Popular Matchings: Structure and Algorithms

Eric McDermid and Robert W. Irving

Department of Computing Science
University of Glasgow
Glasgow G12 8QQ
UK

Technical Report
TR-2008-292
September 2008

Popular Matchings: Structure and Algorithms

Eric McDermid* and Robert W. Irving*
Department of Computing Science, University of Glasgow, Glasgow G12 8QQ, UK.
mcdermid@dcs.gla.ac.uk, rwi@dcs.gla.ac.uk

October 31, 2008

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'. Abraham et al [1] described a linear time algorithm to determine whether a popular matching exists for a given instance of POP-M, and if so to find a largest such matching. A number of variants and extensions of POP-M have recently been studied. 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, generation of a popular matching uniformly at random, finding all applicant-post pairs that can occur in a popular matching, 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 [6].

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 $a \in A$ supplies a strictly ordered preference list of the posts in P that he 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' with the property 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.

The concept of a popular matching was first introduced by Gardenfors [3], but in the setting of *stable* matchings, in which the posts in P also provide preference lists over the applicants in A. In the case of POP-M, it is easy to show that, for a given instance, a popular matching need not exist, and if popular matchings do exist they can have different sizes.

*Supported by EPSRC research grant EP/E011993/1.

Abraham et al [1] described an $O(n+m)$ time algorithm for computing a largest possible popular matching, or reporting that no popular matching exists.

The results of Abraham et al [1] led to a number of subsequent papers covering variants and extensions of POP-M. Manlove and Sng [8] studied the *capacitated popular matching problem*, C-POP-M, in which each post $p \in P$ has a capacity $c(p)$, the maximum number of applicants that can be assigned to it in any matching. Manlove and Sng gave a $O(\sqrt{C}n_1 + m)$ time algorithm for C-POP-M, where C is the sum of the capacities of the posts. (Some authors, including Manlove and Sng, refer to applicants and posts as agents and houses, leading to the terminology of *house allocation*). Mestre [10] gave a linear time algorithm for a version of the problem in which each applicant has an associated weight; the goal is to find a matching M with the property that there is no other matching M' preferred by a weighted majority of agents. Mahdian [7] showed that popular matchings exist with high probability for random instances of POP-M if the number of posts exceeds the number of applicants by a small constant multiplicative factor. Abraham and Kavitha [2] studied a dynamic version of POP-M allowing for applicants and posts to enter and leave the instance, and for applicants to arbitrarily change their preference lists. They showed the existence of a so-called 2-step *voting path* to compute a new popular matching after every such change, assuming that a popular matching exists. McCutchen [9] focused on instances of POP-M for which no popular matching exists, defined two notions of 'near popularity', and proved that for each of these it is NP-hard to find a matching that is as near to popular as possible. Huang et al [4] built upon the work of McCutchen with a study of approximation algorithms in the context of near popularity. Kavitha and Nasre [6] described algorithms to determine an *optimal* popular matching for various interpretations of optimality; in particular they gave an $O(n^2 + m)$ time algorithm to find *minimum cost*, *rank-maximal* and *fair* popular matchings (see Section 3.5 for definitions of these terms).

1.1 The contribution of this paper

Our goal in this paper is to characterize the structure of the set of popular matchings for an instance of POP-M. This characterization is in terms of the so-called *switching graph*, that enables an arbitrary popular matching M' to be obtained from a given popular matching M in linear time. This structure is exploited to enable the design of efficient algorithms for a range of extensions of the basic problem, such as counting and enumerating popular matchings, generating a popular matching uniformly at random, and finding popular matchings that satisfy various additional optimality criteria. In particular, we improve on the algorithm of Kavitha and Nasre by showing how minimum-cost popular matchings can be found in $O(n+m)$ time, and rank-maximal and fair popular matchings in $O(n \log n + m)$ time.

1.2 Preliminaries

The terminology and notation is as in the previous literature on popular matchings (for example, [1, 8]). 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, i.e., in time that is linear in the size of the input. Henceforth, unless explicitly stated, it is assumed that a given instance of POP-M is a reduced instance. For a (reduced) 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. Note that since I is a reduced instance, $O_M(a)$ is well defined. So, if a is matched to $f(a)$ in M, then $O_M(a) = s(a)$, whereas if a is matched to $s(a)$ in M, then $O_M(a) = f(a)$.

Example As an illustration of the reduced instance of a POP-M instance, consider the full POP-M instance I of Figure 2 (Appendix) with applicants $a_1 \ldots a_{16}$ and posts $p_1 \ldots p_{18}$. The reduced instance of I is shown in Figure 3.

2 The structure of popular matchings – the switching graph

The key concept that underlies the characterization of the structure of popular matchings is 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). Each vertex (respectively edge) is labelled with the post (respectively applicant) that it represents. In fact, we refer to posts and vertices of G_M interchangeably, and likewise to applicants and edges of G_M. An applicant (respectively post) is said to be *in* a component, or path, or cycle of G_M if the edge (respectively vertex) representing it is in that component, path or cycle.

A very similar graph was defined by Mahdian [7, Lemma 2]. However, Mahdian used this structure solely to investigate the existence of popular matchings in random instances of POP-M.

Some simple properties of switching graphs are spelt out in the following lemma.

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.

Proof (i) A vertex v in G_M has an outgoing edge for each applicant who is matched in M to the post represented by v, and there can be at most one such applicant because M is a matching.

(ii) A vertex has no outgoing edge if and only if it represents an unmatched post, and by Theorem 1 (i) any such post is an s-post.
(iii) This is an easy consequence of (i). $\square$

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.* So each cycle component of G_M has a unique switching cycle, but each tree component may have zero or more switching paths; to be precise it has one switching path for each s-post vertex that it contains, other than the sink vertex. It is immediate that the cycle components and tree components of G_M can be identified, say using depth-first search, in linear time.

Example Figure 4 (Appendix) provides an illustrative example of the switching graph of a popular matching M in the POP-M instance described in Figures 2 and 3. The switching graph of this instance contains one cycle component and two tree components.

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

Similarly, let P be a switching path of G_M. To *apply* P to M is to assign each applicant a in P to $O_M(a)$, while leaving all other applicants assigned as in M. We denote by $M \cdot P$ the matching obtained by applying the switching path P to M. Note that, if p is the sink vertex in G_M and the path P begins at vertex q, then in $M \cdot P$, the post p is matched but the post q is unmatched (whereas in M, q is matched and p is unmatched). In general, if we apply a switching cycle or switching path that contains the edge representing applicant a, and this edge connects post q to post p, then applicant a is switched from post q to post p as a result.

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. However, all switching graphs for an instance of POP-M have the same number of vertices (one for each post), and the same number of edges (one for each applicant).

The significance of switching paths and switching cycles begins to emerge in the following theorem.

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.

Proof (i) Let $M' = M \cdot C$. By Theorem 1, it is sufficient to argue that (a) every f-post is matched in M' and (b) for each applicant a, $M'(a) \in \{f(a), s(a)\}$. It is clear, from the cyclic nature of the reassignments that take place on applying C, that each post that is matched in M is also matched in M'. Hence all f-posts are matched in M', and condition (a) is established. Furthermore, each applicant $a_i \notin C$ is assigned to the same post in M' as in M, and each applicant $a_i \in C$ is assigned to $O_M(a_i)$ in M', which is clearly either $f(a_i)$ or $s(a_i)$, establishing (b).
(ii) Condition (b) follows by a similar argument to that of (i), since every applicant a is still assigned to either $f(a)$ or $s(a)$ in $M \cdot P$. Also, the only post that is "vacated" by applying P is the s-post corresponding to the initial vertex of P. Each f-post in P is filled by a different

applicant, and all f-posts not in P are filled by the same applicant as in M, so that condition (a) is satisfied. $\square$

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 the switching graph G_M of 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 applicant a has $M'(a) = O_M(a)$.
(ii) Every applicant a in T that is not in C has $M'(a) = M(a)$.

Proof (i) Let $a_{i_0}, \ldots, a_{i_{r-1}}$ be the sequence of applicants in C, and suppose that $M'(a_{i_j}) \neq M(a_{i_j})$ for some a_{i_j} in C. Then, by Lemma 2, a_{i_j} must be assigned in M' to $O_M(a_{i_j}) = M(a_{i_{j+1}})$ (where $j+1$ is taken mod r). It follows that $a_{i_{j+1}}$ must also be assigned to a different post in M' as compared to M, and that this post must be $O_M(a_{i_{j+1}}) = M(a_{i_{j+2}})$ (where $j+2$ is taken mod r). Inductively, this implies that every applicant in C is assigned different posts in M and M' if any one of them is.
(ii) Suppose, for a contradiction, that an applicant a who is in T but not in C has $M'(a) \neq M(a)$. Let the sequence of distinct edges on the path in T that begins with edge a be $(a =) a_{j_1}, \ldots, a_{j_t}, \ldots, a_{j_s}$ where a_{j_t} is the last edge in this path that is not in the cycle C. Then, by an argument similar to that in (i) above, we must have $M'(a_{j_t}) = M(a_{j_{t+1}})$. But, by the same reasoning, we must have $M'(a_{j_s}) = M(a_{j_{t+1}})$, since the edge $a_{j_{t+1}}$ follows the edge a_{j_s} in the cycle. This implies that a particular post, namely $M(a_{j_{t+1}})$, has two applicants, a_{j_t} and a_{j_s}, assigned to it in M', a contradiction. $\square$

Lemma 4 *Let M be a popular matching for an instance I of POP-M, let T be a tree component in the switching graph G_M of 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 $M'(a) = M(a)$.*

Proof Suppose that $M'(a) \neq M(a)$ for some applicant a in T. By an argument similar to that of part (i) of Lemma 3, the same must be true of every applicant on the path from a to the sink vertex of G_M. Suppose that two applicants in T whose edges have a common end point, say p, are both matched to different posts in M' as compared to M. Then, by Lemma 2, both would have to be assigned in M' to p, a contradiction. Hence the applicants in T who are assigned different posts in M and M' form a path ending at the sink vertex. Moreover, this path must begin at an s-post vertex, otherwise the f-post at the start of the path would be unfilled in M', contradicting Theorem 1, so the path is a switching path. □

Suppose that M is a popular matching for an instance I of POP-M, and that T and T' are distinct components of the switching graph G_M of M. If we apply the switching cycle in T (if T is a cycle component) or a switching path in T (if T is a tree component) to obtain a different popular matching, then the assignments of the applicants in T' are unaffected. Hence, the component T' is present in the switching graph corresponding to the new matching. Intuitively, this means that the application of switching cycles and paths are independent processes when in different components of the switching graph. This notion of independence is 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.*

Proof We describe a procedure for obtaining M' from M in a way that will establish the claim. By Lemma 5, we can describe this procedure in terms of its separate effect on each component of the switching graph.

For each cycle component T of G_M, we know by Lemma 3 that either the applicants a in T all have $M(a) = M'(a)$, or those applicants in the unique cycle of T have $M'(a) = O_M(a)$. In the former case, we leave T unchanged, and in the latter case, we apply the switching cycle in T, so that every applicant a in T becomes matched to $M'(a)$.

For each tree component T' of G_M, we know by Lemma 4 that either every applicant a in T' has $M(a) = M'(a)$, or there is a single switching path P in T' such that every applicant a_j in P has $M'(a_j) = O_M(a_j)$, and all applicants a_k in T' but not in P must have $M(a_k) = M'(a_k)$. Hence, by applying P, every applicant a in T' is matched to $M'(a)$. Thus we obtain M' from M by successively applying at most one switching cycle per cycle component of G_M, and at most one switching path per tree component of G_M. Moreover, the order in which these switching cycles and paths are applied is arbitrary. □

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 ($1 \leq i \leq l$).*

The Appendix contains an example of a POP-M instance, its reduced instance, the switching graph of a particular popular matching M, and an indication of how the application of switching paths and cycles leads to different popular matchings with different, but closely related, switching graphs.

3 Algorithms that exploit the structure

In this section we show how the characterization of the structure of the set of popular matchings for an instance of POP-M allows the construction of efficient algorithms to solve a number of extensions of the basic problem, namely to compute the number of popular matchings, to generate a popular matching uniformly at random, to enumerate the set of all popular matchings, to find all applicant-post pairs that can occur in a popular matching, and to find popular matchings that are optimal in one of a number of natural ways.

Each of these algorithms begins in the same way – by constructing the reduced instance, finding an arbitrary popular matching M (if one exists), building the switching graph G_M, and identifying the cycle components and tree components of this graph using, say, depth-first search. As discussed previously, all of this can be achieved in $O(n + m)$ time, where n is the number of applicants and posts and m is the sum of the lengths of the original preference lists, in other words in time that is linear in the input size. This sequence of steps is referred to as the *pre-processing phase.*

3.1 Counting popular matchings

Recall that 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 $2^l * \prod_{i=1}^{k} \mathcal{S}(X_i)$.*

Thus, in light of Theorem 4, it is easy to see that an algorithm for counting the number of popular matchings for an instance I of POP-M first carries out the pre-processing phase, during which the number l of cycle components and $\mathcal{S}(X_i)$ for each tree component X_i in G_M are determined. Once these values are known, the algorithm then returns the product $2^l * \prod \mathcal{S}(X_i)$.

Theorem 5 *The number of popular matchings for an arbitrary instance of POP-M can be computed in linear time.*

3.2 Random popular matchings

Let I be an arbitrary POP-M instance, and let $\mathcal{M}$ denote the set of popular matchings for I. Corollary 1 facilitates the generation of a popular matching from $\mathcal{M}$ uniformly at random in linear time. The procedure again begins with the pre-processing phase, during which the cycle components $Y_1, \ldots, Y_l$ and the tree components $X_1, \ldots, X_k$ of G_M are identified. Next, for each cycle component Y_i a bit b is generated. The unique switching cycle in Y_i is applied if and only if $b = 1$. Likewise, for each tree component X_i, the s-posts other than the sink are numbered $1, 2, \ldots, q-1$ where $q = \mathcal{S}(X_i)$, and a value r is chosen uniformly at random from the set $\{0, 1, \ldots, q-1\}$. If $r = 0$, no switching path from this component is applied, otherwise, the switching path beginning at the s-post numbered r in X_i is applied. The algorithm returns the popular matching obtained by applying this choice of switching cycles and switching paths.

Theorem 6 *Let I be an instance of POP-M, and let $\mathcal{M}$ denote the set of popular matchings for I. There is an algorithm to generate a popular matching from $\mathcal{M}$ uniformly at random in linear time.*

Proof It is immediate that any one of the popular matchings for the instance is equally likely to be returned by the algorithm. To establish the complexity, it suffices to observe that the pre-processing phase is linear, and the total time spent applying the switching paths and cycles is also linear. $\square$

3.3 Enumerating popular matchings

An algorithm for enumerating the set of popular matchings can be obtained by first computing an arbitrary popular matching M along with the switching graph G_M and then generating all possible popular matchings by applying switching paths and switching cycles as described in Corollary 1.

The algorithm begins with the preprocessing phase. During this phase, for each tree component X_i, $\mathcal{S}(X_i)$ is computed, and the s-posts of this component other than the sink are numbered $1, \ldots, q-1$, where $q = \mathcal{S}(X_i)$. Let $j = l + k$. Next, a vector $V = (v_1, \ldots v_j)$ is defined, where $v_i \in \{0, 1\}$ $(1 \leq i \leq l)$ and $v_{l+i} \in \{0, 1, \ldots, \mathcal{S}(X_i) - 1\}$ $(1 \leq i \leq k)$.

At this point the matching M is output, and V is initialized to be $(0, 0, \ldots, 0)$. The algorithm then loops through all possible values of the vector V. At each iteration, the switching cycle in Y_i is applied to M if and only if $v_i = 1$ $(1 \leq i \leq l)$. For each k $(l < k \leq j)$, if $v_k \neq 0$, the switching path beginning with the s-post numbered v_k in component X_k is applied to M. Otherwise, if $v_k = 0$, no switching path in X_k is applied. The popular matching so generated is then output and control passes to the next loop iteration.

Theorem 7 *Let I be a POP-M instance, and let $\mathcal{M}$ denote the set of popular matchings for I. There is an algorithm that enumerates $\mathcal{M}$ in $O(n + m + n|\mathcal{M}|)$ time.*

Proof The pre-processing phase occupies $O(n + m)$ time. The generation of each popular matching requires the identification and application of a set of switching paths and switching cycles. Within each component this can be done in time linear in the size of the component, so overall in time linear in the size of the switching graph, namely $O(n)$. $\square$

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 $(a_i, p_j) \in M$. We show that the popular pairs can be determined in linear time. The following lemma is the key.

Lemma 6 *Let M be a popular matching for an instance I of POP-M, and let G_M be the switching graph of 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.*

Proof The proof of sufficiency is easy, for if (a_i, p_j) is in M, it is by definition a popular pair. If instead, a_i is an incoming edge to p_j in G_M and a_i is in a switching cycle or path in G_M, we know by Theorem 2 that applying this switching cycle or path matches a_i and p_j in a popular matching.

On the other hand, suppose that $(a_i, p_j) \notin M$. If a_i is not an incoming edge to p_j, then $O_M(a_i) \neq p_j$, implying p_j is not on a_i's reduced preference list, so a_i can never be matched to p_j in a popular matching. Suppose that a_i is an incoming edge to p_j in G_M, but a_i is not in a switching cycle or path in G_M. If we suppose that (a_i, p_j) is in a popular matching M', then, by Theorem 3, there is a sequence of switching cycles and paths that can be applied to transform M to M'. But clearly for a_i to become matched to p_j, a_i must be in one of these switching cycles or paths, giving a contradiction. $\square$

Theorem 8 *There is a linear time algorithm to generate all of the popular pairs for a POP-M instance.*

Proof After the pre-processing phase, all pairs in the resulting popular matching (if any) are output. The switching graph G_M is then traversed to find all applicants a_i in a switching path or switching cycle, and for each such a_i outputs the pair (a_i, p_j), where p_j is the end vertex of edge a_i. Lemma 6 guarantees that the correct set of pairs has been generated. The pre-processing phase and traversal of the switching graph can both be accomplished in linear time. $\square$

3.5 Optimal popular matchings

Kavitha and Nasre [6] 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. We first define two particular optimality criteria, in terms of the so called *profile* of the matching, that have been studied in the literature [1, 6].

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. An applicant who is matched to her last-resort post is considered to be matched to her $(n_2 + 1)^{th}$-choice post, regardless of the length of her preference lists.

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 is a popular matching whose profile is maximal with respect to $\succ_R$. A *fair* popular matching is a popular matching 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.

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.

Some examples of optimal popular matchings include, but are not limited to:

- *Maximum cardinality* popular matchings: assign a weight of 0 to each pair involving a last resort post, and a weight of 1 to all other pairs, and find a maximum weight popular matching. (A linear time algorithm to find a maximum cardinality popular matching was already given by Abraham et al [1].)
- *Minimum cost maximum cardinality* popular matchings: assign a large weight, say n^2, to each pair involving a last resort post, and a weight of k to each pair (a_i, p_j) such that p_j is a_i's k^{th} choice in the original instance, and find a minimum weight popular matching.
- *Rank-maximal* popular matchings: assign a weight of 0 to each pair involving a last resort post, and a weight of n^{n-k+1} to each pair (a_i, p_j) where p_j is kth choice of a_i, and find a maximum weight popular matching.
- *Fair* popular matchings : assign a weight of n^{n-k+1} to each pair (a_i, p_j) where p_j is the $(n-k)^{th}$ choice of a_i, and find a minimum weight maximum cardinality popular matching.

Kavitha and Nasre [6] described an $O(n^2 + m)$-time algorithm for finding minimum cost, fair, and rank-maximal popular matchings. In what follows, we give an $O(n + m)$-time algorithm for finding minimum cost maximum cardinality popular matchings and $O(n \log n + m)$-time algorithms for finding fair and rank-maximal popular matchings.

From the above description of fair and rank-maximal popular matchings, it is apparent that we may wish to assign very large weights to the applicant-post pairs, so we cannot assume that weights can be compared or added in O(1) time. We assume that these weights occupy $O(f(n))$ space for some function f, so that this is also the time for comparison or addition of such values.

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 (maximum or minimum weight, as appropriate). 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 the switching graph G_M. The algorithm for computing M_{opt} will compute an arbitrary popular matching M, and make an appropriate choice of switching paths and switching cycles to apply in order to obtain an optimal popular matching. The next step is to show how 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. Intuitively, an optimal orientation of a component T of G_M assigns the applicants in T so that T contributes its optimal weight to the popular matching.

The following lemma establishes the relationship between M, M_{opt}, and optimal orientations of components of G_M.

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.*

Proof Recall that M_{opt} may be generated from M by applying at most one switching cycle in each cycle component of G_M and at most one switching path in each tree component of G_M. Suppose that the set $\{(a, M_{opt}(a)) : a \in T\}$ is not an optimal orientation of T. Then if we generate a matching M' from M in exactly the same way as M_{opt}, except that we deal with the component T according to an optimal orientation, matching M' will have a weight greater than M_{opt}, a contradiction. □

In light of Lemma 7, an algorithm for computing an optimal popular matching can be constructed as follows. The algorithm begins, as always, with the pre-processing phase. The next step is to find an optimal orientation for each component in G_M. An optimal orientation of each cycle component T with switching cycle C can be found by comparing $\sum_{a \in C} w(a, M(a))$ with $\sum_{a \in C} w(a, M \cdot C(a))$. This is easily done in $O(f(n)|T|)$ time, and the outcome tells us whether or not the switching cycle C should be applied to give an optimal popular matching.

In the case of a tree component T, we would like to find an optimal orientation also in $O(f(n)|T|)$ time. This cannot necessarily be achieved by independent evaluation of each switching path in T. Instead, 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, let P_v be the switching path beginning at v. To find the weight of the orientation of T resulting from the application of P_v, suppose that the switching path starting at v has a and b as its first two edges. This evaluation involves subtracting the weight $w(a, v)$ from, and adding the weight $w(b, u)$ to, 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. By this means we can determine an optimal orientation of each tree component T in $O(f(n)|T|)$ time.

These considerations establish the main theorem of this section.

Theorem 9 *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.*

In the case of a maximum cardinality popular matching, or a minimum cost maximum cardinality popular matching, all weights are bounded by n, so that we can take $f(n) = 1$, and we have the following corollary.

Corollary 2 *A maximum cardinality and a minimum cost maximum cardinality popular matching can be found in linear time.*

In the case of rank maximal or fair matchings, we can only assume $f(n) = O(n)$.

Corollary 3 *A rank-maximal popular matching and a fair popular matching can be found in $O(m + n^2)$ time.*

Improving the running time

When computing a rank-maximal or fair popular matching, the complexity of our algorithm is dominated by the time required to compute an optimal orientation of a component of G_M. To improve the complexity for these specific problems, we discard the weights and work directly with matching profiles. This enables us to compute the optimal orientation of a tree component in $O(t \log t)$ time, where t is the number of edges in the given tree component. For simplicity of presentation, this improved algorithm is described in terms of computing rank-maximal popular matchings, then we indicate the changes that need to be made to compute a fair popular matching.

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.* (It is a genuine switching path if and only if $v = z$). Note that, unless $v = z$, applying such a partial switching path does not yield a matching, since two applicants would be matched to the post represented by v. More precisely, $C(u, v)$ is the sequence of ordered pairs $< (i_1, j_1), \ldots, (i_r, j_r) >$, where $j_1 < j_2 < \ldots < j_r$, $i_k \neq 0$ 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 the partial switching path $P(u, v)$ is applied.

Example As an illustration of the notion of profile change, consider the example of Figure 5. Applying the path $P(l_{16}, p_{11})$ causes a_{15} to move from her nineteenth (last resort) choice to her first choice, a_{14} from her first to her second choice, and a_{12} from her fifth to her first choice, so the resulting profile change is $< (1, 1), (1, 2), (-1, 5), (-1, 19) >$.

We define a total order $\succ$ on profile changes (to reflect rank-maximality) in the following way. If $x = < (p_1, q_1), (p_2, q_2), \ldots, (p_k, q_k) >$ and $y = < (r_1, s_1), (r_2, s_2), \ldots, (r_l, s_l) >$ 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

- $k > l$, $j = l$, and $p_{j+1} > 0$; or
- $k < l$, $j = k$ and $r_{j+1} < 0$; or
- $j < \min(k, l)$, $q_{j+1} < s_{j+1}$ and $p_{j+1} > 0$; or

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

An *improving profile change* (with respect to $\succ_R$) is a profile change $< (i_1,j_1),\dots,(i_r,j_r) >$ with $i_1 > 0$. 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 $C = < (i_1,j_1),\dots,(i_r,j_r) >$ and ordered pair (i,j) $(i \neq 0, j > 0)$, define $C + (i,j)$ as follows:

- If $j = j_k$ and $i_k + i \neq 0$, then
  $C + (i,j) = < (i_1,j_1),\dots,(i_k+i,j_k),\dots,(i_r,j_r) >$.
- If $j = j_k$ and $i_k + i = 0$, then
  $C + (i,j) =< (i_1,j_1),\dots,(i_{k-1},j_{k-1}),(i_{k+1},j_{k+1})\dots,(i_r,j_r) >$.
- If $j_{k-1} < j < j_k$, then
  $C + (i,j) =< (i_1,j_1),\dots,(i_{k-1},j_{k-1}),(i,j),(i_k,j_k)\dots,(i_r,j_r) >$.

The algorithm computes an optimal orientation of a tree-component Z by means of a post-order traversal, viewing Z as a tree 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.

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 step (1), we represent a profile change by a balanced binary tree B whose nodes contain the pairs (i,j), ordered by the second member of the pair. The + operation

```
/* Traverse($v$) returns the optimum profile change $C_v$ and corresponding
   starting vertex $u_v$ of a partial switching path ending at vertex $v$ */
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 (($C \succ$ best)){                       (2)
                best = $C_w$;
                start = $u_w$;
            }
        }
        return (best, start);
    }
}
```

Figure 1: The postorder traversal of a tree component

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 step (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 node in a balanced tree originates from one or more edges in Z, and each edge in Z contributes to at most one node in one balanced 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, otherwise, Y is already optimally oriented. 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. Bearing in mind that the pre-processing phase of the algorithm requires $O(n + m)$ time, we conclude that a rank-maximal popular matching can be found in $O(n \log n + m)$ time.

The algorithm can be amended to find a fair popular matching by making appropriate changes to the definition of the order relation on profile changes, as follows.

We define a total order $\prec$ on profile changes (to reflect fairness) in the following way. If $x = <(p_1, q_1), (p_2, q_2), \ldots, (p_k, q_k)>$ and $y = <(r_1, s_1), (r_2, s_2), \ldots, (r_l, s_l)>$ are profile changes ($x \neq y$), we write $x \prec y$ if and only if

- $q_k > s_k$ and $p_k < 0$; or
- $q_k < s_l$ and $r_l > 0$; or
- j is the maximum value such that $(p_{k-j}, q_{k-j}) = (r_{l-j}, s_{l-j})$, and
  - $q_{k-j-1} > s_{l-j-1}$ and $p_{k-j-1} < 0$; or
  - $q_{k-j-1} < s_{l-j-1}$ and $r_{l-j-1} > 0$; or
  - $q_{k-j-1} = s_{l-j-1}$ and $p_{k-j-1} < r_{l-j-1}$.

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

It is now straightforward to verify that an amended version of the postorder traversal algorithm that uses $\prec$ rather than $\succ$ to compare profile changes will determine a switching path in a tree component of the switching graph that is optimal with respect to fairness. All other aspects of the algorithm, and its analysis, are identical to the rank-maximal case. It follows that a fair popular matching can be computed in $O(n \log n + m)$ time.

4 Summary and open problems

This paper has characterized the structure of the set of popular matchings for a POP-M instance in terms of the so-called switching graph. This characterization leads to efficient algorithms for a range of extensions of the basic problem.

We have assumed throughout that the applicants' preference lists are strictly ordered. Abraham et al [1] considered also the case where the preference lists may contain ties, and gave a $O(\sqrt{n}m)$ time algorithm to determine a popular matching in that case. It is an open question to provide a neat characterization of the structure for this more general problem, and to exploit any such structure to give an analogous set of efficient algorithms. Note that such algorithms would necessarily subsume algorithms for arbitrary bipartite graphs (the case in which every applicant's list is a single tie), therefore, this is likely to be considerably more involved than in the no-ties case.

References

[1] D.J. Abraham, R.W. Irving, T. Kavitha and K. Mehlhorn, Popular matchings, *SIAM Journal on Computing* vol. 37, pp. 1030–1045, 2007. (Preliminary version in *Proc. of SODA 2005, the 16th ACM/SIAM Symposium on Discrete Algorithms*, pp. 424–432, 2005.)

[2] D.J. Abraham, and T. Kavitha, Dynamic matching markets and voting paths, in *Proceedings of SWAT 2006, the 10th Scandinavian Workshop on Algorithm Theory*, vol. 4059 of Lecture Notes in Computer Science, pp. 65–76, 2006.

[3] P. Gardenfors, Match making: assignments based on bilateral preferences. *Behavioral Sciences*, vol. 20, pp. 166-173, 1975.

[4] C-C. Huang, T. Kavitha, D. Michail and M. Nasre, Bounded unpopularity matchings, in *Proceedings of SWAT 2008, the 12th Scandinavian Workshop on Algorithm Theory*, vol. 5124 of Lecture Notes in Computer Science, pp. 127-137, 2008.

[5] R.W. Irving, T. Kavitha, K. Mehlhorn, D. Michail and K. Paluch, Rank-maximal matchings, *ACM Transactions on Algorithms* vol. 2, pp. 602-610, 2006. (Preliminary version in *Proc. of SODA 2004, the 15th ACM/SIAM Symposium on Discrete Algorithms*, pp. 68–75, 2004.)

[6] T. Kavitha and M. Nasre, Optimal Popular Matchings, in *Proceedings of MATCH-UP: Matching Under Preferences - Algorithms and Complexity*, satellite workshop of *ICALP 2008.*

[7] M. Mahdian, Random popular matchings, in *Proceedings of EC 2006: the 7th ACM Conference on Electronic Commerce*, pp. 238-242, 2006.

[8] D.F. Manlove, C.T.S. Sng, Popular Matchings in the capacitated house allocation problem, in *Proceedings of ESA 2006, the 14th Annual European Symposium on Algorithms*, vol. 4168 of Lecture Notes in Computer Science, pp. 492–503, Springer, 2006.

[9] R. McCutchen, The least-unpopularity-factor and least-unpopularity-margin criteria for matching problems with one-sided preferences, in *Proc. of the 8th Latin American Symposium on Theoretical Informatics (LATIN'08)*, pp 593-604, 2008.

[10] J. Mestre, Weighted popular matchings, in *Proceedings of ICALP 2006, the 33rd International Colloquium on Automata, Languages and Programming*, vol. 4051 of Lecture Notes in Computer Science, pp 715–726, 2006.

Appendix

To illustrate the notion of the switching graph and the implications of applying switching paths and cycles, we provide a detailed example. Figure 2 shows the full preference lists (without last resorts) for a POP-M instance I. The instance consists of 16 applicants $a_1, a_2, \dots, a_{16}$ and 18 posts $p_1, p_2, \dots, p_{18}$. The set of f-posts of the instance is $\{p_1, p_4, p_6, p_{10}, p_{11}, p_{15}, p_{17}\}$ and, after the inclusion of the last-resort posts, the set of s-posts is $\{p_2, p_3, p_7, p_8, p_9, p_{12}, p_{13}, p_{14}, l_{16}, p_{18}\}$. Note that post l_{16} is a_{15}'s last resort post. This is the only last resort post which is also an s-post for this instance.

Figure 3 shows the reduced instance of I, obtained by removing all posts from an applicant a's preference list except for $f(a)$ and $s(a)$. This instance admits several popular matchings; one such popular matching M is denoted by underlining.

The switching graph G_M for M is given in Figure 4. The graph consists of three components, one cycle component and two tree components containing posts p_{18} and p_9 as sinks. respectively. The cycle component has a switching cycle containing posts p_1, p_2, p_4, p_3. The switching graph has a total of 4 switching paths, all of which are contained in the larger tree component. These switching paths can easily be identified by recalling that the path to the sink from any other s-post vertex is a switching path. Hence, the paths to p_9 starting at p_{12}, p_{14}, p_{13}, and l_{16} are the switching paths for this instance. The other tree component has no switching path; it represents a fixed pair – applicant a_{16} must be matched to post p_{17} (and post p_{18} is unfilled) in every popular matching.

If the switching cycle is applied, then a_1 becomes matched to p_2, a_2 to p_4, a_3 to p_3, and a_4 to p_1, giving a second popular matching M'. All other applicants are matched to the same posts in M' as in M. Figure 5 shows the change in the switching graph when the switching cycle is applied; the direction of each arc in the cycle is reversed while all other arcs in G_M are unchanged.

If, instead of applying the switching cycle, we apply a switching path, say the path beginning at post p_{13}, applicant a_{12} becomes matched to p_{11}, and applicant a_{10} to post p_9 (which was previously unoccupied in M), giving a third popular matching M''. All other applicants remain matched to the same posts in M'' as in M. Figure 6 shows the change in the switching graph resulting from the application of this switching path; the direction of each arc on the path from p_{13} to p_9 is reversed, so that p_{13} becomes the new sink vertex in this tree component.

As an illustration of Theorem 4, this problem instance has a total of ten popular matchings, resulting from the five possible orientations of the larger tree component and the two possible orientations of the cycle component.

a_1 :	p_1	p_4	p_{10}	p_{11}	p_2	p_6	p_8		
a_2 :	p_4	p_6	p_{11}	p_{17}	p_2	p_5	p_{12}	p_{13}	p_{10}
a_3 :	p_4	p_1	p_3	p_{15}	p_8				
a_4 :	p_1	p_{11}	p_6	p_3	p_{15}	p_3			
a_5 :	p_5	p_1	p_{11}	p_4	p_2	p_6	p_{14}		
a_6 :	p_6	p_{10}	p_{11}	p_3	p_1	p_6			
a_7 :	p_6	p_7							
a_8 :	p_6	p_5	p_8						
a_9 :	p_{10}	p_{11}	p_4	p_9	p_2	p_1	p_{18}		
a_{10} :	p_{11}	p_9	p_7	p_1	p_7	p_{12}			
a_{11} :	p_{10}	p_4	p_{17}	p_6	p_{12}	p_7	p_{13}	p_{10}	
a_{12} :	p_{11}	p_5	p_6	p_{15}	p_{13}	p_1	p_9	p_{18}	
a_{13} :	p_{11}	p_{15}	p_4	p_{14}	p_3				
a_{14} :	p_{15}	p_{13}	p_1	p_4	p_9	p_8			
a_{15} :	p_{15}	p_6							
a_{16} :	p_{17}	p_{15}	p_5	p_4	p_{18}	p_8	p_9	p_{13}	

Figure 2: A popular matching instance I

a_1 :	$\underline{p_1}$	p_2
a_2 :	p_4	$\underline{p_2}$
a_3 :	$\underline{p_4}$	p_3
a_4 :	p_1	$\underline{p_3}$
a_5 :	$\underline{p_5}$	p_2
a_6 :	$\underline{p_6}$	p_3
a_7 :	p_6	$\underline{p_7}$
a_8 :	p_6	$\underline{p_8}$
a_9 :	$\underline{p_{10}}$	p_9
a_{10} :	$\underline{p_{11}}$	p_9
a_{11} :	p_{10}	$\underline{p_{12}}$
a_{12} :	p_{11}	$\underline{p_{13}}$
a_{13} :	p_{11}	$\underline{p_{14}}$
a_{14} :	$\underline{p_{15}}$	p_{13}
a_{15} :	p_{15}	$\underline{l_{16}}$
a_{16} :	$\underline{p_{17}}$	p_{18}

Figure 3: The reduced instance of I with popular matching M denoted by underlining

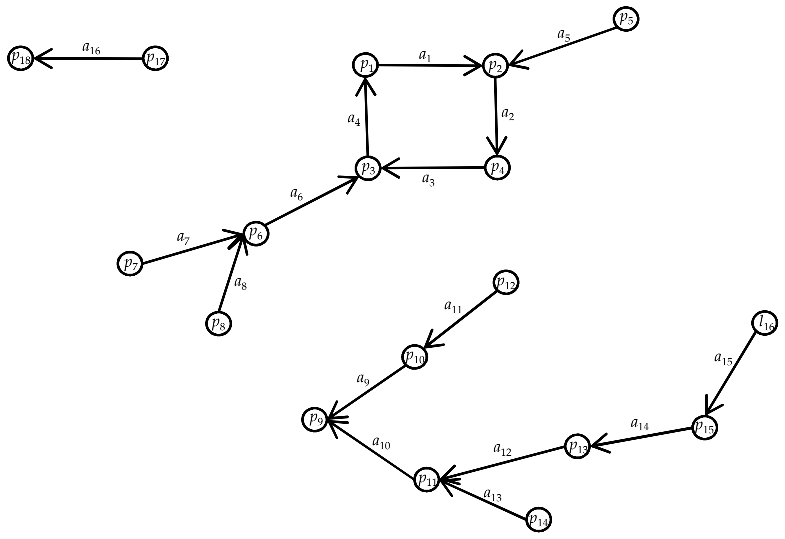

Figure 4: The switching graph G_M for popular matching M

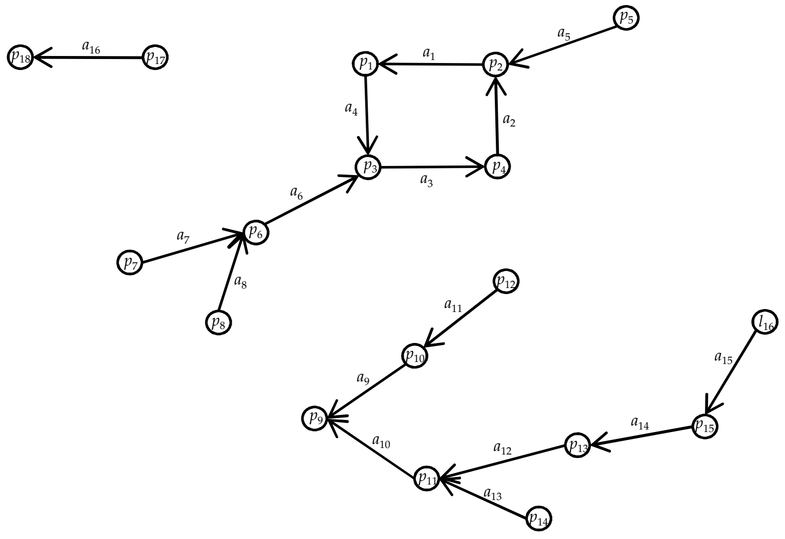

Figure 5: The switching graph for the popular matching M' obtained by applying the switching cycle in G_M

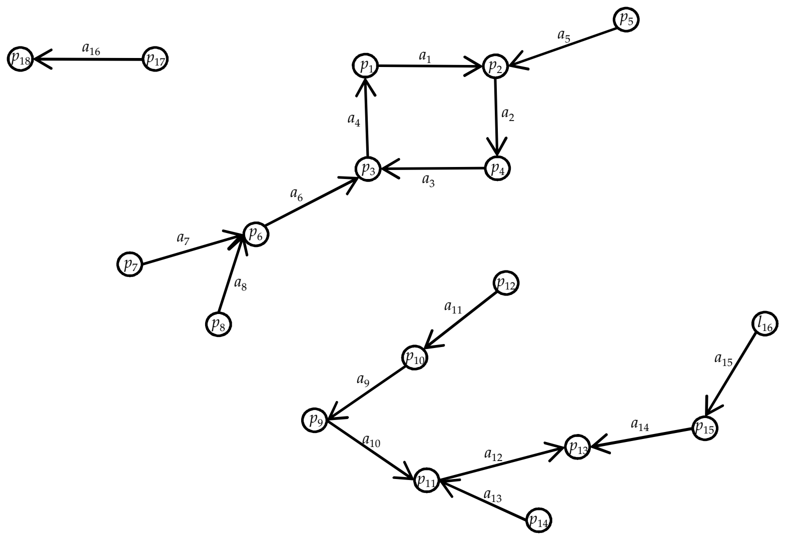

Figure 6: The switching graph for the popular matching M'' obtained by applying the switching path beginning at p_{13} in G_M