Popular Matchings: Structure and Algorithms

Eric Mc Dermid and Robert W. Irving*

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

Abstract. An instance of the *popular matching* problem (POP-M) consists of a set of applicants and a set of posts. Each applicant has a preference list that strictly ranks a subset of the posts. A matching M of applicants to posts is *popular* if there is no other matching M' such that more applicants prefer M' to M than prefer M to M'. This paper provides a characterization of the set of popular matchings for an arbitrary POP-M instance in terms of a structure called the *switching graph*, a directed graph computable in linear time from the preference lists. We show that the switching graph can be exploited to yield efficient algorithms for a range of associated problems, including the counting and enumeration of the set of popular matchings and computing popular matchings that satisfy various additional optimality criteria. Our algorithms for computing such optimal popular matchings improve those described in a recent paper by Kavitha and Nasre [5].

1 Introduction and background

An instance of the *popular matching problem* (POP-M) consists of a set A of n_1 *applicants* and a set P of n_2 *posts*. Each applicant $a \in A$ has a strictly ordered preference list of the posts in P that she finds *acceptable*. A *matching* M is a set of applicant-post pairs (a, p) such that p is acceptable to a, and each $a \in A$ and $p \in P$ appears in at most one pair in M. If $(a, p) \in M$ we write $p = M(a)$ and $a = M(p)$. An applicant *prefers* a matching M to a matching M' if (i) a is matched in M and unmatched in M', or (ii) a is matched in both M and M' and prefers $M(a)$ to $M'(a)$. A matching M is *popular* if there is no matching M' such that more applicants prefer M' to M than prefer M to M'. We let $n = n_1 + n_2$, and let m denote the sum of the lengths of the preference lists. It is easy to show that, for a given instance of POP-M, a popular matching need not exist, and if popular matchings do exist they can have different sizes. Abraham et al [1] described an $O(n+m)$ time algorithm for computing a maximum cardinality popular matching, or reporting that none exists.

The results of Abraham et al [1] led to a number of subsequent papers covering variants and extensions of POP-M. See, for example, [2, 3, 6–8, 10]. Kavitha and Nasre [5] recently described algorithms to determine an *optimal* popular matching for various interpretations of optimality; in particular they gave an

* Both authors supported by EPSRC research grant EP/E011993/1.

$O(n^2+m)$ time algorithm to find *mincost*, *rank-maximal* and *fair* popular matchings (see Section 3.5 for definitions of these terms).

Our goal in this paper is to characterize the structure of the set of popular matchings for an instance of POP-M, in terms of the so-called *switching graph*. This structure is exploited to yield efficient algorithms for a range of extensions, such as counting and enumerating popular matchings, generating a popular matching uniformly at random, and finding popular matchings that satisfy additional optimality criteria. In particular, we improve on the algorithm of Kavitha and Nasre by showing how mincost popular matchings can be found in $O(n+m)$ time, and rank-maximal and fair popular matchings in $O(n \log n + m)$ time. Detailed proofs of the various lemmas and theorems stated in the subsequent sections of this paper may be found in the full version [9].

The terminology and notation is as in the previous literature on popular matchings (for example, [1, 7]). For convenience, a unique *last-resort post*, denoted by $l(a)$, is created for each applicant a, and placed last on a's preference list. As a consequence, in any popular matching, every applicant is matched, although some may be matched to their last-resort. Let $f(a)$ denote the first-ranked post on a's preference list; any post that is ranked first by at least one applicant is called an *f-post*. Let $s(a)$ denote the first non-f-post on a's preference list. (Note that $s(a)$ must exist, for $l(a)$ is always a candidate for $s(a)$). Any such post is called an *s-post*. By definition, the sets of f-posts and s-posts are disjoint.

The following fundamental result, proved in [1], completely characterizes popular matchings, and is key in establishing the structural results that follow.

Theorem 1. *(Abraham et al [1]) A matching M for an instance of POP-M is popular if and only if (i) every f-post is matched in M, and (ii) for each applicant a, $M(a) \in \{f(a), s(a)\}$.*

In light of Theorem 1, given a POP-M instance I we define the *reduced instance* of I to be the instance obtained by removing from each applicant a's preference list every post except $f(a)$ and $s(a)$. It is immediate that the reduced instance of I can be derived from I in $O(n+m)$ time. For an instance I of POP-M, let M be a popular matching, and let a be an applicant. Denote by $O_M(a)$ the post on a's reduced preference list to which a is not assigned in M.

2 The structure of popular matchings – the switching graph

Given a popular matching M for an instance I of POP-M, the *switching graph* G_M of M is a directed graph with a vertex for each post, and a directed edge (p_i, p_j) for each applicant a, where $p_i = M(a)$ and $p_j = O_M(a)$. A vertex v is called an *f-post vertex* (respectively *s-post vertex*) if the vertex it represents is an f-post (respectively s-post). We refer to posts and vertices of G_M interchangeably, and likewise to applicants and edges of G_M. An illustrative example of a switching graph for a POP-M instance is given in the full version of this

paper [9]. The example also illustrates each of the forthcoming ideas regarding switching graphs.

A similar graph was defined by Mahdian [6, Lemma 2] to investigate the existence of popular matchings in random instances of POP-M. Note that the switching graph is uniquely determined by a particular popular matching M, but different popular matchings for the same instance yield different switching graphs. The following easily proved lemma gives some simple properties of switching graphs.

Lemma 1. *Let M be a popular matching for an instance I of POP-M, and let G_M be the switching graph of M. Then*
(i) Each vertex in G_M has outdegree at most 1.
(ii) The sink vertices of G_M are those vertices corresponding to posts that are unmatched in M, and are all s-post vertices.
(iii) Each component of G_M contains either a single sink vertex or a single cycle.

A component of a switching graph G_M is called a *cycle component* or a *tree component* according as it contains a cycle or a sink. Each cycle in G_M is called a *switching cycle*. If T is a tree component in G_M with sink p, and if q is another s-post vertex in T, the (unique) path from q to p is called a *switching path*. It is immediate that the cycle components and tree components of G_M can be identified, say using depth-first search, in linear time.

Let C be a switching cycle of G_M. To *apply* C to M is to assign each applicant a in C to $O_M(a)$, leaving all other applicants assigned as in M. We denote by $M \cdot C$ the matching so obtained. Similarly, let P be a switching path of G_M. To *apply* P to M is to assign each applicant a in P to $O_M(a)$, leaving all other applicants assigned as in M. We denote by $M \cdot P$ the matching so obtained.

Theorem 2. *Let M be a popular matching for an instance I of POP-M, and let G_M be the switching graph of M.*
(i) If C is a switching cycle in G_M then $M \cdot C$ is a popular matching for I.
(ii) If P is a switching path in G_M then $M \cdot P$ is a popular matching for I.

Theorem 2 shows that, given a popular matching M for an instance I of POP-M, and the switching graph of M, we can potentially find other popular matchings. Our next step is to establish that this is essentially the only way to find other popular matchings. More precisely, we show that if M' is an arbitrary popular matching for I, then M' can be obtained from M by applying a sequence of switching cycles and switching paths, at most one per component of G_M. First we state a simple technical lemma, the proof of which is an easy consequence of the definition of the switching graph.

Lemma 2. *Let M be a popular matching for an instance I of POP-M, let G_M be the switching graph of M, and let M' be an arbitrary popular matching for I. If the edge representing applicant a in G_M connects the vertex p to the vertex q, then*
(i) a is assigned to p in M;
(ii) if $M'(a) \neq M(a)$ then a is assigned to q in M'.

Lemmas 3 and 4 consider switching cycles and switching paths respectively.

Lemma 3. *Let M be a popular matching for an instance I of POP-M, let T be a cycle component with cycle C in G_M, and let M' be an arbitrary popular matching for I.*
(i) Either every applicant a in C has $M'(a) = M(a)$, or every such a has $M'(a) = O_M(a)$.
(ii) Every applicant a in T that is not in C has $M'(a) = M(a)$.

Lemma 4. *Let M be a popular matching for an instance I of POP-M, let T be a tree component in G_M, and let M' be an arbitrary popular matching for I. Then either every applicant a in T has $M'(a) = M(a)$, or there is a switching path P in T such that every applicant a in P has $M'(a) = O_M(a)$ and every applicant a in T that is not in P has $M'(a) = M(a)$.*

In terms of the application of switching paths or cycles, separate components of a switching graph behave independently, as captured in the following lemma.

Lemma 5. *Let T and T' be components of a switching graph G_M for a popular matching M, and let Q be either the switching cycle (if T is a cycle component) or a switching path (if T is a tree component) in T. Then, T' is a component in the switching graph $G_{M \cdot Q}$.*

We can now characterize fully the relationship between any two popular matchings for an instance of POP-M.

Theorem 3. *Let M and M' be two popular matchings for an instance I of POP-M. Then M' may be obtained from M by successively applying the switching cycle in each of a subset of the cycle components of G_M together with one switching path in each of a subset of the tree components of G_M.*

An immediate corollary of this theorem is a characterization of the set of popular matchings for a POP-M instance.

Corollary 1. *Let I be a POP-M instance, and let M be an arbitrary popular matching for I with switching graph G_M. Let the tree components of G_M be $X_1, \ldots, X_k$, and the cycle components of G_M be $Y_1, \ldots, Y_l$. Then, the set of popular matchings for I consists of exactly those matchings obtained by applying at most one switching path in X_i for each i $(1 \leq i \leq k)$ and by either applying or not applying the switching cycle in Y_i for each i $(1 \leq i \leq l)$.*

3 Algorithms that exploit the structure

Each of the algorithms in this section begins in the same way – by constructing the reduced instance, finding an arbitrary popular matching M (if one exists), building G_M, and identifying its components using, say, depth-first search. All of this can be achieved in $O(n + m)$ time. This sequence of steps is referred to as the *pre-processing phase*.

3.1 Counting popular matchings

A tree component having q s-posts has exactly $q-1$ switching paths. For a tree component X_i, denote by $\mathcal{S}(X_i)$ the number of s-posts in X_i. The following theorem is an immediate consequence of Corollary 1.

Theorem 4. *Let I be a POP-M instance, and let M be an arbitrary popular matching for I with switching graph G_M. Let the tree components of G_M be $X_1, \ldots, X_k$, and the cycle components of G_M be $Y_1, \ldots, Y_l$. Then, the number of popular matchings for I is $2^l * \prod_{i=1}^{k} \mathcal{S}(X_i)$.*

It follows that an algorithm for counting the number of popular matchings need only carry out the pre-processing phase, counting the number of cycle components and the number of s-posts in each tree component, and all of this can be achieved in linear time.

3.2 Random popular matchings

Corollary 1 facilitates the generation of a popular matching for a POP-M instance, uniformly at random, in linear time. The pre-processing phase identifies a popular matching M and the components of G_M. For each cycle component the unique switching cycle is applied or not according to a random bit. For each tree component T, a (possibly empty) switching path is applied according to a random value r in $0, 1, \ldots, q-1$ where q is the number of s-post vertices in T. The algorithm returns the popular matching obtained by applying this choice of switching cycles and switching paths.

3.3 Enumerating popular matchings

An algorithm for enumerating the popular matchings begins with the preprocessing phase, during which a popular matching M and the components of G_M are identified. It is then straightforward to enumerate popular matchings by applying or not applying the switching cycle in each cycle component, and applying in turn each switching path (including the empty path) in each tree component. The pre-processing phase occupies $O(n+m)$ time, and the delay in generating each matching is linear in the size of the switching graph, namely $O(n)$.

3.4 Popular pairs

A *popular pair* for an instance I of POP-M, is an applicant-post pair (a_i, p_j) such that there exists a popular matching M with $(a_i, p_j) \in M$.

Lemma 6. *Let M be a popular matching for an instance I of POP-M. Then, (a_i, p_j) is a popular pair if and only if (i) (a_i, p_j) is in M, or (ii) a_i is an incoming edge to p_j in G_M, and a_i and p_j are in a switching cycle or switching path in G_M.*

It follows from Lemma 6 that the popular pairs can be found in linear time by executing the pre-processing phase followed by a simple traversal of each component of the switching graph

3.5 Optimal popular matchings

Kavitha and Nasre [5] recently studied the following problem: suppose we wish to compute a matching that is not only popular, but is also optimal with respect to some additional well-defined criterion. They defined a natural optimality criterion and described an augmenting path-based algorithm for computing an optimal popular matching. In this section we will describe faster algorithms that exploit the switching graph of the instance to find an optimal popular matching with respect to certain optimality criteria.

For a POP-M instance with n_1 applicants and n_2 posts, we define the *profile* $\rho(M)$ of M to be the (n_2+1)-tuple $(x_1, \ldots, x_{n_2+1})$ where, for each i $(1 \leq i \leq n_2+1)$, x_i is the number of applicants who are matched in M with their i^{th}-choice post. The last-resort post is always considered to be the $(n_2+1)^{th}$-choice post.

Total orders $\succ_R$ and $\prec_F$ on profiles are defined as follows. Suppose that $\rho = (x_1, \ldots, x_k)$ and $\rho' = (y_1, \ldots, y_k)$. Then

- $\rho \succ_R \rho'$ if, for some j, $x_i = y_i$ for $1 \leq i < j$ and $x_j > y_j$;
- $\rho \prec_F \rho'$ if, for some j, $x_i = y_i$ for $j < i \leq n_2$ and $x_j < y_j$.

A *rank-maximal* popular matching [4] is one whose profile is maximal with respect to $\succ_R$. A *fair* popular matching is one whose profile is minimal with respect to $\prec_F$. (Note that, since the number of (n_2+1)th choices is minimised, a fair popular matching is inevitably a maximum cardinality popular matching.) Finally, a *mincost* popular matching is a maximum cardinality popular matching for which $\Sigma i x_i$ is minimum.

If a *weight* $w(a_i, p_j)$ is defined for each applicant-post pair with p_j acceptable to a_i, then the *weight* $w(M)$ of a popular matching M is $\sum_{(a_i,p_j)\in M} w(a_i, p_j)$. We call a popular matching *optimal* if it is of maximum or minimum weight depending on the context.

With suitable choices of weights, it may be verified that rank-maximal, fair and mincost popular matchings are all examples of optimal popular matchings:

- mincost: assign weight n_2^2 to each pair involving a last resort post, a weight of k to each other pair involving a k^{th} choice, and find a minimum weight popular matching.
- rank-maximal: assign weight $(n_2)^{n_2-k+1}$ to each pair involving a kth choice, and find a maximum weight popular matching.
- fair : assign weight $(n_2)^{k-1}$ to each pair involving a kth choice, and find a minimum weight popular matching.

Kavitha and Nasre [5] described an $O(n^2+m)$-time algorithm for finding mincost, rank-maximal and fair popular matchings. In what follows, we give an $O(n+m)$-time algorithm for finding a mincost popular matching and $O(n \log n + m)$-time algorithms for finding rank-maximal and fair popular matchings.

We see from the above that very large weights may be assigned to the applicant-post pairs, so we cannot assume that weights can be compared or

added in O(1) time. We assume that the time for comparison or addition of such values is $O(f(n))$ for some function f.

Given an instance of POP-M and a particular allocation of weights, let M be a popular matching, and M_{opt} an optimal popular matching. By Theorem 3, M_{opt} can be obtained from M by applying a choice of at most one switching cycle or switching path per component of G_M. The key is to decide exactly which switching cycles and paths need be applied. In the following, for simplicity of presentation, we assume that "optimal" means "maximum". Analogous results hold in the "minimum" case.

If T is a cycle component of G_M, an *orientation* of T is either the set of pairs $\{(a, M(a)) : a \in T\}$, or the set $\{(a, M \cdot C(a)) : a \in T\}$, where C is the switching cycle in T. Likewise, if T is a tree component of G_M, an *orientation* of T is either the set of pairs $\{(a, M(a)) : a \in T\}$, or the set $\{(a, M \cdot P(a)) : a \in T\}$, for some switching path P in T. The weight of an orientation is the sum of the weights of the pairs in it, and an orientation of a component is *optimal* if its weight is at least as great as that of any other orientation.

Lemma 7. *If M is an arbitrary popular matching, T is a component of G_M, and M_{opt} is an optimal popular matching, then the set of pairs $\{(a, M_{opt}(a)) : a \in T\}$ is an optimal orientation of T.*

In light of Lemma 7, an algorithm for computing an optimal popular matching can be constructed as follows. For each cycle component T with switching cycle C, an optimal orientation can be found by comparing $\sum_{a \in C} w(a, M(a))$ with $\sum_{a \in C} w(a, M \cdot C(a))$, which is easily achieved in $O(f(n)|T|)$ time. In the case of a tree component T, a depth-first traversal of T can be carried out, starting from the sink, and traversing edges in reverse direction. For an s-post vertex v, the weight of the orientation of T resulting from the application of P_v can easily be found in $O(f(n))$ time from the weight of the orientation resulting from application of P_u, where u is the nearest s-post ancestor of v in the depth-first spanning tree. So the weight of each orientation can be computed in $O(f(n))$ time, and hence an optimal orientation of each tree component T can be found in $O(f(n)|T|)$ time.

Theorem 5. *There is an algorithm to compute an optimal popular matching in $O(m + nf(n))$ time, where n is the number of posts, m is the sum of the lengths of the original preference lists, and $f(n)$ is the maximum time needed for a single comparison of two given weights.*

We use the uniform cost model, which assumes that an arithmetic or comparison operation on numbers of size $O(n)$ has cost $O(1)$. In the case of a mincost popular matching, all weights are $O(n)$, so that, we can take $f(n) = 1$. However, for rank maximal or fair matchings, we can only assume that the weights are $O(n^n)$, so that $f(n) = O(n)$. Hence we have the following corollary.

Corollary 2. *(i) A mincost popular matching can be found in linear time.*
(ii) A rank-maximal and a fair popular matching can be found in $O(m + n^2)$ time.

3.6 Improving the running time

To improve the complexity of our algorithms for a rank-maximal and a fair popular matching, we discard the weights and work directly with matching profiles. This improved algorithm is described for rank-maximal popular matchings; the changes that need to be made to compute a fair popular matching are similar.

Let Z be a tree component of the switching graph with sink z, let $u \neq z$ be an s-post vertex in Z, and let $v \neq u$ be a vertex such that there is a path $P(u, v)$ in Z from u to v. Any such path $P(u, v)$ is the initial part of the switching path $P(u, z)$ starting at u.

The concept of *profile change* $C(u, v)$ along a path $P(u, v)$ quantifies the effect on the profile of applying the switching path from u, but only as far as v – we call this a *partial switching path*. More precisely, $C(u, v)$ is the sequence of ordered pairs $\langle (i_1, j_1), \ldots, (i_r, j_r) \rangle$, where $j_1 < \ldots < j_r$, $i_k \neq 0$ for all k, and, for each k, there is a net change of i_k in the number of applicants assigned to their j_kth choice post when $P(u, v)$ is applied.

We define a total order $\succ$ on profile changes (to reflect rank-maximality) in the following way. If $x = \langle (p_1, q_1), \ldots, (p_k, q_k) \rangle$ and $y = \langle (r_1, s_1), \ldots, (r_l, s_l) \rangle$ are profile changes ($x \neq y$), and j is the maximum index for which $(p_j, q_j) = (r_j, s_j)$, we write $x \succ y$ if and only if

(i) $k > l$, $j = l$, and $p_{j+1} > 0$; or
(ii) $k < l$, $j = k$ and $r_{j+1} < 0$; or
(iii) $j < \min(k, l)$, $q_{j+1} < s_{j+1}$ and $p_{j+1} > 0$; or
(iv) $j < \min(k, l)$, $q_{j+1} > s_{j+1}$ and $r_{j+1} < 0$; or
(v) $j < \min(k, l)$, $q_{j+1} = s_{j+1}$ and $p_{j+1} > r_{j+1}$.

A profile change $\langle (i_1, j_1), \ldots, (i_r, j_r) \rangle$ is *improving* (with respect to $\succ_R$) if $i_1 > 0$. So an improving profile change leads to a better profile with respect to $\succ_R$. Moreover, if x and y are profile changes with $x \succ y$, and if applying x and y to the same profile ρ yields profiles ρ_x and ρ_y respectively, then $\rho_x \succ_R \rho_y$.

As a next step, we define the following arithmetic operation, which captures the notion of adding an ordered pair to a profile change. For a profile change $C = \langle (i_1, j_1), \ldots, (i_r, j_r) \rangle$ and ordered pair (i, j) ($i \neq 0, j > 0$), define $C + (i, j)$ as follows:

$$j = j_k,\ i_k + i \neq 0 \Rightarrow C + (i, j) = \langle (i_1, j_1), \ldots, (i_k + i, j_k), \ldots, (i_r, j_r) \rangle.$$
$$j = j_k,\ i_k + i = 0 \Rightarrow$$
$$C + (i, j) = \langle (i_1, j_1), \ldots, (i_{k-1}, j_{k-1}), (i_{k+1}, j_{k+1}) \ldots, (i_r, j_r) \rangle.$$
$$j_{k-1} < j < j_k \Rightarrow C + (i, j) = \langle (i_1, j_1), \ldots, (i_{k-1}, j_{k-1}), (i, j), (i_k, j_k) \ldots, (i_r, j_r) \rangle.$$

The algorithm computes an optimal orientation of a tree-component Z by a post-order traversal, viewing Z as rooted at the sink. During this traversal, *processing* a vertex v means determining the best improving profile change C_v obtainable by applying a partial switching path that ends at v, together with the starting vertex u_v of a path $P(u_v, v)$ corresponding to C_v. If no path ending at v has an improving profile change then C_v is null and u_v is undefined.

```
Traverse(v) {
    if v is a leaf
        return null;
    else
        best = null;   start = null;
        for (each child w of v that is not an f-post leaf)
            (C_w, u_w) = Traverse(w);
            C = C_w + (1, j_w)) + (−1, l_w);          (1)
            if (C ≻ best)                              (2)
                best = C_w;   start = u_w;
        return (best, start);  }
```

Fig. 1. The postorder traversal of a tree component

For a leaf vertex v, C_v is trivially null. For a branch node v, C_v and u_v are computed using the best improving profile change C_w for each child w of v in the tree (excluding any such w that is an f-post leaf, since no switching path can begin in such a subtree of v). Let w be a child of v, and let a be the applicant represented by the edge (w, v) of Z. Let posts v and w be the j_wth and l_wth choices, respectively, of applicant a, so that if a were to be re-assigned from post w to post v the profile would gain a j_w^{th} choice and lose an l_w^{th} choice. It follows at once that C_v is determined by the formula

$$C_v = \max\{(C_w + (1, j_w)) + (-1, l_w)\}$$

where the maximum is with respect to $\succ$, and is taken over all children w of v. A pseudocode version of the algorithm appears in Figure 1.

On termination of the traversal, we have determined C_z, the best improving profile change, if any, of a switching path in Z, together with the starting point of such a path. Application of this switching path yields an optimum orientation of Z, or, in case *null* is returned, we know that Z is already optimally oriented.

From the pseudocode in Figure 1, we see that the complexity of the algorithm is determined by the total number of operations involved in steps (1) and (2).

To deal with (1), we represent a profile change by a balanced binary tree B whose nodes contain the pairs (i, j), ordered by the second member. The $+$ operation on profile changes involves amendment, insertion, or deletion of a node in B, which can be accomplished in time logarithmic in the size of B. Since the number of pairs in a profile change cannot exceed the number of edges in Z, this is $O(\log t)$, and since step (1) is executed at most t times, the total number of operations carried out by step (1), summed over all iterations, is $O(t \log t)$.

As far as (2) is concerned, we first note that two profile changes, involving c_1 and c_2 pairs, with $c_1 < c_2$, can be compared in $O(c_1)$ time. So the cost of a comparison is linear in the size of each of the balanced trees involved. Once a profile change is the 'loser' in such a comparison, the balanced tree representing it is never used again. Hence the cost of all such comparisons is linear in s, the sum of the sizes of all of the balanced trees constructed by the algorithm.

But each tree node originates from one or more edges in Z, and each edge in Z contributes to at most one node in one tree. So s is bounded by the number of edges in Z, and hence the total number of operations in step (2), summed over all iterations, is $O(t)$.

It follows that the postorder traversal of a tree component Z with t edges can be completed in $O(t \log t)$ time, and once the optimal switching path is found it can be applied in $O(t)$ time. Hence, since the total number of edges in all tree components is $O(n)$, this process can be applied to all tree components in $O(n \log n)$ time.

Finally, we observe that the optimal orientation of each cycle component can be computed efficiently. For a cycle component Y with switching cycle C, we need only check if the profile change obtained by applying C is an improving profile change, and, if so, C is applied. Hence, the optimal orientation of a cycle component Y with y edges can be computed in $O(y)$ time. This process can therefore be applied to each cycle component in $O(n)$ time. Since the preprocessing phase of the algorithm requires $O(n + m)$ time, we conclude that a rank-maximal popular matching, and by similar means a fair popular matching, can be found in $O(n \log n + m)$ time.

References

1. Abraham D.J. , Irving R.W., Kavitha T., Mehlhorn K.: Popular matchings, SIAM Journal on Computing, 37, 1030–1045, (2007)
2. Abraham D.J. , Kavitha T.: Dynamic matching markets and voting paths. In: 10th Scandinavian Workshop on Algorithm Theory, LNCS, vol. 4059, pp. 65–76, (2006)
3. Huang C-C., Kavitha T., Michail D., Nasre M.: Bounded unpopularity matchings. In: 12th Scandinavian Workshop on Algorithm Theory, LNCS, vol. 5124, pp. 127-137, (2008)
4. Irving R.W., Kavitha T., Mehlhorn K., Michail D., Paluch K.: Rank-maximal matchings, ACM Transactions on Algorithms, 2, 602-610, (2006)
5. Kavitha T., Nasre M.: Optimal Popular Matchings. In: Proceedings of MATCH-UP: Matching Under Preferences - Algorithms and Complexity, satellite workshop of *ICALP 2008*.
6. Mahdian M.: Random popular matchings. In: 7th ACM Conference on Electronic Commerce, pp. 238-242, (2006)
7. Manlove D.F., Sng C.T.S.: Popular Matchings in the capacitated house allocation problem, In: 14th Annual European Symposium on Algorithms, LNCS, vol. 4168, pp. 492–503, Springer, (2006)
8. McCutchen R.: The least-unpopularity-factor and least-unpopularity-margin criteria for matching problems with one-sided preferences, In: 8th Latin American Symposium on Theoretical Informatics, LNCS vol. 4957, pp. 593-604, (2008)
9. E. McDermid and R. Irving, Popular Matchings: Structure and Algorithms, Technical Report TR-2008-292, Department of Computing Science, University of Glasgow, November 2008.
10. J. Mestre: Weighted popular matchings. In: 33rd International Colloquium on Automata, Languages and Programming LNCS, vol. 4051, pp 715–726, (2006)