the value `False` on encountering p_i, for some i. Then it is not hard to show that replacing the edges $(a_1, p_1), \ldots, (a_i, p_i)$ by the edges $(p_1, a_2), \ldots, (p_{i-1}, a_i)$ in the supposed augmented matching will contradict the fact that the augmented matching is greedy. □

Hence we now define a *feasible* alternating path to be an alternating path that starts at an exposed applicant vertex and that has a sequence of edge orders that is stack monotonic.

Lemma 2.3 *Suppose that M is a greedy s-matching of degree k, and that a greedy $(s+1)$-matching has degree $l(\geq k)$. Let P be an alternating path, relative to M, from an exposed applicant vertex a to an exposed post vertex p. Then P is a greedy augmenting path for M if and only if*

(i) the sequence of edge orders in P is stack monotonic, and

(ii) the signature of P is $< l >$.

Proof The 'if' part of the proof is immediate, since augmenting along P changes the multiset of edge orders only by adding one copy of l.

To prove the 'only if' part, the preceding lemma establishes the need for stack monotonicity. To see that the signature of P must be $< l >$, it suffices to observe that the signature characterises the difference between the profile of M and that of the matching M' obtained from M by augmenting along P. To be precise, values in odd-numbered places in the signature represent gains, and values in even-numbered positions represent losses in the profile of M' as compared to the profile of M. The result follows. □

These lemmas provide us with the basis of an algorithm for the greedy matching problem. Suppose we have a greedy s-matching of degree k. We first search for a greedy augmenting path with signature $< k >$, which would lead to a greedy $(s+1)$-matching of degree k. If we fail to find such a path, then, for successive values of i $(= 1, 2, \ldots)$ we search for a greedy augmenting path with signature $< k + i >$, which would lead to a greedy $(s+1)$-matching of degree $k + i$. If we fail to find such a path for any value of i, then the current matching is a greedy matching, and the algorithm terminates. The algorithm is summarised in Figure 2. To justify the loop terminating condition, we note that the size of a matching cannot exceed either the number of applicants or the number of posts, and the degree of a matching cannot exceed the number of posts. (This latter bound could be tightened to the maximum length of any preference list.)

$M := \emptyset$; $--$ the empty matching
$k := 0$;
while $|M| < \min(m, n)$ **and** $k \leq m$ **loop**
    if $\exists$ a greedy augmenting path P relative to M with signature $< k >$ **then**
        augment M along P;
    else
        $k := k + 1$;
    end if;
end loop;

Figure 2: Outline algorithm for a greedy matching

Suppose that we are seeking a greedy augmenting path, relative to a greedy s-matching, with signature $< k >$. Potentially, this search might have to consider each exposed applicant vertex as a starting point. During this search, we need pursue only paths that are stack monotonic and that contain no edge of order greater than k. If we reach an exposed post vertex we have found a greedy augmenting path.

The problem is how to make this search efficient. It appears that we cannot avoid visiting the same vertex repeatedly. It is certainly possible that we reach a vertex by one feasible alternating path that does not extend to a greedy augmenting path, but then we reach it again by a different feasible alternating path that does.

In order to limit the number of times that any vertex is visited during the search, we record, for each post vertex p, the *minimum* signature of any path encountered from an exposed applicant vertex to p. Here, 'minimum' is relative to a total ordering on signatures defined below. We call this the

current *value* of p, denoted $V(p)$. If we subsequently re-visit p, it turns out that we need not pursue the current search path unless its signature is 'less' than the current value $V(p)$, in which case $V(p)$ is updated and the path pursued.

So what do we mean when we say that one signature is *less* than another? Recall that a signature is a strictly increasing sequence of positive integers. This sequence has even or odd length according as the alternating path in question ends at an applicant or a post vertex — we are concerned only with paths to post vertices, so we assume that the signatures are of odd length. Let $\sigma_1 = (x_1, x_2, \ldots, x_p)$ and $\sigma_2 = (y_1, y_2, \ldots, y_q)$ be two such signatures. If $p < q$ and $x_i = y_i$ for $i = 1, \ldots, p$, then we say that $\sigma_1 < \sigma_2$, whereas if $q < p$ and $x_i = y_i$ for $i = 1, \ldots, p$, then we say that $\sigma_2 < \sigma_1$. Otherwise, define r to be the smallest value of i such that $x_i \neq y_i$. If r is odd, then $\sigma_1 < \sigma_2$ if and only if $x_i < y_i$, whereas if r is even, then $\sigma_1 < \sigma_2$ if and only if $x_i > y_i$. It is immediate that $<$ is a total order on the set of possible signatures.

Given any alternating path P relative to a matching M, we define the *difference vector*, or simply the *vector* $v(P)$ of P to be the m-tuple $(x_1, \ldots, x_m)$, where x_i is the difference between the number of match edges of order i and the number of non-match edges of order i in P. A non-zero vector $v(P)$ is *positive*, written $v(P) > 0$, if the first non-zero element in $v(P)$ is positive, and is otherwise negative, written $v(P) < 0$. Note that we allow this definition to apply even to alternating paths that use the same edge more than once — any such edge contributes to the vector according to the number of times that it appears in the path.

Lemma 2.4 *Let C be an alternating cycle relative to a greedy s-matching M. Then $v(C) \geq 0$.*

Proof Suppose that $v(C) < 0$. Then it is immediate that the matching obtained from M by exchanging non-match edges for match edges in C has a greater profile than M, contradicting M's greedy property. □

Lemma 2.5 *If P is a greedy augmenting path relative to a greedy s-matching M, then there is no augmenting path Q with $v(Q) - v(P) < 0$, where $-$ represents element-wise subtraction of the vectors.*

Proof Such a path would contradict the greedy property of the augmenting path P. □

Lemma 2.6 *Let P be a (not necessarily simple) alternating path, relative to a greedy s-matching M, from vertex u to vertex w, and let x be a post vertex and y an applicant vertex such that x precedes y on P. If P' is an alternating path obtained from P by inserting an additional sequence of edges from y to x (none of these additional edges being in P), then $v(P') \geq v(P)$.*

Proof Clearly $v(P') = v(P) + v(C)$, where C is an alternating cycle, and the result follows from Lemma 2.4. □

Lemma 2.7 *Let P be a greedy augmenting path relative to a greedy s-matching M, and let x be a post vertex and y an applicant vertex such that y precedes x on P. If S is the subpath of P from y to x and S' is any alternating path from y to x that contains no edges of P then $v(S') \geq v(S)$.*

Proof If this were not true, then replacing S by S' in P would give an augmenting path P' such that $v(P') < v(P)$, contradicting the greedy status of P, by Lemma 2.5. □

Lemma 2.8 *Let P_1 and P_2 be two feasible alternating paths from exposed vertices to some post vertex p. Then the signature of P_1 is less than the signature of P_2 if and only if $v(P_1) - v(P_2) < 0$.*

Proof Immediate. □

The fact that an alternating path to a post vertex p need be pursued only if its signature is less than that of any previous alternating path to p is a consequence of the following key theorem.

Theorem 2.1 *Let G be the weighted bipartite graph representing an instance of the greedy matching problem, and let $P = a_1p_1a_2p_2 \ldots a_rp_r$ be a greedy augmenting path of order k relative to a greedy s-matching M in G. Then, for each i ($1 \leq i \leq r$), the signature of path $P_i = a_1p_1 \ldots a_ip_i$ is the minimum signature of any alternating path from an exposed applicant vertex to p_i. Furthermore, if P' is any such path to p_i with the same signature then either P' or a prefix of P' can be extended to a greedy augmenting path of order k.*

Proof Denote by σ the signature of path $X = a_1p_1a_2 \ldots a_ip_i$, and suppose there is another feasible alternating path $Y = b_1q_1 \ldots b_jq_j$, with signature $\tau < \sigma$ from an exposed applicant vertex b_1 to $q_j = p_i$.

Consider the path $Z = b_1q_1 \ldots b_jq_ja_{i+1}p_{i+1} \ldots a_rp_r$. This need not be a simple path. There may be one or more pairs (k, l) such that $1 \leq k \leq j$, $i + 1 \leq l \leq r$, with $q_k = p_l$, but $b_k \neq a_l$. Let the pairs of this kind be $(k_1, l_1), \ldots, (k_s, l_s)$, with $k_1 < \cdots < k_s$. There must also be pairs $(k'_1, l'_1), \ldots, (k'_s, l'_s)$, with $k_1 < k'_1 \leq k_2 < k'_2 \leq \cdots \leq k_s < k'_s$, such that $b_{k'_i} = a_{l'_i}$ but $q_{k'_i} \neq p_{l'_i}$ $(1 \leq i \leq s)$. In other words, each subpath of Z of the form $q_{k_i}b_{k_i+1} \ldots b_{k'_i}$ is part of the path P whereas each subpath of Z of the form $b_{k'_i}q_{k'_i}b_{k'_i+1} \ldots q_{k_{i+1}}$ is not part of P.

For each i $(1 \leq i \leq s)$ let Z_i be the path that follows Y from b_1 to q_{k_i} and then follows P from q_{k_i} to p_r. So Z_1 is a simple path and $Z_s = Z$.

Consider how Z_{i+1} differs from Z_i. We split into two cases, depending on whether $q_{k_{i+1}}$ precedes or follows q_{k_i} in path P.
case (i) $q_{k_{i+1}}$ precedes q_{k_i} in path P: Then Z_{i+1} is obtained from Z_i in the manner of Lemma 2.6, and it follows from that lemma that $v(Z_{i+1}) \geq v(Z_i)$.
case (ii) $q_{k_{i+1}}$ follows q_{k_i} in path P: Then Z_{i+1} is obtained from Z_i in the manner of Lemma 2.7, and it follows from that lemma that $v(Z_{i+1}) \geq v(Z_i)$.

Applying this result inductively, we find that $v(Z_1) \leq v(Z)$. Furthermore,

$$\begin{aligned} v(Z) &= v(b_1 \to q_j) + v(p_i \to p_r) \\ &< v(a_1 \to p_i) + v(p_i \to p_r) \\ &= v(P). \end{aligned}$$

Here, we use $x \to y$ to represent the subpath of Y or P, as appropriate, from vertex x to vertex y.

Hence, in view of Lemma 2.5, the augmenting path Z_1 provides a contradiction of the greedy property of the augmenting path P.

For the second part of the theorem, the result is immediate if P' extended by $a_{i+1}p_{i+1} \ldots p_r$ is a simple path, and otherwise the construction used for Z_1 above provides the claimed extension of a prefix of P'. □

A convenient way of organising the search for a greedy augmenting path is in breadth-first order. This ensures that potential alternating paths are considered in increasing order of length. In typical instances, we can expect augmenting paths to be relatively short – this will certainly be true in the early iterations.

A pseudocode version of the breadth-first search (BFS) algorithm for a greedy augmenting path of order k is shown in Figure 3. In this algorithm, a *feasible* neighbour of an applicant vertex a is a post vertex q, not matched to a, such that the order of the edge q is at most equal to the value on top of the current stack (the signature of the current path to a), or at most k if the current stack is empty. It is clear that only such feasible post vertices are of interest since only they can give a feasible alternating path. In fact, a further feasibility criterion that can be used to speed up the algorithm is that, if the edge $\{a, q\}$ has order j then there must be at least one match edge of order j, since a greedy augmenting path that includes edge $\{a, q\}$ must also include at least one such match edge.

Throughout, the predecessor of each post vertex on the best path from an exposed applicant vertex is stored, and these predecessors allow the reconstruction of any augmenting path found.

3 Analysis of the algorithm

Suppose that a greedy matching has size s, degree d, and profile $(x_1, \ldots, x_d)$, and let $s_k = x_1 + \cdots + x_k$ $(1 \leq k \leq d)$. The greedy augmenting path

```
for each post vertex p loop
    V(p) :=< ∞ >; -- greater than any real signature
end loop;
create an empty queue of applicant vertices Q;
for each exposed applicant vertex a loop
    add a to Q;
end loop;
while Q is not empty loop
    remove a from Q;
    if a is matched, say with p then
        S := V(p);
        update(S, (a, p));    -- S is now the signature of the 'best' path to a
    else
        S :=<>; -- empty signature
    end if;
    for each feasible neighbour q of a loop
        -- feasible in terms of the target order k and the signature S
        update(S, (a, q));    -- S is now the signature of the 'current' path to q
        if S < V(q) then    -- this is a signature comparison
            V(q) := S;
            set the predecessor of q to be a;
            if q is exposed then
                set q to be the end of the augmenting path;
                return the augmenting path;
            elsif q's matched applicant is not in Q then
                add q's matched applicant to Q;
            end if;
        end if;
        restore S;    -- S is once again the signature of the current path to a
    end loop;
end loop;
return failure;    -- there is no augmenting path with signature < k > if this point is reached;
```

Figure 3: BFS algorithm for a greedy augmenting path of order k

algorithm will be called $x_k + 1$ times with target order k; x_k of the calls find a greedy augmenting path and the last one fails because the required order at that point is insufficient. In practice, we could avoid all the calls with target order $k = 1$ — instead of starting with the empty matching we could trivially generate an initial greedy x_1-matching.

So to complete the analysis, we need to investigate the worst-case behaviour of the greedy augmenting path procedure. A key question here is how many times the main loop of the augmenting path procedure can be iterated. To provide a bound on this, we establish the following lemma. In stating this lemma, we note that, whenever an applicant vertex b is added to the queue, it is because of some alternating path from an exposed applicant vertex a that is, in a precise sense, *better* than any such alternating path previously encountered.

Lemma 3.1 *For a given applicant vertex b, the ith time that vertex is added to the queue must be as a result of an alternating path from an exposed vertex that contains at least $i - 1$ match edges.*

Proof This is a simple induction argument. The base case ($i = 1$) is immediate. For the induction step, note that we do not add b to the queue if it is already there. When we come to add it the ith time, all the feasible alternating paths with fewer than $(i - 1)$ match edges have already been processed, because of the breadth-first order. □

It follows from this lemma that, if the size of the current greedy matching is at most s_k, then the number of iterations of the main loop of the augmenting path procedure can be at most $n - s_k + s_k(s_k + 1) = n + s_k^2$. This is because, bearing in mind that a vertex is removed from the queue during every loop iteration, each of the unmatched applicant vertices, numbering at least $n - s_k$, is added to the queue at most once, and each of the other vertices, numbering at most s_k, can be added at most $s_k + 1$ times.

Consider the $x_k + 1$ calls of the augmenting path algorithm in which the target order is k. The signature of any vertex is a sequence of length at most k, so signature assignments and comparisons take $O(k)$ time. Also, the inner *for* loop in the algorithm is executed at most k times since applicant vertices cannot have more than k feasible neighbours.

It follows that the total amount of work done in progressing from a greedy s_{k-1}-matching to a greedy s_k-matching is

$$O(k^2(x_k + 1)(n + s_k^2)),$$

and so the overall complexity of our greedy matching algorithm is

$$O(\sum_{k=1}^{d} k^2(x_k + 1)(n + s_k^2)).$$

This is likely to be a significant over-estimate of the true complexity, certainly in the average case, and we conjecture that this is also true even in the worst case, though we have no proof. But it seems unlikely that all of the conditions required for this worst case analysis to be tight can be satisfied simultaneously.

Disappointingly, even in the case where all preference lists are of length bounded by a constant c, this worst-case complexity, when expressed in terms of n and c is no better than $O(c^2n^3)$, and so does not compare favourably with that of the Hungarian algorithm (see Section 1). However, in practice, at least for most preference profiles, we can expect the sequence $< x_k >$ to be very heavily weighted towards small values of k, resulting in an average case that is a substantial improvement on this figure.

4 Empirical evidence

In an attempt to evaluate the practical utility of the new algorithm, we obtained empirical evidence comparing its performance with that of the highly tuned LEDA implementation of the Hungarian algorithm [4]. In all cases we took the number of posts to be the same as the number n of applicants, and we varied the length c of the preference lists from a small constant value up to n. We noted execution time for the two algorithms, averaged over 5 instances of each size. We generated each preference list as a random permutation of a random subset of appropriate size of the set of posts.

Table 1 shows the results of these experiments. Each row of the table represents a particular value of n, and each column the length of the preference lists. The first entry in each cell relates to the new algorithm and the second to the Hungarian algorithm. All times shown are in seconds on a 450 Mhz PC with 128 megabytes of RAM. Times marked * reflect a slowdown in execution time due to swapping, and missing entries represent instances that could not be solved due to memory constraints.

An apparent anomaly in some table entries for the new algorithm is explained by observing that, when c becomes large relative to n, the execution time can vary substantially from instance to instance. This is because it depends on the degree of the matching rather than the value of c and this is quite variable in practice.

It can be seen that the new algorithm is capable of solving many problem instances that are inaccesible to the Hungarian algorithm, namely those with relatively long preference lists, for which the requirement for an exponentially growing sequence of weights causes memory overflow. Also, the new algorithm is faster, except when n is large and c is small (which is, admittedly, likely to be the most important case in any practical application). This results from the observed average case behaviour of the two algorithms,

	c							
n	10	20	50	100	200	500	1000	2000
	0.01	0.01	0.01	0.01				
100	0.05	0.10	0.27	0.66				
	0.01	0.01	0.02	0.02	0.03			
200	0.08	0.17	0.61	1.56	4.73			
	0.05	0.06	0.10	0.12	0.22	0.25		
500	0.18	0.44	1.50	4.14	15.0	342*		
	0.17	0.21	0.31	0.47	0.64	0.90	0.69	
1000	0.36	0.87	3.08	8.67	33.4	-	-	
	0.69	0.78	1.14	1.57	2.35	3.36	3.93	4.69
2000	0.70	1.78	6.35	21.9	350*	-	-	-
	4.15	4.93	5.72	7.05	8.88	15.4	20.9	-
5000	1.90	4.78	18.9	322*	-	-	-	-
	20.7	22.4	26.1	29.1	36.6	54.6	-	-
10000	3.66	9.43	36.4	-	-	-	-	-
	106	117	126	133	157	-	-	-
20000	7.45	19.7	3558*	-	-	-	-	-
	794	852	873	884	-	-	-	-
50000	22.3	1097*	-	-	-	-	-	-

Table 1: Run times for the new algorithm and the Hungarian algorithm

which seems to be little more than linear in n for the Hungarian algorithm, but more like quadratic for the new algorithm. By contrast, the Hungarian algorithm is clearly superlinear in c, on average, whereas the new algorithm is decidedly sublinear in c.

5 Conclusion and Open Problems

In this paper we have introduced the concept of a greedy matching, presented in the context of allocating applicants to posts based on the preferences of the applicants, and we have described a direct algorithm to find such a matching. We have analysed the algorithm, and have presented empirical evidence of its performance in practice relative to a highly tuned version of the Hungarian algorithm for a maximum weight bipartite matching.

The key open question is whether a tighter analysis of this algorithm can be achieved, showing it to be truly competitive in the worst case with the Hungarian algorithm, or alternatively whether a variant, or indeed a different algorithm altogether, can be discovered that will have a more favourable worst-case and/or average/case performance.

There is a whole range of other questions that can be asked about greedy matchings, such as:

Question 5.1 *How different can greedy matchings be?*

Question 5.2 *How can we generate the set of all greedy matchings efficiently?*

Question 5.3 *Does the set of all greedy matchings exhibit any interesting or useful mathematical structure?*

Beyond that, there is the question of direct algorithms for other interesting matchings that can be defined in the same context; for example

- *greedy maximum matchings*, i.e., matchings that satisfy the greedy property but subject to the overall constraint that they have maximum possible cardinality;
- *generous matchings*, i.e., maximum cardinality matchings in which the smallest number of applicants are allocated to their mth choice, and subject to that the smallest number are allocated to their $(m-1)$th choice, and so on.

References

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

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

[3] R.W. Irving. Matching medical students to pairs of hospitals: a new variation on a well-known theme. In *Proceedings of ESA '98: the Sixth European Symposium on Algorithms*, volume 1461 of *Lecture Notes in Computer Science*, pages 381–392. Springer-Verlag, 1998.

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

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

[6] A.E. Roth. The evolution of the labor market for medical interns and residents: a case study in game theory. *Journal of Political Economy*, 92(6):991–1016, 1984.