Structure and Characterization of Popular Matchings

Bachelor Thesis

Oliver Scheel

RWTH Aachen University, Germany
Lehrstuhl für Operations Research

Advisors:

Universitätsprofessor Dr.rer.nat.habil.	Marco Lübbecke
Dipl.-Math.	Florian Dahms
Universitätsprofessorin Dr. rer. nat.	Britta Peis

Submission Date: 09-29-2014

I hereby affirm that I composed this work independently and used no other than the specified sources and tools and that I marked all quotes as such.

Aachen, September 29th, 2014

Contents

1 Introduction

Matchings are present in many common tasks: Be it the matching of applicants to jobs, the matching of persons to housing opportunities or the matching of students to tutorials. But of course the applications extend much further, matchings are essential in many complex mathematical and other scientific problems. Luckily, the case of finding a simple maximum matching is fairly easy, however, many extensions have been proposed and examined. Most center on the aspects fairness, robustness and complexity.
This work covers the topic of popular matchings, which have recently been studied quite extensively. They are relatively robust against manipulation and use a very natural criterion regarding fairness: A matching M is popular if no other matching beats it by majority vote of the participants. Furthermore, popular matchings can be calculated efficiently. One downside though, is the fact that not every instance allows a popular matching.
The main goal of this work is to give an overview over most of the topics regarding popular matchings known and discussed so far.
Section 2 first gives an overview over different matching criteria as well as some results regarding popular matchings. Then in Section 3, an introduction to popular matchings is given. Here, based on [1], a characterization of popular matchings is worked out, which can be used to efficiently find popular matchings for an instance. In the basic case we consider one sided preferences, where applicants get matched to posts, with each post being able to be matched to at most one applicant. Furthermore, in this section the case, where posts have a greater capacity than one, is considered, which leads to the capacitated house allocation problem. Of course this is again solved using popular matchings. In Section 4 an interesting graph structure, the switching graph as defined in [2], is examined. Using this, many problems related to popular matchings can be solved, like counting and enumerating all popular matchings for an instance. In Section 5 it is shown that popular matchings are not strategyproof, that means there exist possibilities for applicants to manipulate the outcome to their advantage. In this section, strategies for single applicants are listed, who have total knowledge of the preferences of all others, and use this to improve their outcome ([3]). In Section 9, a relaxation of the popular matching criterion is presented, in order to always be able to chose one "best" matching, also if no popular matching exists. This uses the so called least-unpopularity factor and turns out to be NP-complete ([4]).
Another part of this work consists of results which are to my knowledge not yet known or investigated. In Section 6 an alternative characterization of popular matchings is presented. Amongst others this could be used to achieve a combinatoric formula to count the number of instances allowing a popular matchings. An attempt is described in Section 7, but this number could only be approximated here. However, the approximation is very accurate. In Section 8 linear programs are used to formulate popular matchings. First, just for describing solutions to the problem of finding popular matchings. The goal was then to extend this to defining a subgraph polytope consisting of all vertex combinations allowing a popular matching. This could not be done in this context, but

a negative result showing what is not sufficient for this is given. In Section 10 another relaxation of the popular matching definition is shown, called the subset maximal popular matching. Here a subset of the applicants can be neglected, the goal is to find a minimal subset such that a popular matching exists for the remaining applicants. This problem is also in NP, but the lower bound remains unknown. Finally, a .Net program was written which implements most of the algorithms and principles described here.

2 A History of Matchings

Let us limit ourselves here to matchings in bipartite graphs, say between men and women. Although the basic matching problem is simple, when allowing the participants to voice preferences about their matchings, we find a problem we cannot solve mathematically: Every matching makes some participants more happy than others, and this can be the other way around in other matchings. So the question arises: Which matching is "the best"? Of course, if there is a way to make everyone happier in another matching compared to the current one, we should do this - this is a Pareto improvement. However, there may exist many Pareto efficient matchings, and we have to decide between these. Therefor, we need some kind of optimality criterion. Of course, it should be "fair" in some sense, and be robust against manipulations. That is, it should be hard for the participants to get better off by lying about their preferences.
Gale and Shapley were among the first to consider this, in 1962 they wrote about fair college admissions as well as matching men and women in pairs, which is known as the marriage problem ([5]). They introduced the concept of stable matchings: A matching is unstable, if there exist two people of opposite sex, who prefer to be matched to each other rather than to their current partner. This is a very useful and needed definition, as in an unstable matching people could oppose the proposed one, which might be against the purpose of a matching and could leave some people very bad off.
Other good criteria are for example rank-maximal or maximum utility matchings. These value that matching most, in which for example the sum of the rank numbers or the sum of some utility function for each participant is greatest. However, one has to be careful here about cheating, one prominent example being the MIT. They once assigned students to dormitories by minimizing the sum of the cubes of the rank numbers. Students then could improve their chances of getting their true first choice by inserting other highly popular dormitories near the top of their preference list ([4]).
The concept of popular matchings was first introduced by Gärdenfors in 1975 in the context of the stable marriage problem ([6]). A matching M is popular if and only if there is no other matching M', such that the number of people preferring M' over M is greater than the number of people preferring M over M'. Not every instance though allows a popular matching and if it does, there may exist multiple ones.
This concept found no more consideration until Abraham and Irving proposed the first polynomial time algorithm for it in 2005 ([1]). In the years after, many papers regarding popular matchings have been published. Manlove and Sng introduced a capacitated version ([7]) in 2006, Abraham and Kavitha also in 2006 a dynamic version, where agents can enter and leave the market ([8]). Kavitha and Nasre ([9]) further introduced different optimality criteria among popular matchings. Then we do not simply wish to find any popular matching, but if there exist multiple ones, we want to chose one which is best with respect to some criterion. Kavitha and Nasre defined for example rank-maximal popular matchings and "fair" popular matchings. In 2008 a structure with interesting properties regarding popular matchings, the switching graph, was introduced by McDermid and Irving ([2]). This concept allows the characterization of dependencies

between single popular matchings and helps to solve some problems related to popular matchings. Based on this, in 2013 finally a paper was published by Nasre ([3]), which defines cheating strategies for single applicants using the switching graph.

3 An Introduction to Popular Matchings

3.1 Preliminaries

An instance of the popular matching problem (POP-M) consists of a set of applicants (A) and a set of posts (P). Every applicant has a preference list, describing how high he regards the different posts. These posts are acceptable for him. The highest rank is 1, which means, the applicant prefers this post above all others, the second highest rank is 2 etc. Multiple posts can have the same rank in a preference list.
A matching M in a POP-M instance is a set of applicant-post pairs (a, p), such that p is acceptable for a and each $a \in A$ and $p \in P$ appear in at most one pair. $M(p)$ denotes the applicant assigned to p. An applicant a prefers a matching M' to the matching M if either a is assigned to a post in M' and not assigned to a post in M, or he is assigned to posts in both matchings but prefers $M'(a)$ over $M(a)$.
A matching M' is more popular than a matching M, if the number of applicants who prefer M' to M exceeds the number of applicants who prefer M to M'. A matching is popular if there is no other matching which is more popular.
The following sections are based on [1], wherein we will give useful lemmas which help us to characterize and find popular matchings efficiently. We will do this first for the case where no ties are allowed in the preference lists, and then for the case where ties are allowed.
For simplicity, for each applicant we introduce a unique last resort l, connected to the applicant with the lowest rank. This way, every applicant can be assigned to a post, some are just matched to their last resort. It should be clear that not for every instance a popular matching exists, as the "more popular" relation is not acyclic. In the following instance for example, where the preference lists are denoted next the respective applicants, no popular matching exists: That is, because in each matching we can denote the applicant matched to p_1 to p_3 and then promote the two other applicants:

$a_1 : p_1\ p_2\ p_3$
$a_2 : p_1\ p_2\ p_3$
$a_3 : p_1\ p_2\ p_3$

When examining the runtime, we will denote by n the number of applicants and posts, and by m the sum of the lengths of all preference lists.

3.2 Strictly Ordered Preference Lists

We begin with an analysis of popular matchings for the case that the preference lists need to be strictly ordered. This will turn out to be slightly easier than the general case, we will discuss it to give some intuition for the latter. Like is shown in [1], it turns out that popular matchings for this situation can be found in linear time, which is a bit

of a surprise, since by the general definition one might suspect that all other possible matchings had to be calculated.

3.2.1 Characterizing Popular Matchings Without Ties

For each applicant a let $f(a)$ denote his first-ranked post on his preference list. Any such post we will call an f-post. Let $f(p)$ be the set of applicants for which p is their first-ranked post. The following lemma is a start towards the finding of sufficient criteria to define popular matchings:

Lemma 1. *([1]) Let M be a popular matching. Then for every f-post p, (i) p is matched in M, and (ii) $M(p) \in f(p)$.*

Proof. ([1]) Suppose some f-post p is unmatched. We could promote any applicant from $f(p)$ to p, thus creating a more popular matching, which is a contradiction. Otherwise suppose every f-post is matched, but some f-post p is matched to some applicant a, such that $a \notin f(p)$. Select any applicant, say a_1, from $f(p)$. Let $a_2 = M(p)$ and $a_3 = M(f(a_2))$ (a_3 is well defined because all f-posts are matched M). When demoting a_3 to $l(a_3)$, promoting a_2 to $f(a_2)$ and promoting a_1 to p, we again get a more popular matching. □

For every applicant a, let $s(a)$ denote the first non f-post on his preference list and call any such post an s-post. Note that for every applicant one s-post has to exist due to the introduction of l-posts.
In the next two lemmas we will show that in every popular matching applicants can only be matched to their f- or s-post.

Lemma 2. *([1]) Let M be a popular matching. Then for every applicant a, $M(a)$ is never strictly between $f(a)$ and $s(a)$.*

Proof. ([1]) Suppose for a contradiction that $M(a)$ is strictly between $f(a)$ and $s(a)$. That means, $M(a)$ is an f-post. Because of Lemma 1, for every f-post p, $M(p) \in f(p)$, contradicting the assumption that $M(a)$ is not $f(a)$. □

Lemma 3. *([1]) Let M be a popular matching. Then for every applicant a, $M(a)$ is never worse than $s(a)$.*

Proof. ([1]) Again suppose for a contradiction that $M(a_1)$ is worse than $s(a_1)$. If $s(a_1)$ is unmatched, we can promote a_1 to $s(a_1)$, constructing a more popular matching. Otherwise, let $a_2 = M(s(a_1))$ and $a_3 = M(f(a_2))$. Note that $a_2 \neq a_3$, because f- and s-posts are disjoint. We then can construct a more popular matching by promoting a_2 to $f(a_2)$, promoting a_1 to $s(a_1)$ and demoting a_3 to $l(a_3)$. □

Thus we already have derived an easy but important classification of popular matchings:

Lemma 4. *([1]) A matching M is popular iff*
(i) every f-post is matched, and
(ii) for every applicant a, $M(a) \in \{f(a), s(a)\}$.

Proof. ([1]) By the previous lemmas we already know that, if a matching is popular, it satisfies (i) and (ii). Now we have to show the reverse. Let M be a matching satisfying (i) and (ii) and suppose for a contradiction that there exists a matching M', which is more popular. Let a be any applicant which prefers M' over M. Thus, he has to be matched to his f-post p in M', and to his s-post in M. But since all f-posts have to be matched, p has to be matched in M too, say to a', with $p = f(a')$ (because of condition (ii) and the fact that f- and s-posts are disjoint). Thus a' prefers M over M', and this way for any applicant preferring M' we can find an applicant preferring M, contradicting the fact that M' is more popular. □

3.2.2 Algorithmic Results

Based on this we can specify Algorithm 3.1 ([1]), whose correctness is immediately proven by Lemma 4. We just have to remark, that after the termination of the loop all f-posts are

Algorithm 3.1 Algorithm for the Popular Matching Problem Without Ties

```
1: Popular Matching (G = (A ∪ P, E))
2: G' := reduced graph of G
3: if G' admits applicant complete matching M then
4:    for all f-post p unmatched in M do
5:       Let a be any applicant in f(p)
6:       Promote a to p in M
7:    end for
8:    return M
9: else
10:   return "no popular matching"
11: end if
```

matched, because every applicant has a unique f-post and these are disjoint from s-posts. Let us look at the runtime, as linear time is not obvious yet. It is clear though that the reduced graph of the instance can be constructed in $O(m+n)$. The loop also runs in linear time collecting the list of applicants in $f(p)$ for any p during the construction. It remains to show that the checking for an applicant complete matching can be done in linear time. Using a standard approach, e.g. finding a maximum-cardinality matching with the Hopcroft-Karp algorithm and then checking for applicant-completeness, takes super-linear time, namely $O(n^{3/2})$. Therefore we use the Algorithm 3.2 ([1]). In that algorithm first all posts with degree 1 are matched to their corresponding applicant. Because these

Algorithm 3.2 Algorithm for Checking for an Applicant-Complete Matching

```
1: Applicant-complete matching (G' = (A ∪ P, E'))
2: M := ∅
3: while some post p has degree 1 do
4:    a := unique applicant adjacent to p
5:    M := M ∪ {(a, p)}
6:    G' := G' − {a, p}
7: end while
8: while some post p has degree 0 do
9:    G' := G' − {p}
10: end while
11: if |P| < |A| then
12:    return "no applicant-complete matching"
13: else
14:    M' := any maximum-cardinality matching of G'
15:    M ∪ M'
16: end if
```

posts are matched and have degree 1, they cannot appear in any further augmenting path and we can remove them. This can obviously be done in linear time. Next, all posts with degree 0 are removed, so that all posts have degree at least 2, while all applicants still have degree 2. Now, if $|P| < |A|$, by Hall's marriage theorem there cannot be an applicant-complete matching. Otherwise $|P| \geq |A|$, and $2|P| \leq \sum_{p \in P} deg(p) = 2|A|$, so $|P| = |A|$ and all posts have degree exactly 2. Therefore, G' decomposes into a set of disjoint cycles and we can simply walk over these cycles, choosing every second edge.

Lemma 5. *([1]) For instances with strictly ordered preference lists, in time $O(n + m)$ a popular matching can be found, if one exists, or decided, that none exists.*

One interesting, a little further leading task, is to find a maximum-cardinality popular matching. As mentioned before, not every instance allows a popular matching, but if there exists a popular matching there can possibly be many. In the maximum-cardinality popular matching problem we now want to find that popular matching, which allocates the most people, that is, that allocates the fewest people to their last resort. With small changes in the algorithm in [1] it is shown that also the following holds:

Theorem 6. *([1]) For instances with strictly ordered preference lists, in time $O(n+m)$ one can find a maximum-cardinality popular matching, if one exists, or decide, that no popular matching exists.*

3.3 Preference Lists With Ties

In this section we will relax the assumption of no ties and also allow applicants to assign different posts the same rank. For this no algorithm which runs in linear time is given, but one that runs in time $O(\sqrt{n}m)$. It can also not be hoped to find a linear time algorithm, since the special case, where all edges have rank one, is equivalent to the finding of a maximum-cardinality matching in a bipartite graph, for which the fastest known algorithm (the Hopcroft-Karp algorithm, [10]) runs in $O(\sqrt{n}m)$. (This is, because if all edges have rank one, a matching is popular if there is no other matching which matches more applicants - thus if it is a maximum-cardinality matching.) That means, the general popular matching problem is at least as hard as the maximum-cardinality matching problem.

3.3.1 Characterizing Popular Matchings With Ties

The first difference, in contrast to the case of no ties, one can see is, that it is now not longer possible to match all f-posts, as there can be more f-posts than applicants. Due to this, also the characterization of s-posts gets slightly more difficult, now remaining f-posts can also take up the role of s-posts. In the following paragraphs generalizations of the previous lemmas will be worked out.
Like before, for each applicant a let $f(a)$ denote his first-ranked posts on his preference list. Any such post we will call an f-post and let $f(p)$ be the set of applicants for which p is their first-ranked post. We define the first-choice graph of G as $G_1 = \{A \cup P, E_1\}$, whereat E_1 contains all edges with rank 1.

Lemma 7. *([1]) Let M be a popular matching. Then $M \cap E_1$ is a maximum matching in G_1.*

Proof. ([1]) If $M_1 = M \cap E_1$ is not a maximum matching, there has to exist an augmenting path $Q = \langle a_1, p_1, \ldots, p_k \rangle$. Then $M(a_1) \neq f(a_1)$ and p_k is either unmatched in M or $M(p_k) \neq f(p_k)$. Let us consider these two cases.
(i) p_k is unmatched in M:
Since both a_1 and p_k are unmatched in M_1, we augment M with Q. Now a_1 is matched to a post he regards higher and all other applicants remain matched to a rank 1 post. This gives a more popular matching, a contradiction.
(ii) p_k is matched in M:
Let $a_{k+1} = M(p_k)$ and note that $p_k \notin f(a_{k+1})$. Remove (a_{k+1}, p_k) from M and augment M with Q. Select any $p_{k+1} \in f(a_{k+1})$. If p_{k+1} is unmatched in M, we promote a_{k+1} to p_{k+1}. Now a_1 and a_{k+1} prefer the new matching to M. If p_{k+1} is not unmatched in M, we denote $a = M'(p_{k+1})$ to either $l(a)$ (if $a \neq a_1$), or back to $M(a_1)$ (if $a = a_1$) and then promote a_{k+1} to p_{k+1}. In the first case a prefers the old matching, but both a_1 and a_{k+1} prefer the new matching. In the second case, a_1 prefers the new matching. Again we have constructed a more popular matching, a contradiction. □

Let us come to the definition of s-posts. As already mentioned, now s-posts can contain a number of surplus f-posts, since not all f-posts have to be or can be matched. But we can exclude some f-posts to restrict the set of possible s-posts. For any maximum matching we can partition the nodes into the sets odd ($\mathcal{O}$), even ($\mathcal{E}$) and unreachable ($\mathcal{U}$). A node v is odd (even), if there exists an alternating path of odd (even) length from an unmatched node to v. If there exists no such path, v is unreachable.

Lemma 8. *([1]) Let $\mathcal{O}$, $\mathcal{E}$ and $\mathcal{U}$ be the sets defined above.*
(a) $\mathcal{O}$, $\mathcal{E}$ and $\mathcal{U}$ are pairwise disjoint. Every maximum matching provides the same partitions.
(b) In every maximum matching, every node in $\mathcal{O}$ is matched with some node in $\mathcal{E}$ and every node in $\mathcal{U}$ is matched with another node in $\mathcal{U}$. The size of any maximum matching is $|\mathcal{O}| + |\mathcal{U}|/2$.
(c) No maximum matching contains an edge between two nodes in $\mathcal{O}$, between an node in $\mathcal{O}$ and in $\mathcal{U}$ or between a node in $\mathcal{E}$ and a node in $\mathcal{U}$.

Proof. See ([1]). □

Since by Lemma 7 M_1 is a maximum matching in G_1, with the previous lemma we now get, that every odd and unreachable post p in G_1 must be matched in M_1. These posts cannot be s-posts, so we define $s(a)$ as the set of the even posts in $a's$ preference list with the same highest rank. Since the l-posts are never matched in M_1, they are always even and thus candidates for the s-posts, so $s(a) \neq \emptyset$ for all posts.

Lemma 9. *([1]) Let M be a popular matching. Then for every applicant a, $M(a)$ can never be strictly between $f(a)$ and $s(a)$.*

Proof. ([1]) Suppose for a contradiction that $M(a)$ is strictly between $f(a)$ and $s(a)$. Since a prefers $M(a)$ to any post in $s(a)$ and these are the top ranked even posts, $M(a)$ has to be an odd or unreachable node. By the previous lemma, these have to be matched in every maximum matching of M_1. But since $M(a)$ is supposed to be strictly worse than $f(a)$, $M(a)$ is not matched in M_1 and M is not a popular matching. □

Lemma 10. *([1]) Let M be a popular matching. Then for every applicant a, $M(a)$ is never worse than $s(a)$.*

Proof. ([1]) Suppose for a contradiction that $M(a_1)$ is strictly worse than $s(a_1)$. Let p_1 be any post in $s(a_1)$. If p_1 is unmatched in M, we can, by promoting a_1 to p_1, construct a more popular matching. Otherwise, let $a_2 = M(p_1)$. We distinguish two cases.
(a) $p_1 \notin f(a_2)$: Select any post $p_2 \in f(a_2)$ and let $a_3 = M(p_2)$. We can construct a more popular matching by (i) demoting a_3 to $l(a_3)$, (ii) promoting a_2 to p_2 and (iii) promoting a_1 to p_1.
(b) $p_1 \in f(a_2)$: Since p_1 is an s-post now as well, p_1 must be even. So G_1 contains

an even length alternating path $Q = \langle p_1, a_2, \ldots, p_k \rangle$ from p_k to p_1. If p_k is unmatched, $M \oplus Q$ is a more popular matching. If not, let $a_{k+1} = M'(p_k)$ (note that $a_{k+1} \notin f_{a_{k+1}}$). Remove (a_{k+1}, p_k) from M' and augment M' with Q. Select any $p_{k+1} \in f_{a_{k+1}}$. If p_{k+1} is unmatched in M', promote a_{k+1} to p_{k+1}. Otherwise, we demote $a = M'(p_{k+1})$ to either $l(a)$ (if $a \neq a_1$) or otherwise back to $M(a_1)$. Then promote a_{k+1} to p_{k+1}. □

The previous lemmas hinted the following characterization of a popular matching:

Lemma 11. *([1]) A matching is popular iff*
(i) $M \cap E_1$ is a maximum matching in G_1, and
(ii) for each applicant a, $M(a) \in \{f(a) \cup s(a)\}$.

Proof. ([1]) By the previous lemmas all popular matchings satisfy condition (i) and (ii). Let M be a matching satisfying (i) and (ii) and suppose there is a more popular matching M'. For every applicant that prefers M' we will find an applicant preferring M. The graph $H = (M \oplus M') \cap E_1$ consists of disjoint cycles and paths. Suppose applicant a prefers matching M'. That means he is assigned to his second post in M, which is his top ranked even node, and therefor must be matched to an odd or unreachable node in M'. This node has to be also matched in M. Since $M'(a) \neq M(a)$, a is not isolated in H. So a belongs to some non-empty path Q in H. Since $M'(a)$ is odd or unreachable, every post in Q must also be odd or unreachable and thus matched in $M \cap E_1$ too, and only another applicant, say a' can be the endpoint. This applicant though prefers M to M', because $M(a') \in f(a')$ but $M(a) \notin f(a')$. So we found the desired applicant. □

Now we define the reduced subgraph G' out of G by deleting all edges from each applicant a except to $f(a) \cup s(a)$. Note that G' need not admit an applicant-complete matching anymore since $l(a)$ is now isolated if $s(a) \neq \{l(a)\}$. Thus we get the final characterization:

Lemma 12. *A matching is popular iff*
(i) $M \cap E_1$ is a maximum matching in G_1
(ii) M is an applicant-complete matching in G'

3.3.2 Algorithmic Results

As before, the direct realization of the characterization leads to an efficient algorithm, namely Algorithm 3.3. To show the correctness of the algorithm we first have to show, that successively augmenting the matching M_1 does not destroy the maximum matching on rank-1 edges and that it thus returns a maximum matching M on rank-1 edges. Let us first consider Step 3, we will show that here only edges of rank 1 are deleted. Any odd or unreachable post p is by definition never contained in $s(a)$ for any applicant a, so no edges of the form (p, a), $p \in s(a)$ are deleted. For any odd applicant a, there exists by definition an alternating path from a in M_1 unmatched node to a. So a is connected

Algorithm 3.3 Algorithm for the Popular Matching Problem With Ties

1: Compute maximum matching M_1 in G_1
2: Construct G' using it
3: Delete all edges connecting two nodes from $\mathcal{O}$ or a node in $\mathcal{O}$ with a node in $\mathcal{U}$
4: Compute maximum matching in G' by augmenting M_1
5: **if** M is applicant-complete **then**
6: return M
7: **else**
8: return "no popular matching"
9: **end if**

to even posts which are his highest ranked even nodes. Simultaneously, these are his f-posts, because there are no edges from an odd node to odd or unreachable nodes in G_1. So $s(a) \subseteq f(a)$. The case that an applicant is odd, and an adjacent post unreachable never occurs as then the applicant would be even. So the edges removed in Step 3 are all rank-1 edges, which anyway cannot be used by any maximum matching on rank-1 edges by Lemma 8 c). After the edge deletion the only neighbors of nodes in $\mathcal{O}$ remain the nodes in $\mathcal{E}$ and the only neighbors of nodes in $\mathcal{U} \cap A$ are nodes in $\mathcal{U} \cap P$. Since M is obtained by augmenting M_1, nodes matched in M_1 stay matched in M. That means, that M has to match every node in $\mathcal{O}$ with a node in $\mathcal{E}$ and every node in $\mathcal{U}$ with another node in $\mathcal{U}$. So in M there are at least $|\mathcal{O}| + |\mathcal{U} \cap P| = |\mathcal{O}| + |\mathcal{U}|/2$ matched edges, which have rank 1. By Lemma 8 b) this is thus a maximum matching on rank-1 edges.
This proves one necessary condition for being a popular matching, the other, that the matching must be applicant complete, is secured by the IF clause in the last step.
To find a maximum matching in G_1 and compute G' we use the Hopcroft-Karp Algorithm, which takes time $O(\sqrt{n}m)$. Then we iteratively augment this matching using also the Hopcroft-Karp Algorithm, and get a final runtime of $O(\sqrt{n}m)$. In ([1]) again a slight modification of Algorithm 3.3 is presented to find a maximum-cardinality popular matching.

Lemma 13. *([1]) In time $O(\sqrt{n}m)$ a popular matching can be found, if one exists, or decided, that none exists.*

3.4 Usage in the Capacitated House Allocation Problem

In this chapter we will shortly mention the possibility that posts have a capacity, which is known as the Capacitated House Allocation Problem. Again we will first consider the case without ties (CHA), and then with ties (CHAT). Manlove and Sng analyzed the use of popular matchings in this context ([7]). On the one hand, they mentioned that instances can be "inflated", this principle will also be used here, and then Algorithm 3.1 or Algorithm 3.3 can be run on them. On the other hand, they repeated the above lemmas in an alternative version for posts with capacities and came up with slightly faster algorithms. By "inflating" an instance we mean that we replace every post p with capacity c by c posts with capacity 1, keeping the edges. So if post p appeared on an applicant's preference list, we replace this edge by c edges with the same rank pointing to the newly created c posts. Then we can run Algorithm 3.3, eventually getting a popular matching, if one exists. We denote the original instance by O and the inflated one by I. The new posts created instead of post i we name $s_{i1} \dots s_{ic(i)}$. From a matching in O we get the corresponding matching in I by matching the k to p assigend applicants to the posts $s_{p1} \dots s_{pk}$ and vice versa.
To be able to apply the lemmas from before, we need the following easy lemma:

Lemma 14. *O admits a popular matching $\Leftrightarrow$ I admits a popular matching.*

Proof. (i) $\Rightarrow$: Show this by contraposition. Suppose I does not admit a popular matching. This means, for any matching M there exists a more popular matching M', so at least one applicant a prefers M' to M. Let $M(a) = s_{ab}$, $M'(a) = s_{cd}$. So in the corresponding matching of M' in O a prefers tutorial c to a, and this matching is more popular than then corresponding matching of M.
(ii) $\Leftarrow$: Show this by contraposition. Suppose O does not admit a popular matching. This means, for any matching M there exists a more popular matching M', so at least one applicant a prefers M' to M. Let $M(a) = p_1$, $M'(a) = p_2$. Then in the corresponding matching of M' in I, a prefers the corresponding post of p_2 to that of p_1, and this matching is more popular than the corresponding matching of M.

□

3.4.1 CHA

In this section we will repeat the results from [7] regarding the House Allocation Problems without ties, which is known as CHA. After adapting the lemmas, Manlove and Sng came up with

Theorem 15. *[7] Let C be the sum of all capacities, n_1 the number of applicants and m the sum of the lengths of the preference lists. For an instance of CHA in time $O(\sqrt{C}n_1 + m)$ a popular matching can be found, if one exists, or decided, that there exists none.*

When applying Algorithm 3.3 to the inflated instance, we get a running time of roughly $O(n\sqrt{mC})$. Manlove and Sng also showed a better estimation of $O(\sqrt{C}mc_{min})$, so this method is slower by a factor of $\Omega(\sqrt{C}c_{min})$ or $\Omega(mc_{min}/n_1)$, depending on whether $\sqrt{C}n_1 \leq m$ or $\sqrt{C}n_1 > m$ ([7]).

3.4.2 CHAT

This section covers the general case of the House Allocation Problem, where ties are allowed. By modifying the lemmas from [1], in [7] the following was shown:

Theorem 16. *[7] Let C be the sum of all capacities, n_1 the number of applicants and m the sum of the lengths of the preference lists. For an instance of CHAT in time $O((\sqrt{C}+n_1)m)$ a popular matching can be found, if one exists, or decided, that there exists none.*

When again comparing this result to applying Algorithm 3.3 on the inflated instance, we find a time advantage of $\Omega(c_{min})$ ([7]).

4 The Switching Graph

In this section we will discuss a structure introduced in [2], the switching graph, which is a useful tool to solve many problems related to popular matchings, amongst others counting the number of popular matchings for an instance or generating them randomly. In this section we will only define the switching graph for instances with strictly ordered preference lists, in Section 5 we will generalize this concept for all instances and use it then to define cheating strategies for the popular matching problem.

4.1 Structure and Properties of the Switching Graph

By the previous lemmas we know that any applicant a in a popular matching can only be matched to either his f- or s-post. Call the post matched to him $M(a)$, the other one $O_M(a)$. Given a popular matching M for an instance I the switching graph G_M is defined as follows:

Definition 17. *([2]) G_M is a directed graph with one vertex for each post in M, and a directed edge (p_i, p_j) for each pair of posts p_i and p_j, if $M(a) = p_i$ and $O_M(a) = p_j$.*

We will use vertices and posts, and edges and applicants interchangeably. We call an vertex f-post vertex (respectively s-post vertex), if its corresponding post is an f-post (respectively s-post). The edges of the graph will be labeled with the corresponding applicants. We say an applicant belongs to a component, if his incident post vertices belong to that component.
Basic properties of the switching graph are given by the following lemma:

Lemma 18. *([2]) Let G_M be the switching graph of a popular matching M for an instance I. Then the following holds:*

1. *Each vertex in G_M has outdegree at most 1.*
2. *The sink vertices of G_M are those vertices which correspond to unmatched posts in M and are all s-post vertices.*
3. *Each component of G_M contains either a single sink vertex or a single cycle.*

Proof. ([2])

1. Any outgoing edge of a vertex represents a matching of that post to some applicant. Since M is a matching, each post can only be matched once.

2. A vertex with no outgoing edge represents an unmatched post. By Lemma 1 we know that every f-post has to be matched in a popular matching, therefore any unmatched post has to be an s-post.

3. This is an easy consequence of (1).

□

A component of a switching graph G_M we call a cycle component or a tree component, depending on whether it contains a cycle or a sink. Each cycle in G_M we call switching cycle. If T is a tree component in G_M with sink p and q is another s-post vertex in T, we call the unique path from q to p a switching path. So each component of G_M contains either one switching cycle, or if it does not, it can contain multiple switching paths. More exactly, for each s-post vertex other than the sink there exists one switching path from this vertex to the sink. It should be clear that the cycle and tree components of G_M can be found in linear time, for example with a depth-first search.
The following example should clarify this design. There are 8 applicants, a_1 to a_8, which have the following properties and are matched to the underlined post:

a_1: $\underline{p_1}$ p_2
a_2: p_3 $\underline{p_2}$
a_3: $\underline{p_3}$ p_4
a_4: p_1 $\underline{p_4}$
a_5: $\underline{p_5}$ p_2
a_6: $\underline{p_6}$ p_7
a_7: $\underline{p_8}$ p_7
a_8: $\underline{p_9}$ p_7

The resulting switching graph is depicted in Figure 1.

In the next paragraphs we will explain the importance of the switching graph and show how to use it to find all popular matchings.
For a matching M, to apply a switching cycle C means to match every applicant a in it to $O_M(a)$ but leave all other applicants matched as in M. Similiar, to apply a switching path P means, to match every applicant a in it to $O_M(a)$ but leave all other applicants matched as in M. We denote the resultings matchings with $M \cdot C$ respectively $M \cdot P$.

Theorem 19. *([2]) Let M be a popular matching for an instance I. Then:*
(i) If C is a switching cycle in G_M, $M \cdot C$ is a popular matching in I.
(ii) If P is a switching path in G_M, $M \cdot P$ is a popular matching in I.

Proof. ([2]) (i) By Lemma 4 it is sufficient to show that any applicant in the new matching is only matched to his f- or s-post and that all f-posts are matched. By the definition of the switching graph it is obvious that in $M \cdot C$ any applicant still can only be matched to his f- or s-post. Furthermore, in the cycle the matching partner of all applicants are

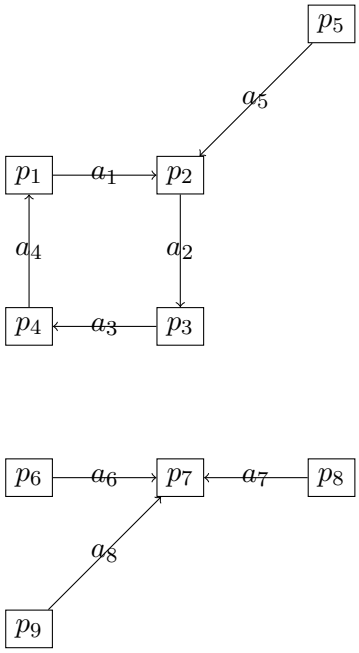


Figure 1: An exemplary switching graph

changed, but the same posts remain matched.
(ii) When examining $M \cdot P$ it should also be obvious, that every applicant can only be matched to his f- or s-post. The only post vacated after applying the path is its starting vertex p which is, by definition, an s-post vertex. Thus all f-posts are still matched. □

Theorem 19 shows, that we can achieve new popular matchings out of a given one by applying switching circles and paths. We now want to show that this is also the only way to find these matchings, given some existing ones. More precisely we will show that any popular matching for an instance I can be constructed out of any popular matching M for that instance by only applying switching circles and paths.

Lemma 20. *([2]) Let M be a popular matching for an instance I and M' another arbitrary popular matching for it. 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'.

Proof. This is an easy consequence of the definition of the switching graph. □

The following lemmas, Lemma 21 and 22, consider switching cycles and switching paths respectively:

Lemma 21. *([2]) Let M be a popular matching for an instance I, M' another arbitrary popular matching for it and T a cycle component in G_M with switching cycle C. Then the following holds:*
(i) Either for every applicant a in C $M'(a) = M(a)$, or for every applicant a in C $M'(a) = O_M(a)$.
(ii) For every applicant in T, which is not in C, $M'(a) = M(a)$.

Proof. ([2]) (i) Because of the cyclical structure, if one applicant a in C is matched to $O_M(a)$ in M', all the others have to be, too.
(ii) Let C consist of the posts $p_1, \dots p_k$ and let p_p be the last post before the cycle, meaning for the applicant a matched to it $O_M(a) = p_i$ for some p_i in C. Suppose $M(a) \neq M'(a)$. Then p_p must be matched to p_i in M'. By (i) we know that then also all other applicants in C must be matched differently in M' than in M. This leads to a contradiction, since then p_i would have to be matched also to the applicant corresponding to the edge (p_{i-1}, p_i). □

Lemma 22. *([2]) Let M be a popular matching for an instance I, M' an arbitrary other popular matching for it and T a tree component in G_M. Then the following holds:*
Either for every applicant a in T $M'(a) = M(a)$, or there exists a switching path P in T, such that for every applicant a in P $M'(a) = O_M(a)$, and for every applicant in T, which is not in P, $M'(a) = M(a)$.

Proof. ([2]) Suppose $M(a) \neq M'(a)$ for some a, by a similar argument as in Lemma 21 (i) the same has to hold for all other applicants on the path from a to the sink vertex. Now suppose two applicants in T whose edges have a common end point, say p, are both matched to different posts in M' than in M. This leads to a contradiction, since both had to be assigned to p by Lemma 20. Thus the applicants who are assigned to different posts in M' than in M form a path ending at the sink vertex. Furthermore, the path has to start at an s-post vertex, because its starting post will be unmatched in M', a state which is not allowed for f-posts in popular matchings. Therefore, the path is a switching path. □

As we have seen before, popular matchings differ with respect to switching components only in the essential parts of the component, their circles or paths. That means, if we have a popular matching M and two switching components T and T' and we apply the switching cycle or some switching paths of T (depending on what kind of component T is) to M, then the applicants from T' are unaffected, T' can still be found in $M \cdot T$. This notion of independence is captured in the next lemma:

Lemma 23. *([2]) Let T and T' be components of a switching graph G_M for a popular matching M, and let Q either be a switching cycle or a switching path (depending on whether T is a cycle or a tree component) in T. Then, T' is a component in the switching graph $G_{M \cdot Q}$.*

Now we can prove a lemma that shows, that we can indeed construct all popular matchings of an instance out of an arbitrary popular matching by only applying switching circles and paths:

Lemma 24. *([2]) Let M and M' be two popular matchings for an instance I of POP-M. Then M' can 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. ([2]) We will describe a procedure of obtaining M' from M which works in a way as required in the claim.
For any circle component we know that either for each applicant in the circle $M'(a) = M(a)$ or $M'(a) = O_M(a)$. In the first case we do nothing, in the second one we apply the circle to M, so that every applicant in it becomes matched to $M'(a)$.
For any tree component we know, that either for each applicant in it $M'(a) = M(a)$, or there exists exactly one switching path, for whose applicants $M'(a) = O_M(a)$. Exactly like above, in the first case we do nothing, in the second one we apply the switching path so that every applicant in it becomes matched to $M'(a)$. Thus we obtain M' out of 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, which proves the desired. Moreover, the order in which the assignments are done, is irrelevant. □

We will formulate this important procedure in

Corollary 25. *([2]) Let I be a POP-M instance and let M be any arbitrary popular matching for I with the switching graph G_M. Denote with $X_1, \ldots, X_k$ the tree components of G_M, and with $Y_1, \ldots, Y_l$ the cycle components of G_M. Then, the set of popular matchings for I consists of exactly those matchings which are obtained by applying at most one switching path in X_i for $i \in \{1, \ldots, k\}$ and by applying at most one switching cycle in Y_i for $i \in \{1, \ldots, l\}$.*

4.2 Algorithms That Exploit the Structure

In this section we show how to use the switching graph to solve a number of problems efficiently, namely counting popular matchings, enumerating them and generating them randomly.
All algorithms begin the same way, they calculate the reduced instance, then run Algorithm 3.1 on it to find an arbitrary popular matching (if one exists) and then construct the switching graph and identify the different components, for example by depth-first search. This sequence we call the pre-processing phase - as discussed before, this can be done in $O(n+m)$ time, where n is the number of applicants and posts and m is the sum of lengths of the original preference list.

The list of algorithms in the following section does not contain every algorithm of the original paper, the sections about popular pairs and different optimality criteria for popular matchings are missing. Furthermore, also proofs of some of the following theorems will be omitted and instead we will refer to the original paper.

4.2.1 Counting Popular Matchings

Recall that a tree component having q s-posts has exactly $q-1$ switching paths and that, in order to get from one popular matching to another, at most one of these can be applied. For a tree component X_i we denote by $S(X_i)$ the number of s-posts in X_i. That is also the number of choices we have for a tree component (apply one of the $q-1$ switching paths or take none), and since we either can or cannot apply a cycle from any cycle component the following theorem is an immediate consequence of Corollary 25:

Theorem 26. *([2]) 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 $X_1, \dots X_k$ and the cycle components of G_M be $Y_1, \dots Y_l$. Then the number of popular matchings for this instance is $2^l * \prod_{i=1}^{k} S(X_i)$.*

Thus an algorithm for counting the number of popular matchings for a given instance would first perform the pre-processing phase, during which also the number of circles and the $S(X_i)$ are calculated, and then evaluate the formula above. With the linear running time of Algorithm 3.1 we get

Theorem 27. *([2]) The number of popular matchings for an arbitrary instance of POP-M can be calculated in linear time.*

4.2.2 Enumerating Popular Matchings

This section describes an algorithm to enumerate all popular matchings for an instance I.
Again, the algorithm starts with the pre-processing phase, it finds an arbitrary popular matching M for I (if existing) and calculates the switching graph. Also, for each tree component X_i $S(X_i)$ is calculated and the s-posts other than the sink are numbered $1, \dots q-1$, where $q = S(X_i)$. Let the number of cycle components be l and the number of tree components be k. Next a vector $V = (v_1, \dots, v_j)$ is defined with $j = l + k$, where $v_i \in \{0, 1\}$, $1 \leq i \leq l$ and $v_{l+i} \in \{0, 1, \dots S(X_i)\}$, $1 \leq i \leq k$.
The algorithms then outputs M as the first popular matching and initializes V with $(0, \dots, 0)$. Then all possible values of the vector are looped through. In each iteration the cycle i is applied to M if $v_i = 1$ $(1 \leq i \leq l)$. Furthermore, in each iteration, if $v_{l+i} \neq 0$ $(1 \leq i \leq k)$, the switching path beginning with node v_{l+i} of tree component i is applied. The resulting matching is then outputted in that iteration.

Theorem 28. *([2]) 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. See [2]. □

4.2.3 Random Popular Matchings

The same principle as in the previous section can be applied to generate random popular matchings. For this the vector V is simply generated randomly: For each cycle component a random bit is generated, for the tree components a number between 0 and $q - 1$ (if that tree component has $S(X_i) = q$) is chosen uniformly at random.

Theorem 29. *([2]) 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.*

5 Cheating Strategies

In this section we will show that popular matchings can be manipulated. The finding, that this is possible, is important for the general question how fair popular matchings are. We will consider the case, where one applicant is aware of the preferences of all others, and define cheating strategies, which allow him in some cases to get matched to a higher ranked post when falsifying his true preferences according to the strategies. These strategies were introduced in [3], we will follow its structure closely. To define the cheating strategies, we will generalize the definition of the switching graph from the previous chapter to general instances where ties are allowed. The optimal cheating strategies can be found in polynomial time.
Important in this section is the notation of a "better always" strategy. This is a strategy, which allows an applicant to always be better off by changing his preferences. Formally, an applicant a_1 is always better off in a falsified instance H compared to the original instance G, if (i) every popular matching in H matches him to a post that is at least as good as the most preferred post that he gets matched to in G, and (ii) there exists some popular matching in H that matches a_1 to a post that is better than the most preferred post p he gets matched to in G, assuming that p is not a rank-1 post for a_1.
First we will define a generalization of the switching graph defined in Section 4, with the difference that here ties are allowed in the preference lists of the applicants. Then we will prove some essential lemmas which are needed for later. In Theorem 44 and 48 we then prove two important characteristics of the instance in which the cheating applicant has falsified his preferences. Using all this, in Section 5.4 we eventually define the cheating strategies for two possible types of applicants. Theorem 54 expresses the time effort needed to find these strategies.
Now let us first list Algorithm 5.1 to find a popular matching in instances where ties are allowed, which is the same as the one presented in Section 4, except that more edges are deleted. The correctness of this algorithm is immediate because of the correctness of the original algorithm. After having found a solution we simply delete more edges. Because every popular matching is an applicant-complete and thus a maximum matching in G', we can again partition the nodes, this time according to M in G' and know that no popular matching contains edges between a node in $\mathcal{O}_2$ and a node in $\mathcal{O}_2 \cup \mathcal{U}_2$. However, since every matching matches all applicants, this implies that $A \cap E_2 = \emptyset$ and thus $P \cap O_2 = \emptyset$, which means that there are no $\mathcal{O}_2\mathcal{O}_2$ edges. Therefore, any edge deleted in Step 13 of Algorithm 5.1 is of the form (a, p), where $a \in \mathcal{O}_2$ and $p \in \mathcal{U}_2$. With this knowledge the following claim is immediate:

Claim 30. *In Step 13 of Algorithm 5.1, no edge incident to an applicant $a \in \mathcal{U}_2$ is deleted. In Step 4 of Algorithm 5.1, no edge incident to an applicant $a \in \mathcal{E}_1$ is deleted.*

Algorithm 5.1 Algorithm for the Popular Matching Problem With Ties

1: Compute maximum matching M_1 in G_1
2: Construct G' using it
3: Partition $A \cup P$ as $\mathcal{O}_1$, $\mathcal{E}_1$ and $\mathcal{U}_1$ with respect to M_1 in G_1
4: Delete all edges connecting two nodes from $\mathcal{O}_1$ or a node in $\mathcal{O}_1$ with a node in $\mathcal{U}_1$
5: Compute maximum matching in G' by augmenting M_1
6: **if** M is applicant-complete **then**
7: return M
8: **else**
9: return "no popular matching"
10: **end if**
11: **if** M is applicant-complete **then**
12: Partition $A \cup P$ as $\mathcal{O}_2$, $\mathcal{E}_2$ and $\mathcal{U}_2$ with respect to M in G'
13: Remove any edge in G' between a node in $\mathcal{O}_2$ and a node in $\mathcal{U}_2$
14: Denote the resulting graph by $G'' = (A \cup P, E'')$
15: **end if**

5.1 Generalizing the Switching Graph

In this section we explain the needed definition of a switching graph for instances, where ties are allowed. A definition of the switching graph for instances, where no ties are allowed, was already given in Section 4. Let G be an instance of POP-M, where ties are allowed, and let M be an arbitrary popular matching on G. Then the switching graph $G_M = (P, E_M)$ is a directed, weighted graph and is defined as follows with respect to M: For every post in G we add a node. The set of edges is defined using the graph G'', as constructed in Step 14 of Algorithm 5.1. Iff for some $a \in A$, $M(a) = p_i$ and $(a, p_j) \in E''$ we add a directed edge from p_i to p_j to the graph, the weight is determined as follows:

$w(p_i, p_j) = 0$ if a is indifferent between p_i and p_j
$w(p_i, p_j) = -1$ if a prefers p_i to p_j
$w(p_i, p_j) = +1$ if a prefers p_j to p_i.

It is easy to see that this graph can be constructed in time $O(\sqrt{n}m)$ using Algorithm 5.1.
Note that some of the results presented in the next paragraphs will be very similar to results proved in Section 4, however we will still show them again since there are some small but important differences.
The following easy lemma characterizes sink vertices in G_M (cf Lemma 4.1):

Theorem 31. *A post p is a sink vertex iff p is unmatched in M.*

Let X be a maximal weakly connected component of G_M. We call X a sink component iff X contains at least one sink, otherwise a non-sink component.
For a path $T = \langle p_1, \ldots, p_k \rangle$ (respectively cycle $C = \langle c_1, \ldots, c_k \rangle$) its weight ($w(T)$ re-

spectively $w(C)$) is the sum of the weights on the edges in T (respectively C).
Here, a switching path is a path in G_M which ends in a sink vertex and has weight 0. Similarly, a switching cycle is a cycle in G_M with weight 0. Let $A_T = \{a_i : M(p_i) = a_i\}$ for $i \in \{1 \dots k\}$ and $A_C = \{a_i : M(p_i) = a_i\}$ for $i \in \{1 \dots k\}$. By $M' = M \cdot T$ we denote the matching, which is created when applying the switching path T to M, that is for $a_i \in A_T$, $M'(a_i) = p_{i+1}$, and $M(a_i) = M(a_i)$ for all $a_i \notin A_T$. Similarly, by $M' = M \cdot C$ we denote the matching which is created when applying the switching cycle C to M, that is for $a_i \in A_T$, $M'(a_i) = p_{i+1} \bmod k$, and $M(a_i) = M(a_i)$ for all $a_i \notin A_C$.

5.1.1 Properties of the Switching Graph

In this section we will prove some useful properties of the switching graph G_M.

Property 32. *([3]) All sink vertices of G_M belong to the set $\mathcal{E}_1$.*

Proof. We know that a popular matching is a maximum matching on M_1. Such a matching matches every node in $\mathcal{O}_1 \cup \mathcal{U}_1$ and since sink vertices are unmatched, they have to be in $\mathcal{E}_1$. □

Property 33. *([3]) Every post p belonging to a sink component has a path to a sink and thus belongs to the set $\mathcal{E}_2$. Every post belonging to a non-sink component belongs to the set $\mathcal{U}_2$.*

Proof. ([3]) We show that a post p belongs to a sink component of G_M iff $p \in \mathcal{E}_2$.
Let p be a post such that $p \in \mathcal{E}_2$, then p is either unmatched or there exists an alternating path with even length from an unmatched node p' to p. If p is unmatched, it is a sink and we are done. Otherwise, let P denote such a path. Note that, since p is even, every edge is of the form $\mathcal{O}_2\mathcal{E}_2$ or $\mathcal{E}_2\mathcal{O}_2$, thus no edge gets deleted in Step 13 of Algorithm 5.1. Thus P is still present in G_M and p belongs to the sink component containing p'.
Now let X be a sink component and let p be in X. If $p \in \mathcal{E}_2$ we are done, since p is even and contains a path to a sink. Otherwise let $p \in \mathcal{U}_2$ (recall that $P \cap \mathcal{O}_2 = \emptyset$). Since p belongs to X, there has to exist some vertex $p' \in \mathcal{E}_2$ such that p' has a path to p, denote a minimal path by $\langle p' = p_1, p_2, \dots, p_k = p \rangle$. Let $M(p_i) = a_i$ for all $i \in \{1, \dots k\}$. Since the path is minimal and $p \in \mathcal{U}_2$, we know that all $p_i \in \mathcal{U}_2, i \in \{2, \dots k\}$. However $p' \in \mathcal{E}_2$ implies $a_1 \in \mathcal{O}_2$ and thus the edge from a_1 to p_2 in G'' would be of the form $\mathcal{O}_2\mathcal{U}_2$. This should have been deleted in Step 13 of Algorithm 5.1 and is a contradiction to $p \in \mathcal{U}_2$.
Because $P \cap \mathcal{O}_2 = \emptyset$, the above immediately implies that p belongs to a non-sink component iff $p \in \mathcal{U}_2$ and finishes the proof. □

Property 34. *([3]) For an edge (p_i, p_j) its weight is determined by the partitions p_i and p_j belong to when partitioning the vertices in $\mathcal{O}_1, \mathcal{E}_1$ and $\mathcal{U}_1$ according to M_1. Let (p_i, p_j) be an edge with $M(p_i) = a$, then its weight is determined as follows:*

p_i/p_j	$\mathcal{O}_1$	$\mathcal{E}_1$	$\mathcal{U}_1$
$\mathcal{O}_1$	*0*	*-1*	-
$\mathcal{E}_1$	*+1*	*0*	-
$\mathcal{U}_1$	-	*-1*	*0*

Proof. ([3]) Note that if a post $p \in \mathcal{O}_1 \cup \mathcal{U}_1$, then it has only rank-1 edges incident in G'. Thus if $p_i \in \mathcal{O}_1 \cup \mathcal{U}_1$, then p_i is a rank-1 post for a. We examine all possible combinations:

- $p_i \in \mathcal{O}_1$: p_i is a rank-1 post for a and since nodes in $\mathcal{O}_1$ get matched with nodes from $\mathcal{E}_1$ this implies that $a \in \mathcal{E}_1$.
  - $p_j \in \mathcal{O}_1$: p_j is also a rank-1 post for a and thus $w(p_i, p_j) = 0$.
  - $p_j \in \mathcal{E}_1$: If p_j would be a rank-1 post for a, an $\mathcal{E}_1\mathcal{E}_1$ edge would exist in G_1, a contradiction to Lemma 8c. Thus p_j cannot be a rank-1 post for a and $w(p_i, p_j) = -1$.
  - Since nodes in $\mathcal{U}_1$ have only rank-1 edges incident to them, p_j would be a rank-1 post for a. That would imply that an $\mathcal{U}_1\mathcal{E}_1$ edge would exist in G_1, a contradiction to Lemma 8c. Thus such an edge cannot exist.
- $p_i \in \mathcal{E}_i$: Here we distinguish two cases:
  (i) p_i is a rank-1 post for a: In this case, note that $s(a) \subseteq f(a)$ and hence a has only rank-1 edges incident to it in G', all ending in posts which belong to $\mathcal{E}_1$. Thus the only possible case is $p_j \in \mathcal{E}_1$ and $w(p_i, p_j) = 0$.
  (ii) p_i is a non-rank-1 post for a: First note that $a \in \mathcal{E}_1$, because posts in $\mathcal{O}_1 \cup \mathcal{U}_1$ get matched along rank-1 edges.
  - $p_j \in \mathcal{O}_1$: p_j is a rank-1 post for a and thus $w(p_i, p_j) = +1$.
  - $p_j \in \mathcal{E}_1$: Assume p_j is a rank-1 post for a. Then in M_1 an $\mathcal{E}_1\mathcal{E}_1$ edge exists, a contradiction to Lemma 8c. Thus p_j is not a rank-1 post for a and $w(p_i, p_j) = 0$.
  - $p_j \in U_1$: Such an edge cannot exist, because then an $\mathcal{E}_1\mathcal{U}_1$ edge would exist in G_1, which is a contradiction to Lemma 8c.
- $p_i \in \mathcal{U}_1$: p_i is a rank-1 post for a, and since nodes in $\mathcal{U}_1$ get matched to nodes in $\mathcal{U}_1$, $a \in \mathcal{U}_1$.
  - $p_j \in \mathcal{O}_1$: p_j is a rank-1 post for a, however $\mathcal{O}_1\mathcal{U}_1$ edges get deleted in Step 4 of Algorithm 5.1, thus such an edge cannot exist.
  - $p_j \in \mathcal{E}_1$: Assume for a contradiction that p_j is a rank-1 post for a. But then an $\mathcal{U}_1\mathcal{E}_1$ edge would exist in the graph G_1, a contradiction to Lemma 8c. Thus

p_j is a non-rank-1 post for a and $w(p_i, p_j) = -1$.

– $p_j \in \mathcal{U}_1$: p_j is a rank-1 post for a and therefore $w(p_i, p_j) = 0$.

□

Property 35. *([3]) Every path T in G_M has weight $w(T) \in \{-1, 0, +1\}$. Every circle C in G_M has weight $w(C) = 0$. Furthermore, there exists no path T in G_M which ends in a sink with weight $w(T) = +1$.*

Proof. ([3]) Using the table above it is easy to see that every path T in G_M has weight $w(T) = \{-1, 0, 1\}$ and every circle C in G_M has weight $w(C) = 0$. First, because there are no edges from nodes in $\mathcal{O}_1 \cup \mathcal{E}_1$ to nodes in $\mathcal{U}_1$, all circles can only contain nodes from $\mathcal{O}_1 \cup \mathcal{E}_1$ or $\mathcal{U}_1$. All edges in circles in $\mathcal{U}_1$ have weight 0, edges in circles in $\mathcal{O}_1 \cup \mathcal{E}_1$ have the alternating weights $+1$ and -1, thus in both cases the weight of the circle is 0. Paths can contain an edge from a node in $\mathcal{U}_1$ to a node in $\mathcal{O}_1 \cup \mathcal{E}_1$ and they do not have to be closed, such possible weights are $-1, 0, 1$.
It remains to show, that there is no path of weight $+1$ ending in a sink vertex. Assume there was such a path T. Consider applying this path to M and examine $M \cdot T$. The number of applicants preferring $M \cdot T$ is one greater than the number of applicants preferring M, contradicting the fact that M is popular. □

Property 36. *([3]) For every switching path T (respectively switching cycle C) in G_M, the matching $M \cdot T$ (respectively $M \cdot C$) is a popular matching.*
Furthermore, every popular matching can be obtained from M by applying one or more vertex disjoint switching paths and switching cycles in each of a subset of sink components of G_M together with applying one ore more vertex disjoint switching cycles in each of a subset of the non-sink components of G_M.

Proof. ([3]) Let $\langle p_1, p_2, \ldots, p_k \rangle$ be a switching path in G_M with p_k unmatched and let $A_T = \cup_{i=1}^{k-1} \{M(p_i) = a_i\}$. Denote by M' the resulting matching $M \cdot T$. Note that in M' every applicant $a \in A_T$ is still matched to $f(a) \cap s(a)$, because only these edges are present in the switching graph, and $M(a) = M(a)\ \forall a \notin A_T$. Furthermore, because $w(T) = 0$, for every applicant that got demoted from his f-post, there exists a unique applicant who got promoted, so M' is popular. A similar arguments proves that $M \cdot C$ is a popular matching, for every switching circle C in G_M.
Let M' be any popular matching in G and consider $M \oplus M'$, this is the set of vertex disjoint path and cycles in G. All paths have even length, since they cannot start or end at applicants (then this applicant was unmatched in one matching), same holds for cycles, as they contain applicants and posts in alternating order. Let $T_G = \langle p_1, a_1, \ldots, p_k, a_k, p_{k+1} \rangle$ be any even length alternating path or cycle in $M \oplus M'$ with p_{k+1} unmatched in M and p_1 unmatched in M'. For every $1 \leq i \leq k$ let $M(p_i) = a_i$. Note that every in M unmatched edge is of the form $\mathcal{O}_2\mathcal{E}_2$ and hence not got deleted in Step 13 of Algorithm

5.1. Thus, this path is present in G_M and ends in a sink. T_G cannot have strictly positive weight since M is a popular matching. Furthermore, for every path with strictly negative weight, another path with strictly positive weight has to exist since both M and M' are popular. But this again contradicts the fact that M is popular and shows that T_G has to have weight 0, ends in a sink and thus is a switching path. A similar arguments shows the same for circles. Thus we have proven that any popular matching can be obtained by applying these paths and cycles. □

Definition 37. *([3]) For an applicant a, let $choices(a)$ be the set of posts p such that (a, p) is an edge in G'' after applying Algorithm 5.1.*

We now define the notation of a tight pair, that is a set of applicants A_1 and a set of posts P_1 with $|A_1| = |P_1|$. Further, for every $a \in A_1$, $choices(a) \subseteq P_1$. We show, that such a tight pair exists in every non-sink component of G_M.

Lemma 38. *([3]) Let Y be a non-sink component in G_M and $q \in Y$. Define $P_q = q \cup \{p : q \text{ has a path to } p \text{ in } G_M\}$. Then there exists a set of applicants A_q such that (i) $|A_q| = |P_q|$, and (ii) for every $a \in A_q$, $choices(a) \subseteq P_q$.*

Proof. ([3]) Let $A_q = \cup_{p \in P_q} M(p)$. Since every $p \in P_q$ is matched, $|A_q| = |P_q|$ follows. For every applicant $a \in A_q$ we have $M(a) \in P_q$ and $M(a) \in choices(a)$. Furthermore, for every $p' \in choices(a) \setminus M(a)$ there exists an edge $(M(a), p')$ in G_M, thus every such p' belongs to P_q and we have $choices(a) \subseteq P_q$. □

5.1.2 Generating Popular Pairs and Counting Popular Matchings

Let $G = (A \cup P, E)$ be an instance of POP-M. Define
$PopPairs = \{(a, p) \in E :$ there exists a popular matching M with $M(a) = p\}$.
It is easy to calculate this set using the switching graph. Let G_M be a switching graph with respect to some popular matching M. From Property 36 we can conclude that $e = (a, p)$ is a popular pair iff (i) $M(a) = p$, (ii) $(M(a), p)$ belongs to some switching path in G_M, or (iii) $(M(a), p)$ belongs to some switching cycle in G_M. The following theorem explains the complexity of checking these conditions.

Theorem 39. *([3]) The set of popular pairs for an instance $G = (A \cup P, E)$ of POP-M where ties are allowed can be computed in time $O(\sqrt{n}m)$.*

Proof. ([3]) For more details about the implementation see [3], here just note that

- Condition (i) can be checked in time $O(\sqrt{n}m)$ using Algorithm 5.1 and
- Condition (ii) and (iii) can be checked using a simple depth-first-search, which takes linear time in the size of G_M.

□

In [3] also the following was proved using a reduction from the problem of counting perfect matchings in 3-regular bipartite graphs:

Theorem 40. *([3]) Given an instance $G = (A \cup P, E)$ of POP-M where ties are allowed, counting the total number of popular matchings in G is #P-Complete.*

5.2 Properties of the Cheating Instance

In this section we will start defining the cheating strategies. For that we will partition the applicants into three sets, depending on which posts they get matched to when being truthful in the instance G:
$A_f = \{a$: every popular matching in G matches a to one of his rank-1 posts$\}$
$A_s = \{a$: every popular matching in G matches a to one of his non-rank-1 posts$\}$
$A_{f/s} = A \setminus (A_f \cup A_s)$
The set A_f denotes applicants who get matched to one of their rank-1 posts in every popular matching, the set A_f applicants who get matched to one of their non-rank-1 posts in every popular matching. The set $A_{f/s}$ contains applicants who belong to neither of the previous partitions, there exist some popular matchings that match them to their rank-1 posts but also some popular matchings that do not match them to their rank-1 posts. This partition can easily be obtained after calculating the set of popular pairs.
Now we will let one applicant a_1 be the cheater, he is aware of the true preferences of all other applicants and can change his preference list, creating an instance H, to get himself better off. Let $L = P_1, P_2, \ldots, P_t, \ldots, P_l$ denote the true preference list of a_1, where P_i denotes his set of i-th ranked posts. For any applicant a we will index his preferences with respect to the desired instance, for example $f_G(a)$ and $s_G(a)$ denotes $f(a)$ and $s(a)$ in the instance G. Note that $f_G(a_1) = P_1$ and assume that $s_G(a_1) \subseteq P_t$, $t > 1$.
Recall the definition of a better always strategy. For any applicant $a \in A_f$, stating his true preferences is his best strategy, therefore in the following we will only consider applicants in A_s and $A_{f/s}$.

- If $a_1 \in A_s$, then in order to get better always there has to exist at least one popular matching in H that matches a_1 to a post higher ranked that t.
- If $a_1 \in A_{f/s}$, then in order to get better always every popular matching in H has to match him to one of his true rank-1 posts.

$s(a)$ for Other Agents Remains Unchanged

Let H be the instance obtained by falsifying the preference list of a_1 alone. Since the rest of the applicants is truthful, we have $f_G(a) = f_H(a)$ for all $a \in A \setminus \{a_1\}$. However, since the definition of s-posts is dependent of other f-posts, changing a's preference list might result in the change of some s-posts. We claim that, if a falsifies his preference list only to improve the rank of the post he gets matched to, this is not the case. We will prove that in Theorem 44, whose proof requires the following lemmas.

Lemma 41. *([3]) Let $a_1 \in A_s \cup A_{f/s}$ when he is truthful. Then, $f_G(a_1) \subseteq (O_1)_G$.*

Proof. ([3]) Assume for a contradiction that there exists $q \in f_G(a_1)$ and $q \subseteq (\mathcal{E}_1 \cup \mathcal{U}_1)_G$. This implies that $a_1 \in (\mathcal{O}_1 \cup \mathcal{U}_1)$. If $a_1 \in (\mathcal{O}_1)_G$, then $s(a_1) \subseteq f(a_1)$, because nodes in $\mathcal{O}_1$ have only edges to nodes in $\mathcal{E}_1$. Thus, a_1 has only rank-1 edges incident and always gets matched to one of his rank-1 posts. If $a_1 \in \mathcal{U}_1$, he also always gets matched to one of his rank-1 posts. This is because nodes in $\mathcal{U}_1$ always get matched to nodes in $\mathcal{U}_1$. A contradiction to $a_1 \in A_s \cup A_{f/s}$. □

Lemma 42. *([3]) Let H be such that $H \succ G$ w.r.t. a_1. Then, $f_H(a_1) \subseteq (\mathcal{O}_1 \cup \mathcal{U}_1)_G$.*

Proof. ([3]) Assume for a contradiction that $f_H(a_1) \cap (\mathcal{E}_1)_G = \{q_1, \ldots, q_k\}$. We show that every popular matching in H matches $f_H(a_1)$ to one of q_i, which is a contradiction because then a_1 never gets matched to a higher ranked post in H than in G.
We first show that, if $f_H(a_1) \cap (\mathcal{E}_1)_G \neq \emptyset$, then the size of a maximum matching on rank-1 edges in H is strictly greater than that of a maximum matching on rank-1 edges in G. Let M_1 be a maximum matching on edges of rank 1 in G which leaves a_1 unmatched. Note that such a matching exists because $f_G(a_1) \subseteq \mathcal{O}_1$ (Lemma 44), which implies that $a_1 \in \mathcal{E}_1$. Consider the graph H_1, which is the graph of rank-1 edges in H. Note that M_1 is a matching in H_1, as no other applicant except a_1 changes his preferences. Then the edge (q_i, a_i) represents an augmenting path, giving a maximum matching M_2 of size $M_1 + 1$.
Now consider a popular matching M' in H and let M'_1 denote the matching restricted to edges of rank 1. Let us consider all possible cases how a_1 can be matched to get the desired contradiction:

- $M'(a_1) \in \{q_1, \ldots, q_k\}$: In this case we are directly done.
- $M'(a_1) = q$ with $q \in f_H(a_1) \cap (\mathcal{O}_1 \cup \mathcal{U}_1)_G$: Assume $q \in f_G(a_1)$. In this case the edge $(a_1, q) \in G_1$ and M'_1 is a maximum matching in G_1. But $|M'_1| = |M_1 + 1|$ contradicts the fact that M_1 is a maximum matching in G_1. Thus assume $q \notin f_G(a_1)$. Note that $M'_1 \setminus \{(a_1, q)\}$ is a maximum matching in G_1 since no other applicants changed their preference lists. However, M'_1 leaves a_1 unmatched, which is a contradiction

to $q \in (\mathcal{O}_1 \cup \mathcal{U}_1)_G$, since posts from $(\mathcal{O}_1 \cup \mathcal{U}_1)_G$ get matched in any maximum matching of G_1.

- $M'(a_1) \in s_H(a_1)$: If $s_H(a_1) \subseteq f_H(a_1)$, this is covered in the previous cases. Otherwise assume $s_H(a_1) \cap f_H(a_1) = \emptyset$. This implies, that a_1 is unmatched in M'_1. Because no other applicants changed their preferences, M'_1 is also a matching in G_1. But $|M'_1| = |M + 1|$ contradicts the fact that M_1 is a maximum matching in G_1.

□

Lemma 43. *([3]) Let M_1 be a maximum matching in G_1, such that M_1 leaves a_1 unmatched. Then in any instance with $H \succ G$ w.r.t. a_1, M_1 is a maximum matching in H_1.*

Proof. ([3]) Like in the proof of the previous lemma, note that a matching M_1 in G_1 that leaves a_1 unmatched exists because $f_G(a_1) \subseteq \mathcal{O}_1$ (Lemma 41), which implies that $a_1 \in \mathcal{E}_1$. Assume M_1 is not a maximum matching in H_1, then there exists an augmenting path $\langle a_1, p_1, \ldots, a_k, p_k \rangle$ in H_1 with respect to M_1, such that a_1 and p_k are unmatched. However this path is also an alternating path of even length from p_k to p_1, contradicting the fact that $p_1 \in (\mathcal{O}_1 \cup \mathcal{U}_1)_G$. □

Theorem 44. *([3]) Let H be an instance such that $H \succ G$ w.r.t a_1. Then, (i) $(\mathcal{E}_1)_G \cap P = (\mathcal{E}_1)_H \cap P$ and therefore $s_H(a) = s_G(a)$ for every $a \in A \setminus \{a_1\}$, and (ii) $(\mathcal{O}_1)_G \cap A = (\mathcal{O}_1)_H \cap A$.*

Proof. ([3]) If $f_H(a_1) = f_G(a_1)$, $H_1 = G_1$ and (i) and (ii) are trivially true. Thus consider the case that $f_H(a_1) \neq f_G(a_1)$ and let M_1 be a maximum matching in G_1 that leaves a_1 unmatched. By Lemma 43 we know that it is also a maximum matching in H_1.
We first prove $(\mathcal{E}_1)_G \cap P = (\mathcal{E}_1)_H \cap P$, consider for this the following two cases:

- $p \in (\mathcal{O}_1 \cup \mathcal{U}_1)_G \cap P$: Assume $p \in (\mathcal{E}_1)_H \cap P$. This implies, that there exists an even length alternating path T with respect to M_1 in H_1 from an unmatched node to p. The path cannot contain a_1, since a_1 is unmatched in M_1. Thus, T is also present in G_1, which contradicts the fact that $p \in (\mathcal{O}_1 \cup \mathcal{U}_1)_G$.

- $p \in (\mathcal{E}_1)_G \cap P$: There exists an even length alternating path T with respect to M_1 in G_1 from an unmatched node to p, which does not contain a_1, since a_1 is unmatched in M_1. Thus, T also exists in H_1, which proves that $p \in (\mathcal{E}_1)_H \cap P$.

Since all other applicants leave their preferences unchanged, it is clear that $s_G(a) = s_H(a)$ $\forall a \in A \setminus \{a_1\}$.
Now consider any $a \in (\mathcal{O}_1)_G \cap A$. Assume for a contradiction that $a \in (\mathcal{E}_1)_H \cap A$. Then there exists an even length alternating path T w.r.t. M_1 from an unmatched node in H_1 to a. T has to contain a_1, otherwise it was already present in G_1. Because a_1 is

unmatched, T has to start at a_1. But since a is odd in G_1, the post adjacent to a_1 in T, say p', would be even. Since $p' \in f_H(a_1)$, this is a contradiction to Lemma 42. □

An A_s Applicant Cannot Get One of Her True Rank-1 Posts

In this section we will prove, that, although an applicant from A_s is able to improve the rank of the post matched to him, he is never able get his true rank-1 post. This is shown in Theorem 48, for which the following lemmas are needed.

Lemma 45. *([3]) Let $a_1 \in A_s$ and let $q \in f_G(a_1)$. Then, q belongs to a non-sink component of G_M and the edge $(M(a_1), q)$ is not contained in a cycle in G_M.*

Proof. ([3]) We first show that $(M(a_1), q)$ is not contained in any cycle in G_M. Assume it was in some cycle C. Since every cycle has weight 0, every cycle is a switching circle. When examining the matching $M \cdot C$, we find another popular matching where a_1 is matched to one of his rank-1 posts, a contradiction to $a_1 \in A_s$.
Now we show that q does not belong to a sink component. Assume not, then q belongs to some sink component X of G_M. By Property 33 we know that there exists a path T from q to a sink q' in X. Furthermore, by Lemma 41 we know that $q \in (\mathcal{O}_1)_G$ and by Property 32 that $q' \in (\mathcal{E}_1)_G$. From the table in Property 34 we can see, that any path from a node in $\mathcal{O}_1$ to a node in $\mathcal{E}_1$ with alternating nodes in between has weight -1. Furthermore, $w(M(a_1), q) = +1$, because $M(a_1) \in s_G(a_1)$ and $q \in f_G(a_1)$. Thus, $(M(a_1), q)$ connected with T forms a switching path, since it ends in a sink and has weight 0, and applying this path gives a popular matching where $M(a_1) \in f_G(a_1)$. This is a contradiction to $a_1 \in A_s$. □

Lemma 46. *([3]) Let $a_1 \in A_s$ and let $q \in f_G(a_1)$. Let $P_q = q \cup \{q' :$ there is a path from q to p' in $G_M\}$ and $A_q = \cup_{q' \in P_q} M(q')$. Then, $a_1 \notin A_q$.*

Proof. ([3]) Since $a_1 \in A_s$, by Lemma 45 q belongs to a non-sink component, say Y, of G_M. If $M(a_1) \notin Y$ then $a_1 \notin A_q$ is obvious. Otherwise, let $M(a_1) \in Y$. Assume for a contradiction that there exists a path from q to $M(a_1)$ in Y. Since $M(a_1)$ belongs to a non-sink component of G_M, $M(a_1) \in (\mathcal{U}_2)_G$, which implies $a_1 \in (\mathcal{U}_2)_G$. Further, by Lemma 41 $q \in \mathcal{O}_1$ since $q \in f_G(a_1)$ and thus $a_1 \in (\mathcal{E}_1)_G$. Then according to Claim 30 the edge $(M(a_1), q)$ does not get deleted in either Step 4 or Step 13 of Algorithm 5.1. This implies, that the edge $(M(a_1), q)$ is contained in a circle in G_M, a contradiction to Lemma 45. □

Lemma 47. *([3]) Let $a_1 \in A_s$ and $q \in f_G(a_1)$. Then there exist sets A_q and P_q with $|A_q| = |P_q|$, such that for every $a \in A_q$ $choices_H(a) \in P_q$.*

Proof. ([3]) Since $a_1 \in A_s$ and $q \in f_G(a_1)$, by Lemma 45 we know that q belongs to a non-sink component of G_M, say Y. Thus we can use the definition from Lemma 38 to

construct A_q and P_q with $|A_q| = |P_q|$ and for each $a \in A_q$ $choices_G(a) \in P_q$. Thus, to prove the lemma we just have to show $choices_H(a) \subseteq choices_G(a)$.
By Lemma 46 $a_1 \notin A_q$, and thus $f_G(a) \cup s_G(a) = f_H(a) \cup s_H(a)$ for all $a \in A_q$. Different edges though could be deleted in Step 4 and Step 13 of Algorithm 5.1 when run on G and H, which we have to check.
Since every $a \in A_q$ belongs to a non-sink component due to Lemma 45, $a \in \mathcal{U}_2$. By Claim 30 we know that in Step 13 of Algorithm 5.1 no edges incident to applicants in $\mathcal{U}_2$ get deleted.
It remains to show that, if an edge (a', p') gets deleted in Step 4 of Algorithm 5.1 when run on instance H, then it also gets deleted in this step when run on instance G, to uphold $choices_H(a) \subseteq choices_G(a)$. Lemma 41 implies, that $q \in (\mathcal{O}_1)_G$. From the table in Property 34 we can see that no post in $(\mathcal{U}_1)_G$ can be reached from a node in $(\mathcal{O}_1)_G$, so for any $q \in P_q$ $q \in (\mathcal{O}_1)_G \cup (\mathcal{E}_1)_G$. Furthermore, also for every $a \in A_q$ $a \in (\mathcal{O}_1)_G \cup (\mathcal{E}_1)_G$, because posts in $(\mathcal{O}_1)_G \cup (\mathcal{E}_1)_G$ get matched to applicants in $(\mathcal{O}_1)_G \cup (\mathcal{E}_1)_G$. If $a' \in (\mathcal{E}_1)_G$, by Claim 30 no edge incident to a' gets deleted in Step 4 of Algorithm 5.1. If $a' \in (\mathcal{O}_1)_G$ and (a', p') gets deleted in Step 4, then $q' \in (\mathcal{O}_1)_G \cup (\mathcal{U}_1)_G$. Then, by Theorem 44 also $a' \in (\mathcal{O}_1)_H$ and $p' \in (\mathcal{O}_1)_H \cup (\mathcal{U}_1)_H$ and the edge also gets deleted in Step 4 as well when run on instance H. □

Theorem 48. *([3]) Let $a_1 \in A_s$. Then, by falsifying his preferences, a_1 cannot get matched to a post in $f_G(a_1)$ in any popular matching in the falsified instance H.*

Proof. ([3]) For a contradiction assume there exists a popular matching M' in H such that a_1 gets matched to a post q in $f_G(a_1)$. By Lemma 45 we know that q belongs to a non-sink component. By Lemma 47 we know that there exist sets of applicants and posts, A_q and P_q, with $|A_q| = |P_q|$ and $choices_H(a) \in P_q$ for all $a \in A_q$, but $a_1 \notin A_q$. Thus if a_1 gets matched to q, one applicant a from A_q cannot get matched to $choices(a)$, a contradiction to the fact that M' is a popular matching. □

5.3 The Modified Instance $\tilde{G}$

In this section we will describe a modified instance $\tilde{G}$, which is used to find a_1's cheating strategies. One reason we do this is that some rank-1 edges, which got deleted in Algorithm 5.1, can be used in falsified instances. Thus when creating $\tilde{G}$, we try to imitate H's characteristics regarding edge deletion, of course while still allowing the popular matchings available in G. More precisely, the instance $\tilde{G}$ created out of G should have the following properties: (i) Every popular matching in G is a popular matching in $\tilde{G}$, and (ii), any edge (a, p) that gets deleted in Step 4 of Algorithm 5.1, when run on instance $\tilde{G}$, also gets deleted in Step 4 of Algorithm 5.1, when run on an instance H, with $H \succ G$ w.r.t. to some applicant a_1. However, the definition of $\tilde{G}$ is independent of a_1.
The graph $\tilde{G}$ is defined as follows: Let $\{q_1, \ldots, q_k\}$ be the posts in $\mathcal{U}_1$. Add a dummy

applicant b to the instance G, who has all these unreachable posts tied as his rank-1 preference. Formally, $\tilde{G} = (\tilde{A} \cup P, \tilde{E})$, with $\tilde{A} = A \cup \{b\}$, $\tilde{E} = E \cup \{(b, q_1), \ldots, (b, q_k)\}$, and each of these edges is a rank-1 edge.
Note that $f_{\tilde{G}} = \{q_1, \ldots, q_k\}$ and $s_{\tilde{G}} = l(b)$. Since all uneven nodes are matched in G_1, every maximum matching in G_1 is also a maximum matching in $\tilde{G}_1$, with the difference that in $\tilde{G}_1$ every node is odd or even.
First we show that this change does not affect the s-posts of any applicant.

Lemma 49. *([3]) For every $a \in A$, $s_G(a) = s_{\tilde{G}}(a)$.*

Proof. ([3]) Let M_1 be any maximum matching in G_1. Since M_1 is also a maximum matching in $\tilde{G}_1$, examine the partition given by this. The only difference caused by the addition of the unmatched node b is that every formerly unreachable post is now odd. Thus $(\mathcal{E}_1)_G \cap P = (\mathcal{E}_1)_{\tilde{G}}$ and the set $s(a)$ is unchanged for every $a \in A$. □

Now let M be a popular matching in G and let $\tilde{M}$ be the corresponding matching in $\tilde{G}$, where $M(a) = \tilde{M}(a)$ for every $a \in A$ and $\tilde{M}(b) = l(b)$. Since $\tilde{M}_1$ is a maximum matching in $\tilde{G}_1$ and every applicant is either matched to his f- or s-post, $\tilde{M}$ is a popular matching.

Lemma 50. *([3]) Let (a, p) be an edge which gets deleted in step 4 of Algorithm 5.1 when run on $\tilde{G}$. Then this edge also gets deleted in Step 4 when run on any instance H such that $H \succ G$ w.r.t a_1.*

Proof. ([3]) As mentioned before, nodes in $\tilde{G}_1$ are either even or odd, so if an edge got deleted in Step 4 of Algorithm 5.1, then it is of the form $\mathcal{O}_1\mathcal{O}_1$. To prove the lemma, it thus suffices to show that any odd node in $\tilde{G}_1$ stays odd in H_1.
Let $a \in A$ such that $a \in (\mathcal{O}_1)_{\tilde{G}}$. This means, that there exists an alternating path T_1 of odd length starting at an unmatched post p in $\tilde{G}_1$ to a. T_1 cannot contain a_1, because a_1 is unmatched in $\tilde{G}_1$. For a contradiction assume that $a \in (\mathcal{E}_1)_H$, so there exists an alternating path T_2 of even length starting at an unmatched applicant p in H_1. T_2 has to start at a_1, otherwise it was already present int $\tilde{G}_1$. But then, joining T_1 and T_2 gives an augmenting path in M_1, contradicting the maximality of M_1 in H_1 (that M_1 is a maximum matching was shown in Lemma 43).
Now consider $p \in P$ such that $p \in (O_1)_{\tilde{G}}$ and denote with T_1 the odd length alternating path starting at an unmatched applicant. T_1 cannot contain a_1 as not an endpoint because a_1 is unmatched, and has to start at a_1, otherwise it was also present in H_1. Now for a contradiction assume that $p \in (\mathcal{E}_1)_H$. Denote with T_2 the even length alternating path with respect to M_1 starting at an unmatched post in H_1. T_2 cannot contain a_1 because a_1 is unmatched, and joining T_1 and T_2 again gives an augmenting path in $\tilde{G}_1$, contradicting the maximality of M_1 in $\tilde{G}_1$. □

Lemma 51. *([3]) Let $a \in A \setminus \{a_1\}$, such that a belongs to a non-sink component $\tilde{G}_M$. Let H be an instance with $H \succ G$ w.r.t. a_1. Then $choices_H(a) \subseteq choices_{\tilde{G}}(a)$.*

Proof. ([3]) Since all $a \in A \setminus \{a_1\}$ keep their preferences, the only way to falsify $choices_H(a) \subseteq choices_{\tilde{G}}(a)$ is to delete some edges in $\tilde{G}$. Since a belongs to a non-sink component, both a and $M(a)$ belong to the set $(\mathcal{U}_2)_{\tilde{G}}$. Thus, by Claim 30, no edge adjacent to a gets deleted in Step 13 of Algorithm 5.1. Furthermore, according to Lemma 50, the same edges get deleted in Step 4 when run on H and $\tilde{G}$, thus we have $choices_H(a) \subseteq choices_{\tilde{G}}(a)$. □

5.4 Definition of the Strategies for Single Manipulative Applicants

In this section we will use the modified instance $\tilde{G}$ to formulate cheating strategies for a single applicant a_1 in A_s and $A_{f/s}$. The true preference list of a_1 is denoted by $L = P_1, \ldots, P_t, \ldots, P_l$, where P_i denotes a_1's set of i-th ranked posts and $s_G(a_1) \subseteq P_t$.

A_s Applicant

Let $a_1 \in A_s$ and let M be any popular matching in G, denote by $\tilde{M}$ the corresponding popular matching in $\tilde{G}$ which matches b to $l(b)$. By the definition of A_s we know that $\tilde{M}(a_1) \in s_G(a_1)$ and thus $M(a_1) \in P_t$. We use the switching graph to check whether a_1 can always be better off. Let L_f denote a_1's falsified preference list. The strategy is reflected in Algorithm 5.2

Algorithm 5.2 Cheating Strategy for an A_s Applicant

1. For $i = 2 \ldots t-1$ check if there exists a post $p \in P_i$ in a_1's preference list such that
(a) p belongs to a sink component of $\tilde{G}_M$ or,
(b) p has a path to $\tilde{M}(a_1)$ in $\tilde{G}_M$
2. If no post satisfies (a) or (b), then a_1's true preference list is optimal
3. Else let p be a_1's most preferred post satisfying (a) or (b). Set p as a_1's first ranked post
4. Let a_2 be some applicant with $\tilde{M}(a_2) \in f_G(a_1)$ and $p' \in s_G(a_2)$. Set p' as a_1's second ranked post
5. $L_f = p, p'$

Theorem 52. *([3]) Let $a \in A_s$. Then there exists a cheating strategy for a which makes him always better off iff there exists some post p in his preference list ranked $2 \ldots t-1$ such that either*
(a) p belongs to a sink component in $\tilde{G}_{\tilde{M}}$ or
(b) p has a path to $\tilde{M}(a_1)$ in $\tilde{G}_{\tilde{M}}$.

Proof. ([3]) We first prove that, if there exists a post which satisfies either of the above conditions, then in every popular matching in H a_1 gets matched to p and thus is better

off than when listing his true preferences.
For a contradiction assume there exists a popular matching M'' in H such that $M''(a_1) = p'$. Note that $\tilde{M}$ is a popular matching in $\tilde{G}$, which is constructed from a popular matching M in G. Let $\tilde{M}(a_2) = M(a_2) = q$. Since $q \in f_G(a_1)$, by Lemma 45 q belongs to a non-sink component. By Lemma 47 there exist sets A_q and P_q with $|A_q| = |P_q|$, $a_1 \notin A_q$ and for every $a \in A_q$, $choices(a) \subseteq P_q$.
We show that p' also belongs to P_q and thus, if M'' matches a_1 tp p', at least one applicant a in A_q cannot get matched to a post in $choices(a)$.
If G_M contains the edge (q, p'), then we are done. Since $M(a_2) = q$ and $p' \in s(a_2)$, the edge (q, p') can only be absent in G_M if the edge (a_2, p') got deleted. In Step 4 of Algorithm 5.1 it cannot get deleted, as here only rank-1 edges get deleted. Since q belongs to a non-sink component of G_M, $q \in (\mathcal{U}_2)_G$. Then also $M(q) = a_2 \in (\mathcal{U}_2)_G$, thus the edge (a_2, p') does not get deleted in Step 13 of Algorithm 5.1 either.
Finally we prove, that if our strategy does not find a post matching either condition, there is no strategy which makes a_1 always better off. More precisely, if no matching post is found, there exists no instance H, in which there exists a popular matching which matches a_1 to a post with rank less than t.
For a contradiction assume there exists an instance H in which a_1 listed fake preferences. Let M' be some popular matching of H, such that $M'(a_1) = q$ and a_1 strictly prefers q to his t-th ranked post. Since our strategy did not find q, it belongs to a non-sink component Y of $\tilde{G}_{\tilde{M}}$ and there exists no path from q to $\tilde{M}(a_1)$ in $\tilde{G}_{\tilde{M}}$. Consider the sets A_q and P_q as defined in Lemma 38. We know $|A_q| = |P_q|$ and for every $a \in A_q$ we have $choices(a) \subseteq P_q$. Note that $a_1 \notin A_q$, as then $\tilde{M}(a_1) \in P_q$ and thus there was a path from q to $\tilde{M}(a_1)$, a contradiction. Also see that $l(b) \notin P_q$ and thus $b \notin A_q$. That means, for every $a \in A_q$ we have $choices_H(a) \subseteq P_q$. Therefore, if M' matches a_1 to q, there exists at least one applicant in A_q who cannot get matched to one of his posts in $choices$, a contradiction to the fact that M' is a popular matching. □

$A_{f/s}$ Applicant

Now let us formulate a strategy for an applicant a_1 in $A_{f/s}$, with which he can improve the rank of the post matched to him, if possible. Let $\tilde{M}$ be a popular matching that matches a_1 to one of his rank-1 posts and denote by $\tilde{M}$ the corresponding popular matching in $\tilde{G}$ which matches b to $l(b)$. Again we use the switching graph to define the strategy, as shown in Algorithm 5.3.

Theorem 53. *([3]) Let $a_1 \in A_{f/s}$. There exists a cheating strategy for a_1 to get always better off iff there exists some post p' in $(\mathcal{E}_1)_G$ which satisfies the following properties:*
(a) p' belongs to a non-sink component Y of $\tilde{G}_{\tilde{M}}$ and,
(b) p' does not have a path T to $\tilde{M}(a_1)$ in $\tilde{G}_M$ with $w(T) = +1$.

Proof. ([3]) Like before, we first show that, if such a post exists, the strategy makes a_1 indeed always better off and then, if no such post exists, that his true preferences are

Algorithm 5.3 Cheating Strategy for an $A_{f/s}$ Applicant

1. For every $p' \in (\mathcal{E}_1)_G \cap P$ check if
(a) p' belongs to a non-sink component Y of $\tilde{G}_M$ and,
(b) p' does not have a path T to $\tilde{M}(a_1)$ in $\tilde{G}_M$ with $w(T) = +1$
2. If no post satisfies both properties, then a_1's true preference list is optimal
3. Else set $M(a_1) = p$ as the rank-1 post of a_1 and p' as the rank-2 post in a_1's falsified preference list
4. $L_f = p, p'$

optimal.
Suppose there exists a post p' satisfying both properties and let H be the resulting instance when a_1 changes his preferences according to the above cheating strategy. We will show that every popular matching in H matches a_1 to p. $M = \tilde{M} \setminus \{b, \tilde{M}(b)\}$ is a popular matching in H, because only a_1 changes his preferences, but his assigned post is still in $choices(a_1)$ and is an f-post. Let us now show that every popular matching in H matches a_1 to p. Assume for a contradiction that there exists a matching M' with $M'(a_1) = p'$.
We know that p' belongs to a non-sink component Y of $\tilde{G}_M$ and p' does not have a path T to $\tilde{M}(a_1)$ in $\tilde{G}_{\tilde{M}}$. Define $A_{p'}$ and $P_{p'}$ as in Lemma 38, thus $|A_{p'}| = |P_{p'}|$ and for every $a \in A_{p'}$ $choices(a) \subseteq P_{p'}$. Since p' does not have a path to $\tilde{M}(a_1)$, $\tilde{M}(a_1)' \notin P_{p'}$ and thus $a_1 \notin A_{p'}$. Further, $\tilde{M}(b) \notin P_{p'}$ since $\tilde{M}(b) = l(b)$ does not have any incoming edges. Thus, for every $a \in A_{p'}$, $choices(a) \subseteq P_{p'}$. But if $M'(a_1) = p'$, then at least one applicant a from $A_{p'}$ cannot get matched to $choices(a)$, a contradiction to the fact that M' is a popular matching in H.
Now we prove that, if no post exists which satisfies both properties, a_1's true preferences are optimal. For a contradiction assume there is no post satisfying both properties from Theorem 53, but there exists an instance H in which in every popular matching a_1 gets matched to a post in $f_G(a_1)$. Let $q' \in s_H(a_1)$. First note that $q' \in f_G(a_1)$, as otherwise $q' \in (\mathcal{O}_1)_G$ due to Lemma 41 and thus q' could be no s-post. Since the algorithm did not find q', q' either belongs to a sink component of $\tilde{G}_{\tilde{M}}$ or it belongs to a non-sink component of $\tilde{G}_M$ and has a path of strictly positive weight to $\tilde{M}(a_1)$. In both cases we will construct a popular matching which matches a_1 to q'.

- q' belongs to a sink component Y of $\tilde{G}_{\tilde{M}}$. Thus there exists a path T starting in q' and ending in a sink (Property 33). Because $q' \in s_H(a_1)$, $q \in (\mathcal{E}_1)_H \cap P = (\mathcal{E}_1)_{\tilde{G}} \cap P$, whereat the last equation follows from Theorem 44. Since the end point of T is a sink vertex in $\tilde{G}_{\tilde{M}}$, which therefore also belongs to $(\mathcal{E}_1)_{\tilde{G}} \cap P$, we can see from the table in Property 34 that $w(T) = 0$. Thus T is a switching path and we consider the matching $M_1 = \tilde{M} \cdot T$, which leaves q' unmatched. Let $M_2 = M_1 \setminus \{(a_1, \tilde{M}(a_1))\}$ and $M' = \{(a_1, q'), (a_2, \tilde{M}(a_)\} \cup M_2$. Assume now that $\tilde{M}(a_1) \notin Y$ (i), or if yes, then q' does not have a path to $\tilde{M}(a_1)$ (ii). The case when q' does have a path to $\tilde{M}(a_1)$ (iii) is handled in the next case. Let $A_T = \cup_{p \in T}\{\tilde{M}(p)\}$. In either case,

(i) or (ii), $a_1 \notin A_T$. Also, $b \notin A_T$, since $\tilde{M}(b) = l(b)$ does not have any incoming edges. That means, for any $a \in A_T$ we have $M(a) \in \{f_{\tilde{G}}(a) \cup s_{\tilde{G}}(a)\}$ and thus $M(a) \in \{f_H(a) \cup s_H(a)\}$ since their preferences remain unchanged. Furthermore, $M'(a_2) \in f_{\tilde{G}}(a_2) = f_H(a_2)$ and $M'(a_1) \in s_H(a_1)$. Thus for every a $M(a) \in \{f_H(a) \cup s_H(a)\}$. Further, since $w(T) = 0$ and a_1 prefers M' but a_2 $\tilde{M}$, the number of rank-1 edges in M' is the same as the number of rank-1 edges in $\tilde{M}$. Thus, M' is in fact a popular matching and we have the desired contradiction.

- q' belongs to a non-sink component Y of $\tilde{G}_M$: Then q' has a directed path T to $\tilde{M}(a_1)$ with $w(T) = +1$, otherwise the above algorithm would have found q'. The case (iii) from before is also covered here, let also T be the path from q' to $\tilde{M}(a_1)$. Since $q' \in (\mathcal{E}_1)_{\tilde{G}}$ and $\tilde{M}(a_1) \in (\mathcal{O}_1)_{\tilde{G}}$, we see from the table in Property 34 that also $w(T) = +1$.
  Let $M_1 = \tilde{M} \setminus \{(a_1, \tilde{M}(a_1)), (b, \tilde{M}(b))\}$, which leaves $\tilde{M}(a_1)$ unmatched. Let $M_2 = M \cdot T$ and $M' = M_2 \cup (a_1, q')$. Like before we can show that for every a $M(a) \in \{f_H(a) \cup s_H(a)\}$. Further, since $w(T) = +1$ and a_1 is no longer matched to one of his rank-1 posts, the number of rank-1 edges in M' is the same as the number of rank-1 edges in $\tilde{M}$. Therefore we have a popular matching M' in H with $M'(a_1) \in s_H(a_1)$, the desired contradiction.

□

Theorem 54. *([3]) The optimal falsified preference list for a single manipulative applicant to get better always can be calculated in time $O(\sqrt{n}m)$ if the preference lists contain ties, and in time $O(n + m)$ if not.*

Proof. ([3]) Like seen before, to calculate the optimal falsified preference list for an applicant we have to (i) compute the set of popular pairs, (ii) construct the switching graph and (iii) run one of the defined strategy algorithms. (iii) can be done in linear time with respect to the switching graph. In Theorem 39 we have shown, that (i) and (ii) can be done in time $O(\sqrt{n}m)$ in the case of ties. In the absence of ties we can reduce this to time $O(n + m)$ using Algorithm 3.1. □

6 An Alternative Characterization of Popular Matchings

In this section we want to give a different characterization of popular matchings than the one obtained in Section 3. The idea first came up when trying to count all instances that allow popular matchings and when trying to define the popular matching polytope. Both of this did not work, however, this characterization might be interesting for some other applications.
For any subsets A_1, A_2 of the applicants we define $B_{A_2}(A_1) = \{p \in P \mid \exists a \in A_1, f(a) = p \vee s(a) = p\}$, when respecting all applicants in A_2, so it is the reduced subgraph of A_1 with respect to A_2. Denote by $F(A_1)$ and $S(A_1)$ the sets of f- and s-posts for all applicants in A_1. With this an characterization of popular matchings is immediate:

Lemma 55. *An instance does not allow a popular matching $\Leftrightarrow$ there exists a subset A_1 of the applicants, $|B_A(A_1)| < |A_1|$.*

Proof. An instance does not allow a popular matching $\Leftarrow$ for some subset A_1 of the applicants, $|B_A(A_1)| < |A_1|$: If such a subset exists, i applicants must be matched to j posts with $j < i$, which is not possible.
If an instance does not allow a popular matching, some subset has to exist with A_1 $|B_A(A_1)| \leq |A_1|$, otherwise there would be a matching in the reduced graph. □

We can refine this characterization where we consider only sets of applicants with some kind of independence:

Lemma 56. *An instance does not allow a popular matching $\Leftrightarrow$ there exists a subset A_1 of the applicants, $|B_A(A_1)| < |A_1| \wedge B_A(A \setminus A_1) \cap B_A(A_1) = \emptyset$*

Proof. An instance does not allow a popular matching $\Leftarrow$ for some subset A_1 of the applicants, $|B_A(A_1)| < |A_1| \wedge B_A(A \setminus A_1) \cap B(A_1) = \emptyset$: If such a subset exists, i applicants must be matched to j posts with $j < i$, which is not possible. Furthermore, when examining the complete instance, the $B_A(A_1)$ does not change, thus there exists no popular matching.
An instance does not allow a popular matching $\Rightarrow$ for some subset A_1 of the applicants, $|B_A(A_1)| < |A_1| \wedge B_A(A \setminus A_1) \cap B_A(A_1) = \emptyset$. If an instance does not allow a popular matching, we know that for some subset A_1 $|B_A(A_1)| \leq |A_1|$. Trivially, $F(A \setminus A_1) \cap S(A_1) = \emptyset$. If not yet $B_A(A \setminus A_1) \cap B(A_1) = \emptyset$ holds, we can itaratively expand A_1 until the condition holds. Since $B_A(A \setminus A_1) \cap B(A_1) \neq \emptyset$, there exists one applicant a for which either one of the following is true: 1. $f(a) \in F(A_1)$ 2. $s(a) \in S(A_1)$ 3. $f(a) \in F(A_1) \wedge s(a) \in S(A_1)$. In either case, include a in A_1. If 1 or 2 holds, $B_A(A_1)$ grows by 1, $|A_1|$ also. So $|B_A(A_1)| < |A_1|$ still holds. If 3 is true, $B_A(A_1)$ grows by 0, $|A_1|$ by 1, so $|B_A(A_1)| < |A_1|$ still holds. Since in every step we include an applicant to A_1 and A is finite, we at some stage reach a set with the desired properties. □

7 Counting Instances that Allow Popular Matchings

In this section we try to count the exact number of instances allowing a popular matching using a closed combinatoric formula. The formula could not be completed here, however still results achieved so far shall be listed here, because by this principle the number of groups of applicants with a specific size, who prohibit a popular matching, can be counted, and using combinatorics to count popular matchings itself is a very interesting thing. The formula was not finished because preventing double counting is not easy, and was not studied more intensively, since the formula takes long to compute anyway and the exact number of popular matchings can be approximated very well using measures, see Section 7.3.

7.1 Preliminaries

We first define and explain some sub functions needed for the complete calculation. In this section we will actually, contrary to the notion in some other sections, denote by n the number of applicants and by m the number of posts. For the basic case, only the case that $|n| = |m|$ is considered under the assumption that each applicant has a strictly ordered preference list of length $|m|$.

$EVEN(a) = 1$ if a odd, -1 else

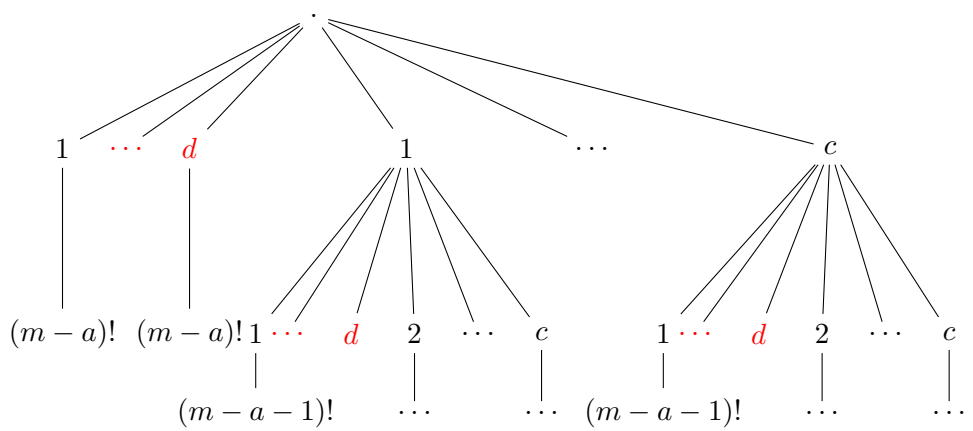
$SAME(i_1, \ldots, i_k) =$ factorial for each number of distinct values in $i_1, \ldots, i_k$, e.g. $SAME(1, 1, 2, 3, 3) = 2! * 1! * 2!$

$DIFFPOST(a, b, c) = \frac{(m-a)!}{((m-a-b)!)} * \sum_{i_1>=1,\ldots i_b>=1, i_1+\ldots+i_b=n-c, i_1 \leq i_2 \ldots \leq i_b} \frac{1}{SAME(i_1,\ldots,i_b)} * \frac{(n-c)!}{i_1!*\ldots*i_b!}$
We use this function to assign exactly b different posts to $n-c$ positions, for this $m-a$ posts can be used.
$\frac{(m-a)!}{((m-a-b)!)}$ describes the number of possibilities to chose b different values out of $m-a$ possibilities. The order is important here, as we will think of them as the value with the lessest occurrences, the value with the second lessest occurrences etc. If $n-c > b$, the b chosen values will be used more than once. The sum is used to iterate over all possible combinations of occurrences. Value i_1 has to occur at least once, since we assume the values to be ordered, i_2 has to occur at least i_1 times and so on, the sum of all i_k has to equal the number of desired positions, $n-c$. If multiple values have the same number of occurrences, the order becomes irrelevant for these, we prevent double counting by dividing through $SAME(i_1, \ldots, i_k)$. Finally $\frac{(n-c)!}{i_1!*\ldots*i_b!}$ denotes the number of permutations of $n-c$ elements, if the elements occur respectively i_k times.

$DP(a, b) = \frac{(m-a)!}{(m-a-b)!*b!}$
This function describes the number of possibilities to choose b different elements out of

$m - a$.
$\frac{(m-a)!}{(m-a-b)!}$ denotes the number of possibilities to chose b elements, we divide by $b!$ since we do not care for the order.

$ERPOST(a, b, c, d) = (1 * (m - a)! + c(1 + (m - a - 2)! + (c - 1)[\ldots(c - 2)(1 + 1 * (m - a - c)!)]))^b * DIFFPOST(m - d, d, n - b)$
This function describes the number of possibilities, to chose the 2-nd till m-th priority of an applicant, if exactly d s - posts exist and up to c other posts can be ranked higher than his s - post.
With $DIFFPOST(m - d, d, n - b)$ we choose these d s - posts, we draw here from $m - (m - d) = d$ posts and repeat the process for $n - (n - b) = b$ applicants. Choosing exactly d posts from d possibilities maybe seems unintuitive, but this way we assign a priori every applicant some s - post out of the preselected d ones, making sure every post of the d is used.



The tree above depicts, how the 2nd till m-th priorities of any applicant may look like. As his second priority, on the one hand one the d s posts may directly be chosen (left side). Then for the remaining priorities, there are $(m - a)!$ priorities. On the other hand, the second priority can be any of the other acceptable c posts (right side). For the third priority this is repeated, and so an. But since we have already chosen beforehand via the function $DIFFPOST$, which applicant gets which s post, instead of d possibilities there remains only one, in the tree the red nodes are deleted. The inner part of $(1 * (m - a)! + c(1 + (m - a - 2)! + (c - 1)[\ldots(c - 2)(1 + 1 * (m - a - c)!)]))^b$ describes the structure of the tree, the exponent b takes into account, that each of these b applicants have this choice.

7.2 A Combinatorial Formula

My idea was to get to the total formula:

$$\sum_{f,s\geq 1, f+s<n} DIFFPOST(0,f,0) * DP(f,s) * ERPOST(2,n,f-1,s) \tag{1}$$

$$+ \sum_{size=3}^{n-1} \sum_{f,s\geq 1, f+s<size} \sum_{i=1}^{n-size} \sum_{j=1\geq n}^{n-size} DIFFPOST(0,f,n-size) * DP(f,s) * ERPOST(2,size,f-1+i,s)$$
$$* SUBSET(size) * DIFFPOST(f+s,i,size) * ERPOST(2,n-size,f-1+i,j) \tag{2}$$

It is based on Lemma 56. The idea is to count exactly those instances for which for some subset A_1, $|B(A_1)| < |A_1|$ and thus no popular matching exists. Therefore we examine every possible size of A_1, first we consider subsets containing all applicants. Here an instance does not allow a popular matching, if the number of f-posts plus the number of s-posts is smaller than the number of applicants. We iterate over all values for the number of f- and s-posts satisfying this condition and calculate for each pair of values the number of instance allowing no popular matchings: We have $DIFFPOST(0,f,0)$ possibilities to choose the f-posts and $DP(f,s)$ possibilities to choose the s-posts (since the f-posts are already chosen, $n-s$ posts remain). By $ERPOST(2,n,f-1,s)$ we select the remaining priorities (1)
Basically we repeat this for all other sizes of subsets greater than 3. For this though the formula slightly changes. First, in $ERPOST$ also the f posts of the applicants not in the subset (let i be this number) have to be taken care of, thus we add i to the number of posts which can appear before the s-post. Second, we multiply the calculated number with the number of possible subsets of the current size. Furthermore, we have to count the number of possibilities how the priorities of the applicants not in the subset can look like. $DIFFPOST(f+s,i,size)$ describes their choosing of their i f-posts, the f- and s-posts of the applicants in the subset are already taken and subtracted from the possible posts. In the $ERPOST$ part, the f-posts of the subset as well as the i f-posts of the applicants not in the subset are remembered. Call the number of occurring s-posts of the applicants not in the subset j.
By this though we now count, as mentioned, too much. There seem to exist four reasons for this, call the applicants forming the subset with size s S, the set containing the other applicants R:
1. We can move one applicant from R to S, if in S still no popular matching exists, we already counted that instance in an iteration of a bigger subset S.
2. We can move multiple applicants from R to S, with the same argumentation as in 1, we count too much here.
3. / 4. The same can be done from S to R.
Removing 1 and possibly 3 should not be too hard. However, 2 and 4 are hard to take

account of in the formula. Especially by moving applicants from S to R, in R of course the number of applicants increases. Also the number of f- and s-posts increases, but depending on how many of the moved posts share these, probably not as fast. This can lead to other not popular matchings, which might have been counted in other iterations. One valid result though the reader can take, is that the number of instances not allowing a popular matching, where exactly s applicants prohibit the popular matching, can be counted, for this just take the corresponding iteration of the calculation:

Lemma 57. *The number of instances, in which at least "size" applicants exist for which no popular matching can be found, is given by*

$$\sum_{f,s\geq 1, f+s<size} \sum_{i=1}^{n-size} \sum_{j=1\geq n}^{n-size} DIFFPOST(0, f, n-size) * DP(f,s) * ERPOST(2, size, f-1+i, s) * SUBSET(size) * DIFFPOST(f+s, i, size) * ERPOST(2, n-size, f-1+i, j) \quad (3)$$

For me this is still an interesting result, thinking for example towards the relaxation of the definition of popular matchings as done in Section 10, where subsets of applicants may be removed to find a popular matching in every instance.

7.3 Estimating the Number of Popular Matchings

Despite not knowing how to count the exact number of popular matchings, we can estimate this number very accurately testing many random instances.
When creating a number m of random instances, dividing the number of instances allowing a popular matching by m and then taking this factor for the total number of possible instances (which of course can be easily calculated by y^n, where y is the number of possible permutations of a preference list and n the number of applicants), the difference between the estimated result and the actual result will converge to a normal distribution when m gets large. For the normal distribution a 95 % confidence interval is given by $[\mu - 1.96 * \delta, \mu + 1.96 * \delta]$, with μ being the mean and δ the standard deviation. Since we are here interested in estimations based on empirical data, we also have to estimate mean and standard deviation. Interpreting $x_i = 1$ as instance i allowing a popular matching and $x_i = 0$ as the opposite, we use the following easy estimators for mean and standard deviation respectively: $\mu = \frac{\sum_{i=1}^{m} x_i}{m}$, $\delta = \sqrt{\frac{1}{m-1} \sum_{i=1}^{m} (x_i - \mu)^2}$.
Details of the implementation can be found in Section 13.1.7, using that algorithm we get the result shown in Figure 2 (for each instance size 1000000 random instances were created).

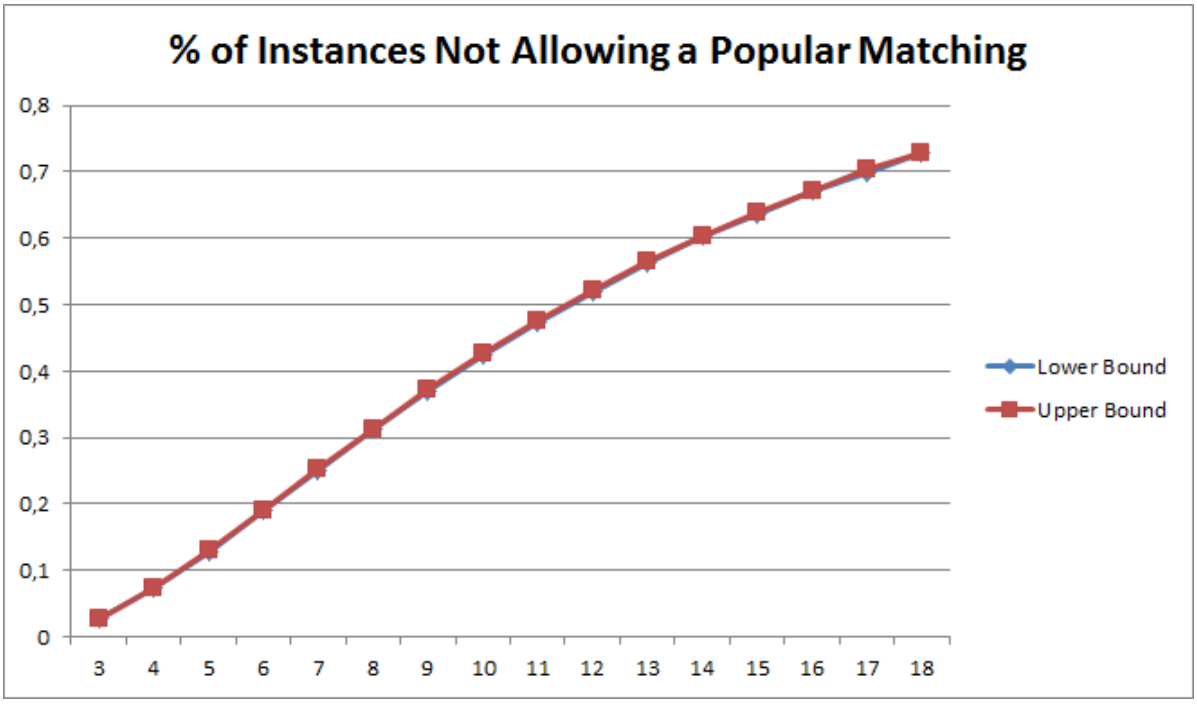


Figure 2: Estimated Percentage of Instances Not Allowing a Popular Matching

8 Using Linear Programs in the Popular Matching Problem

In this section we want to introduce the usage of linear programs for problems related to popular matchings. The plan was to first describe POP-M using a linear program and then extending this concept to define a polytope consisting of subsets of the applicants and posts allowing a popular matching. This would be a very interesting theoretical result and has been done for regular matchings in [11]. The popular matching linear program is formulated, but the polytope could not be described. However here a negative result showing what is not sufficient to define it could be stated.

8.1 The Popular Matching LP

In this section we want to formulate the popular matching problem as a linear program. This approach is not present yet in the existing literature, however a formulation of the optimal edge weighted popular matching problem as a linear program ([12]).
A representation of a standard maximum matching problem on a bipartite graph with edge set E and node set V looks as follows, for every edge e there exists a corresponding variable x_e:

Maximize $\sum_{e\ inE}$ s.t.
$\forall u \in V \sum_{e \sim u} x_e \leq 1$
$\forall e \in E\ x_e \geq 0$

As all optimal solutions of this turn out to be already integer, we can formulate the matching problem as it is and do not have to add integer constraints.
We will use a similar characterization for the popular matching problem here, but since the behavior of the optimal solutions is unknown, we will formulate it as an integer linear program.
Again there are variables x_e for each edge e, A denotes the set of applicants and P the set of posts. Note that, since the graph is bipartite, to every edge belongs exactly one applicant and one post. The ILP looks as follows:

Maximize 0 s.t.
$\forall a \in A \sum_{e \sim a} x_e = 1$ (1)
$\forall p \in P \sum_{e \sim p} x_e \leq 1$ (2)
$\forall e \in E\ x_e \leq 0$, if e's rank is not 1 and e points to an f-post (3)
$\forall e \in E\ x_e \leq \begin{cases} \leq 1, \text{ if } e \text{ is the highest ranked edge of the incident applicant pointing to a non f-post} \\ \leq 0, \text{ otherwise} \end{cases}$
(4) $\forall e \in E\ x_e \in \{0, 1\}$ (5)

Constraint (1) requires, that the found solution is a valid matching in every applicant, but also, that all applicants are matched, which is a necessary criteria for popular matchings. Constraint (2) requires, that the solution is a valid matching in every post. With

constraint (3) we forbid the choosing of edges, which have not rank 1 but point to f-posts. Constraint (4) enforces the correct selection of an s-post for every applicant. Only one edge per applicant is allowed to point towards a non f-post, and this is required to be the highest ranked edge, which ensures the existence of the correct edge towards the s-post for each applicant. Constraint (5) finally is the integer constraint.
Note though, that formulating the LP for the popular matching problem and solving it is not feasible. The effort is probably exponential with respect to the input, and we anyway already have a very efficient polynomial time algorithm for this problem. However, it would be practically useful to formulate the NP-complete problems defined in Section 9 and 10, namely the relaxations of the popularity condition, as then we could calculate their solutions more efficiently.

8.2 The Popular Matching Polytope

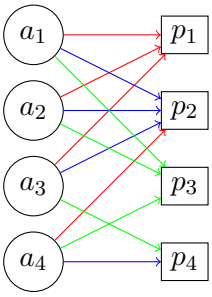
After having successfully modeled the popular matching problem as a linear program, a logical extension is close: It would be very interesting to define the so called popular matching polytope, that is the polytope formed of all vectors representing subsets of the graph nodes, such that these subsets admit a popular matching. Unfortunately, an efficient characterization for this polytope could not be found, however a negative result shall be presented, showing what is not sufficient for the definition.
In [11] this was done for the regular matching problem. There a system of linear inequalities was given to classify the perfectly matchable subgraph polytope of a given bipartite graph - that is, all those subsets of the nodes, such that there exists a perfect matching in them. For this, some sort of "counting argument" was used, the inequalities expressed, that for every subset of the nodes at least as many neighbors had to be picked to ensure a possible matching. More specifically, the following inequalities were used to describe the polyhedron [11], here $x(Y)$ is the number of nodes included in the set Y and $\Gamma(Y)$ is the set of neighbors from nodes in Y:

$0 \leq x_i \leq 1$ for all $i \in V$
$x(V_1) - x(V_2) = 0$
$x(S) - x(\Gamma(S)) \leq 0$ for all $S \subseteq V_1$

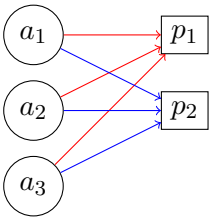
Furthermore, this LP turned out to have integer optimal solutions, which made it extremely useful for the practice. Since a popular matching is a perfect matching in the reduced subgraph, the principle could be tried to copy for this topic. However, because the reduced graph can be different for each subset of the applicants, such a "simple counting argument" like before, for example setting the number of applicants in the subset in relation to the posts of their corresponding reduced graph, is not sufficient. More precisely, it is not hard to show:

Lemma 58. *A system of linear inequalities containing the inequalities $x(A_1) \leq f(A_1)$ for every $A_1 \subset A, |A_1| \geq 3$ cannot define the popular matching subgraph polytope for any function f.*

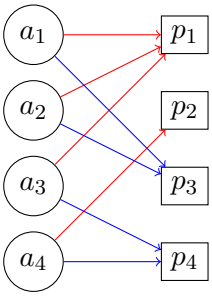
Proof. See this graph, here red lines represent rank-1 edges, blue lines rank-2 edges, green lines rank-3 edges:



When restricting our attention to a_1, a_2 and $a_3 = A_3$ the reduced graph looks as follows:



So for this instance no popular matching exists, in order to define the desired polytope $f(A_3) < 3$.
However for a_1, a_2, a_3 and $a_4 = A_4$ the inequality should be fulfilled, since the complete instance allows a popular matching, as the reduced graph looks as follows:



But when considering the LP for A_4, also the inequality regarding A_3 is included, which is still not satisfied. Therefore, no LP of the above classification can define the polytope for instances of the size 4. Furthermore, we can add arbitrarily many applicants with a distinct first preference post to construct arbitrarily big instances containing our counter examples for which the approach described above does not work. □

9 The Least-Unpopularity Factor

In this section we will discuss the least-unpopularity factor of matching instances ([4]), which is a criteria for choosing a matching if no popular matching exists. The idea of it is to calculate for every possible two matchings N, M a factor u/v, where u is the number of people who prefer N over M and v is the number of people who prefer M over N. This way, matchings can be compared in their "goodness", eventually allowing us to chose the "best" matching of a selection of "bad" ones.

9.1 Calculating the Least-Unpopularity Factor

First we need to define

Definition 59. *([4]) A matching N dominates a matching M by a factor of u/v, if u people prefer N over M, and v people prefer M over N.*

Definition 60. *([4]) The unpopulariy factor of a matching is the greatest factor by which it is dominated by any other matching (ignore matchings with $u = v = 0$).*

Definition 61. *([4]) The least-unpopularity factor of an instance is the minimum of the unpopularity factors of all possible matchings. The matchings which achieve this factor are considered optimal.*

Note that a matching M has an unpopularity factor of ∞, if and only if another matching represents a Pareto-improvement over M. That means that a matching has a finite unpopularity factor if and only if it is Pareto efficient. Furthermore, a matching is popular according to our basic definition, iff its unpopularity factor is 1 or less.
Like done before with switching paths and switching circles (e.g. in Section 4), here we will consider applying paths and circles to matchings. The paths and circles are the same as switching paths and circles except the additional requirements, so they are paths and circles containing applicants and pots alternatively. Again we can apply these paths and circles, matching every applicant in it to its other neighbor in the path or circle. We say a path T or circle C is applicable to a matching M if $M \cdot T$ or $M \cdot C$ is a valid matching. Each reassignment in a path or circle may represent a promotion or demotion with respect to the applicant, depending on his preferences.

Pressures

Directly using the definition above, to calculate the unpopularity factor of a matching we would have to calculate all other matchings and compare them, which would mean exponential effort. Luckily though there is a way to calculate the unpopularity factor efficiently, using the concept of pressures.

Definition 62. *([4]) Let M be a pareto efficient matching. The pressure of a filled position p in M is the largest k for which there exists an alternating path applicable to M, which promotes k people without demoting anyone and ends with the demotion of the occupant of p to his last resort. Note that the demotion itself is also such a path for $k = 0$. The term pressure comes from the idea of k people stacked behind the current occupant of p, all of them wanting to improve their current position.*

Theorem 63. *([4]) The unpopularity factor of a matching is the greatest pressure of any of its filled positions.*

Proof. ([4]) We have to proof: If (a) M has a position with pressure k, then there exists another matching N which dominates M by at least k and (b), if M is dominated by another matching N with factor k, then there exists a position with pressure at least k. (a) is fairly easy. For this simply consider the position with pressure k and its k "stacked" applicants. Let N be the result of applying this path to M, then there are k people better off compared to one person who is worse off, giving the desired unpopularity factor of k. In (b) we let u and v be the numbers of people who are better and worse off in N than in M. First we demote all people, who are worse off in N than in M, to their last resorts, this does not change u nor v. When now examining $M \oplus N$ and leaving out paths and cycles which do not demote or promote anyone, we find only disjoint paths, no cycles. This is, because all applicants worse off in N are matched to their last resort, letting the path end in this. Furthermore, there has to exist at least one path demoting someone, so that M is not a Pareto-improvement. So there exist i paths demoting exactly one person and promoting the others, with $i = v$. Let u_i be the number of people being promoted in path i, of course $\sum u_i = u$. By the pigeon hole principle then at least one u_i exists with $u_i \geq \lceil u/v \rceil = k$. Take the position whose applicant is demoted, this gives the desired pressure. □

Corollary 64. *([4]) The unpopularity factor of a matching, if finite, is an integer.*

Computing the Unpopularity Factor

Theorem 65. *The unpopularity factor of a given matching can be calculated in time $O(m\sqrt{n})$, where n is the number of positions and m is the total number of entries in the preference lists.*

The pressures of a matching can be calculated with the following algorithm (presented in [4]), which is based on Goldberg's shortest-path algorithm ([13]):
Out of a matching M create a graph G, whose vertices are positions and with edges defined as follows:

1. Add an edge from p_1 to p_2 with weight -1, if p_1 is filled by a person, who strictly prefers p_2 over p_1.

2. Add an edge from p_1 to p_2 with weight 0, if p_1 is filled by a person who is indifferent between the two posts.

Run Goldberg's algorithm.
If a cycle with negative length or a path with negative length arriving in an unfilled position is found, output that M is not Pareto-optimal. Otherwise the pressure of each position is the negative of the length of the shortest path arriving at it.

Correctness ([4]): The edges in G represent possible reassignments of people, exactly the ones which are not demotions are included in the graph. If there exists a cycle with negative length or a path of negative length arriving at an unfilled position, each person in it can be promoted without demoting anyone, thus giving a Pareto-improvement. Otherwise, for every position the shortest negative path represents the maximum number of people "stacking up" behind that position, which is the pressure of the position.

Taking the running time of Goldberg's algorithm which dominates the overall algorithm, we find a running time of $O(m\sqrt{n})$.

9.2 NP-hardness of Finding Least-Unpopularity Matchings

In this section we will explain the polyomial time reduction from the NP-complete problem 3-satisfiability (3SAT) to finding the least-unpopularity factor of an instance, like presented in [4], which of course then shows, that finding the least-unpopularity factor of an instance is also NP-hard. The reduction converts an instance of 3SAT to a matching instance, which has a least-unpopularity factor of at most 2 iff 3SAT is satisfiable, otherwise the least-unpopularity factor is higher.

Overview over the Reduction

Like many reductions this one here also uses different gadgets with a different purpose to model an instance of 3SAT as a matching instance. For each variable x we create a m-n gadget, with the purpose of forcing a decision whether to satisfy x or $\bar{x}$. For each positive and negative occurrence of the variable a post is created, which is left open if x is allocated that way. Furthermore for each clause a pool is created, which is connected to all m-n gadgets of the variables occurring in the clause. The pool needs to get at least one post, otherwise it produces a pressure higher than the one allowed. Then an instance of 3SAT is satisfiable, iff the matching instance has a least-unpopularity factor of at most 2. Each gadget consists of internal and linking people, as well as internal and external posts. Internal people are only willing to occupy internal positions, whereas a linking

person is open to occupy also exactly one linking position, which is his first choice. From his point of view any reassignment from that position is a demotion and could also just be a demotion to his last resort. Thus the state of domination, which can be achieved by demoting a linking person, depends only on the state of the gadget providing the replacement - gadgets are isolated from each other unpopularity wise. Therefore, each gadget represents a specific constraint, if it is satisfied the gadget produces a pressure of at most the ideal pressure of 2, otherwise a higher pressure results.

The Gadgets

The first type of gadget is the box. It consists of four internal positions (x, y, z and u), three internal people (i_1, i_2 and i_3) and three linking people (w, n_1 and n_2). The priority lists look as follows:

$$\begin{array}{c} \\ w \\ i_1 \\ i_2 \\ i_3 \\ n_1 \\ n_2 \end{array} \begin{array}{c} \begin{array}{ccccccc} x & y & z & u & l_w & l_{n_1} & l_{n_2} \end{array} \\ \begin{pmatrix} 2 & 3 & 5 & 4 & 1 & - & - \\ 1 & 2 & 3 & 4 & - & - & - \\ 1 & 2 & 3 & 4 & - & - & - \\ 1 & 2 & 3 & 4 & - & - & - \\ - & - & - & 2 & - & 1 & - \\ - & - & - & 2 & - & - & 1 \end{pmatrix} \end{array}$$

We call w the wide person and n_1 and n_2 narrow people. A box is satisfied and produces a pressure of 2, if either the wide person or both narrow people get their linking positions. However, if both the wide person and at least on narrow person are denied their linking position, the gadget constraint is not satisfied and a pressure of 3 results.
A box is a "two-for-one" gadget. If it is denied its wide linking position, it demands both narrow linking positions. Building on this we can create an m-for-one and an m-for-n gadget. When taking m-1 one boxes and always identifying the wide positions of each box after the first one with a narrow position of the previous box, we get a m-for-one gadget. When combining two such gadgets, respectively with $m-1$ and $n-1$ boxes, by giving the two boxes in the middle the same wide position, we get an m-for-n gadget. This is, because if one of the middle boxes claims the central wide position, the m narrow positions on that side can remain free, but the n narrow positions of the other side have to be occupied. See Figure 3 for an exemplary 4-for-3 gadget. In it the circles denote linking positions and the lines adjacent to them linking people, internal people and positions are not shown but just symbolized by the box. On the left side of the gadget both narrow positions of each box are visualized to its left, the wide person to its right - on the right side it is the other way around.

A pool consists of two internal positions and three linking people with the following structure:

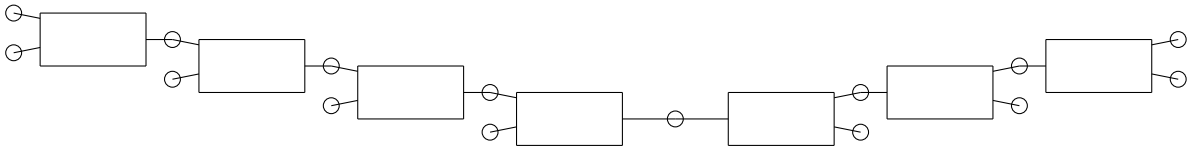


Figure 3: A 4-for-3 gadget

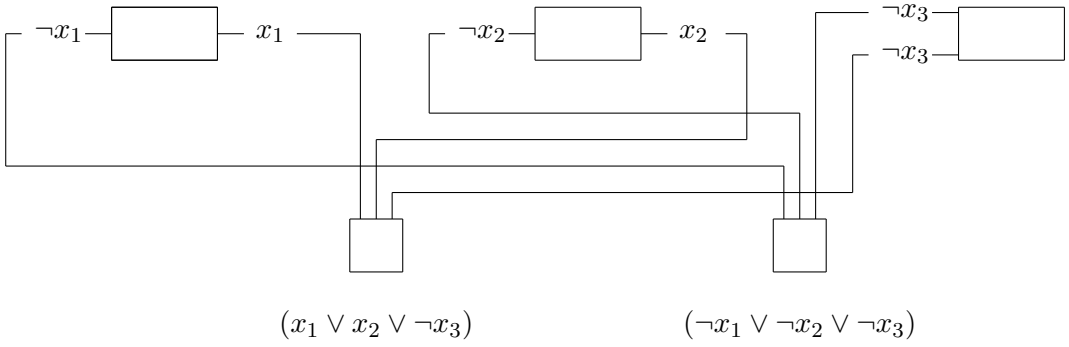


Figure 4: An Exemplary Matching Instance

$$\begin{array}{c} \\ f_1 \\ f_2 \\ f_3 \end{array} \begin{array}{c} \begin{array}{cccc} x & y & l_{f_1} & l_{f_1} & l_{f_3} \end{array} \\ \begin{pmatrix} 2 & 3 & 1 & - & - \\ 2 & 3 & - & 1 & - \\ 2 & 3 & - & - & 1 \end{pmatrix} \end{array}$$

If k of the people in the pool are denied their linking positions, there exists one linking position with a pressure of k.

The Reduction

For each variable x_i occurring in the instance of 3SAT we create an m-for-n gadget, where m and n are the numbers of positive and negative occurrences of x_i. We will think of the set of narrow positions of the respective side as references to x_i or $\bar{x_i}$. Then for each clause we add a pool and identify its three linking positions with narrow posts of the m-for-n gadgets of the variables it contains.
The example in Figure 4 shows how the 3SAT formula $(x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3)$ is converted to a matching instance: In this diagram rectangles represent m-for-n gadgets and squares pools. The narrow linking positions of the m-for-n gadgets for the corresponding variables are displayed by their label (e.g. x_1), lines display linking people.

Correctness

We now show how to find a valid matching with pressure at most 2 in the created instance, if the corresponding instance S of 3SAT is satisfiable by a tuple of truth values $(t_1, \ldots, t_k)$. If t_i is true we match each box on the non-negated (x_i) side according to the first table below, fillings its wide linking position, in each box on the negated ($\neg x_i$) side according to the second table, filling both its narrow linking positions. Here the superscripts over the posts show the resulting pressure. In the first table n_1 is the linking person whose linking position is shared with a pool, and n_2 is matched to his last resort.

	x^2	y^1	z^0	u^1	l_w^0	$l_{n_1}^2$	$l_{n_2}^1$
w	2	3	5	4	(1)	–	–
i_1	(1)	2	3	4	–	–	–
i_2	1	(2)	3	4	–	–	–
i_3	1	2	(3)	4	–	–	–
n_1	–	–	–	(2)	–	1	–
n_2	–	–	–	2	–	–	1

	x^2	y^1	z^0	u^0	l_w^1	$l_{n_1}^0$	$l_{n_2}^0$
w	2	3	5	(4)	1	–	–
i_1	(1)	2	3	4	–	–	–
i_2	1	(2)	3	4	–	–	–
i_3	1	2	(3)	4	–	–	–
n_1	–	–	–	2	–	(1)	–
n_2	–	–	–	2	–	–	(1)

Observe now, that all narrow linking positions l_{n_1} are not occupied and can be taken by linking people of x_i containing pools. If t_i is false, we match the boxes on each side the other way around. Then we assign every person in the pools to their linking position if it is available, otherwise to the best available position in the pool. Since the assignment of the t_i satisfies S, for each pool at least one person can be assigned to his linking position, resulting in a pressure of at most 2. Now since nowhere we found a pressure of more than 2 and the gadgets are "isolated", like stated before, M has the least-unpopularity factor of at most 2.
Now let us assume S is not satisfiable and let M be an arbitrary Pareto-efficient matching of the created instance. Then, by the previous comments, one can see that not all gadget constraints can be satisfied. It remains to show that we then indeed get a least-unpopularity factor of greater than 2. First suppose a box constraint is dissatisfied, i.e. the wide person and at least one narrow person (say n_1) are denied its linking position. If one position of the box was not occupied in it, one person was matched to his last resort and the matching would not be Pareto-optimal, as that person could be matched to the empty post. If n_1 is matched to his last resort, we could promote him to u, its occupant to y, its occupant to x and demote its occupant to his last resort. Otherwise, one of the persons w, i_1, i_2 or i_3 is matched to his last resort. Then we can promote him to z, its occupant to y, its occupant to x and again demoting its occupant to his last resort. Either way, in both cases at least a pressure of 3 occurs. Now suppose a pool constraint is dissatisfied, meaning all persons in the pool are denied their linking positions. Then one person must be matched to x, one to y and one to his last resort, say the order is f_1, f_2 and f_3. We can promote f_3 to y, f_2 to x and f_1 to its linking position, demoting its previous occupant to his last resort. Again this gives a pressure of 3, which in total shows, that M cannot achieve a least-unpopularity factor of 2 or lower.

10 Subset Maximal Popular Matchings

After Section 9 in this section another relaxation of the definition of popular matchings, in order to find a satisfying result for every instance, is to be proposed. This is the concept of subset maximal popular matchings, which is a very natural criteria to apply, if no popular matching exists, yet it has not been investigated though. For this we define the applicant number α of an instance of POP-M:

Definition 66. *Let $G = (A \cup P, E)$ be an instance of the popular matching problem. Then, the applicant number of G, $\alpha(G)$, is defined as:*
$\alpha(G) = max\{|A'| \mid A' \subseteq A \wedge G' = (A' \cup P, E)$ *admits a popular matching*$\}$.

Using this, we define which matching is optimal, and we are always able to output a matching when presented an instance of POP-M:

Definition 67. *Given an instance $G = (A \cup P, E)$ of POP-M, a matching M' is a subset maximal popular matching iff it is a popular matching in a subset A' of the applicants with $|A'| = \alpha(G)$. In M', every applicant not contained in A' is unmatched.*

Definition 68. *Given an instance $G = (A \cup P, E)$ of POP-M, the problem POPSUB-M is the problem of finding a subset maximal popular matching of G.*

Note that for every instance a subset maximal popular matching exists (trivially with $|A'| = 1$) and that there can exist multiple subset maximal popular matchings. Furthermore, every popular matching is also a subset maximal popular matching with $|A'| = |A|$.

10.1 Complexity of Finding a Subset Maximal Popular Matching

The question arises how difficult it is to find a subset maximal popular matching. It is easy to see that POPSUB-M $\in NP$. When examining the corresponding decision problem to find a subset maximal popular matching with $A' \geq d$, we can present a string coding the resulting subset for which the needed conditions can be checked in polynomial time. Using the decision problem, POPSUB-M can be solved by, for example, a binary search over possible values for d.
It is not yet known what a lower bound for solving POPSUB-M is, intuitively it also seems to be NP-hard, as exponentially many subsets of the applicants exist and for each the reduced graph can be different. A similar approach as in the reduction for finding the least-unpopularity factor was tried but not finished.

11 Conclusion

Popular matchings are a natural criterion for defining fair matchings. The facts that they are efficiently computable and also somewhat robust against manipulations justify their practical application. However, so far they are just examined in theory, but recently the interest in them has grown due to certain publications. A negative point though is that not every instance allows a popular matching. Two generalizations of the original definition, so that for every instance a matching can be chosen, are known: A definition using the least-unpopularity factor and subset maximal popular matchings. What is still missing is a generalization which can be computed efficiently, as for the previous two no polynomial time algorithm is known.
This work has hopefully collected the most important results regarding popular matchings. Hopefully, also the new facts and questions presented here are helpful and interesting for further research.

12 Acknowledgments

I want to thank Marco Lübbecke and Florian Dahms for introducing me to the fascinating topic of popular matchings. And, of course, for the opportunity to write my bachelor thesis in this field and their continuous and great support. I also want to thank Britta Peis for being the second corrector.

References

[1] David J. Abraham, Robert W. Irving, Telikepalli Kavitha, and Kurt Mehlhorn. Popular matchings. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '05, pages 424–432, Philadelphia, PA, USA, 2005. Society for Industrial and Applied Mathematics.

[2] Eric McDermid and RobertW. Irving. Popular matchings: Structure and algorithms. In HungQ. Ngo, editor, *Computing and Combinatorics*, volume 5609 of *Lecture Notes in Computer Science*, pages 506–515. Springer Berlin Heidelberg, 2009.

[3] Meghana Nasre. Popular matchings – structure and cheating strategies. *CoRR*, abs/1301.0902, 2013.

[4] RichardMatthew McCutchen. The least-unpopularity-factor and least-unpopularity-margin criteria for matching problems with one-sided preferences. In EduardoSany Laber, Claudson Bornstein, LoanaTito Nogueira, and Luerbio Faria, editors, *LATIN 2008: Theoretical Informatics*, volume 4957 of *Lecture Notes in Computer Science*, pages 593–604. Springer Berlin Heidelberg, 2008.

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

[6] P. Gärdenfors. Match making: Assignments based on bilateral preferences. *Syst. Res.*, 20:166–173, 1975.

[7] Colin T. S. Sng and David F. Manlove. Popular matchings in the weighted capacitated house allocation problem. *J. of Discrete Algorithms*, 8(2):102–116, June 2010.

[8] DavidJ. Abraham and Telikepalli Kavitha. Dynamic matching markets and voting paths. In Lars Arge and Rusins Freivalds, editors, *Algorithm Theory – SWAT 2006*, volume 4059 of *Lecture Notes in Computer Science*, pages 65–76. Springer Berlin Heidelberg, 2006.

[9] Telikepalli Kavitha and Meghana Nasre. Optimal popular matchings. *Discrete Applied Mathematics*, 157(14):3181 – 3186, 2009.

[10] J. Hopcroft and R. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2(4):225–231, 1973.

[11] E. Balas and W. Pulleyblank. The perfectly matchable subgraph polytope of a bipartite graph. *Networks*, 13:495–516, 1983.

[12] Vamski Kundeti. Algorithms for optimal popular matching in a weighted bipartite

graph under general preferences.

[13] A. Goldberg. Scaling algorithms for the shortest paths problem. *SIAM Journal on Computing*, 24(3):494–504, 1995.

[14] Bettina Klaus and Flip Klijn. Procedurally fair and stable matching. *Economic Theory*, 27(2):431–447, 2006.

[15] F. Masarani and S.S. Gokturk. On the existence of fair matching algorithms. *Theory and Decision*, 26(3):305–322, 1989.

[16] Bettina Klaus and Flip Klijn. Median stable matching for college admissions. *International Journal of Game Theory*, 34(1):1–11, 2006.

[17] Chien-Chung Huang and Telikepalli Kavitha. Popular matchings in the stable marriage problem. In Luca Aceto, Monika Henzinger, and Jiří Sgall, editors, *Automata, Languages and Programming*, volume 6755 of *Lecture Notes in Computer Science*, pages 666–677. Springer Berlin Heidelberg, 2011.

[18] Rupam Acharyya, Sourav Chakraborty, and Nitesh Jha. Counting popular matchings in house allocation problems. In EdwardA. Hirsch, SergeiO. Kuznetsov, Jean-Éric Pin, and NikolayK. Vereshchagin, editors, *Computer Science - Theory and Applications*, volume 8476 of *Lecture Notes in Computer Science*, pages 39–51. Springer International Publishing, 2014.

13 Appendix

13.1 Implementation

In this section a program written during the course of this bachelor thesis shall be presented, which implements most of the algorithms described in this work. In my opinion it is a useful tool to find and understand popular matchings and related problems, especially because of its graphical output. In this section also example runs of described algorithms are depicted. The source code and an executable can be found online at https://github.com/OliverScheel/PopularMatchings.
The implemented functionalities include:

1. Finding popular matchings using Algorithm 3.1 for instances without ties and using Algorithm 3.3 for instances with ties.

2. Finding popular matchings using a linear program.

3. Creating the switching graph for a popular matching, counting and enumerating all popular matchings.

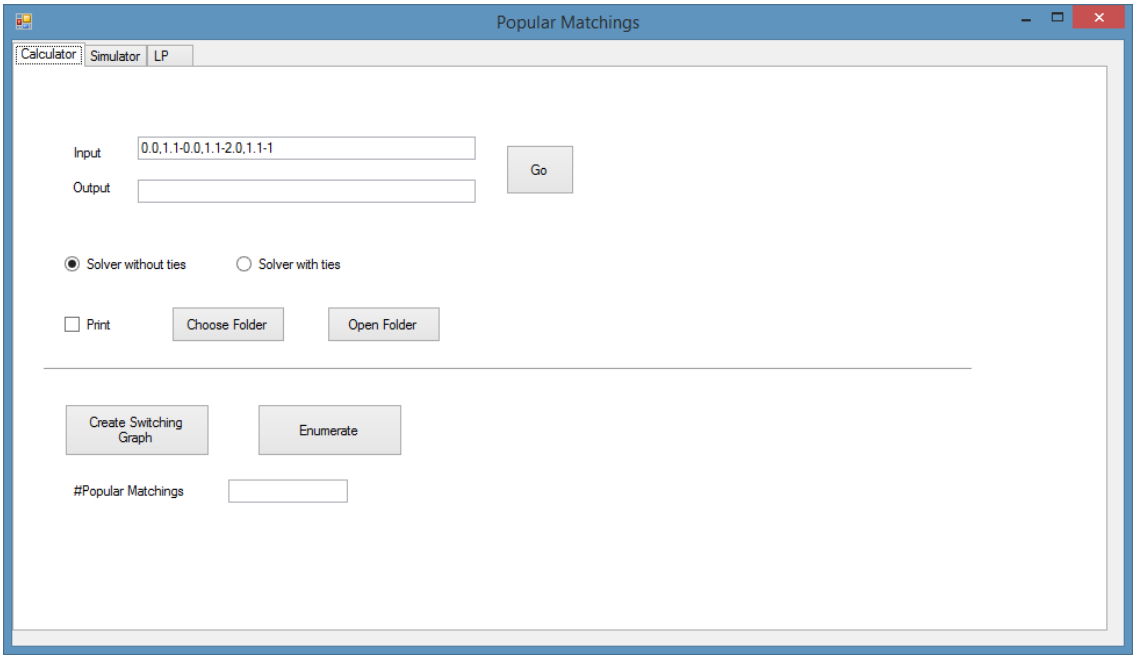
4. Counting all instances that allow a popular matchings of a given size n, that means all instances with n applicants and n posts are checked, with all applicants having a complete preference list.

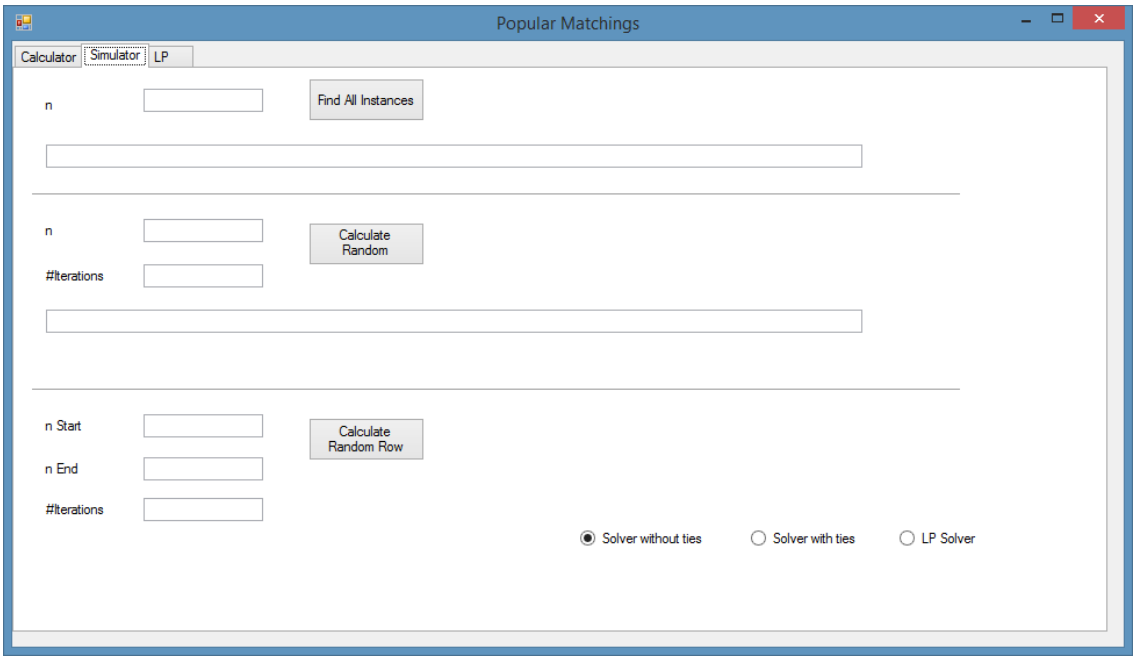
5. Creating m random instances of a given size n, that means all instances with n applicants and n posts are checked, with all applicants having a complete preference list, and counting the number of instances that allow a popular matching.

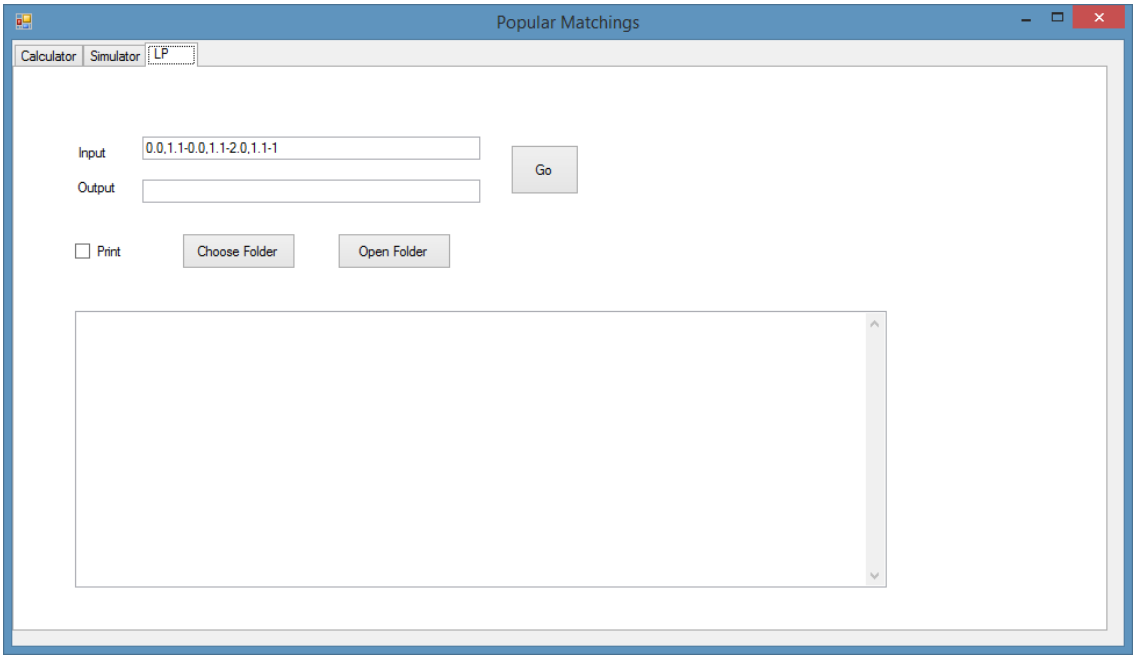
6. Repeating 5) for each m in a row of numbers, and calculating for each m a 95 % confidence interval to estimate the actual number of instances allowing a popular matching.

13.1.1 Overview

The program has been written in C# for Windows computers.
Before coming to the code, we will briefly look at the user interface and give an overview of its usage. The program consists of one form with three tab pages, which look as follows:


Popular Matchings
Calculator
Simulator
LP
Input
0.0,1.1-0.0,1.1-2.0,1.1-1
Output
Go
Solver without ties
Solver with ties
Print
Choose Folder
Open Folder
Create Switching Graph
Enumerate
#Popular Matchings


Popular Matchings
Calculator
Simulator
LP
n
Find All Instances
n
Calculate Random
#Iterations
n Start
Calculate Random Row
n End
#Iterations
Solver without ties
Solver with ties
LP Solver

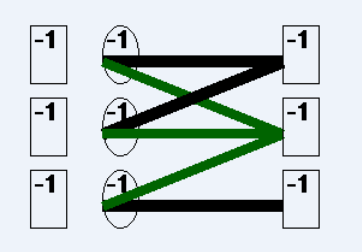


On the first tab page, "Calculator", one can find a popular matching using either the solver for instances without ties or with ties. For this, the instance has to be encoded and entered in the textfield "Input". After clicking "Go", the result is displayed next to "Output". If the option "Print" is enabled, the solver will create a graphical output. Via "Choose", the output path for this can be determined, "Open Folder" opens it in the Windows Explorer.
On the bottom half of the page, one can experiment with the switching graph after a popular matching has been found. When clicking "Create Switching Graph", the switching graph is created and the number of popular matchings for this instance is calculated and outputted next to "#Popular Matchings". If "Print" is enabled, the switching graph is pictured in the selected directory. Clicking "Enumerate" creates an enumeration of all popular matchings, which are displayed in the selected path in the subfolder "Enumeration".

On the second tab page "Simulator", via "Find All Instances", all instances of the given size are created and the number of instances which do not allow a popular matching is displayed, as well as the total number of instances. Note that his function is merely for testing purposes and is not feasible already for sizes greater than 4 or 5.
Via "Calculate Random", "#Iterations" many instances of the desired size are created and tested for popularity. The output is a 95 % confidence interval for the actual number of instances not allowing a popular matching.
"Calculate Random Row" does the previous for the row of numbers between "n Start" and "n End" and outputs the results in the Excel file "confidence.xls" in the application directory.

On the last tab page, "LP", the popular matching problem is solved using linear programming. Next the result also the constructed LP, i.e. the constraints, is displayed in the text field at the bottom.

Let us conclude this section with the description of the input and output format of the program.
As input simply the joint priority list of the instance is required, that is the concatenation of the input lists of all existing applicants. One single preference list entry of an applicant consists of the ID of the post and the rank corresponding to that edge, it is written as $targetID.rank$. Each entry is separated by a comma, whole preference lists are separated by a minus. At the end of the input string we append $-capacity$ to describe the capacity of each post. Note that this way IDs of applicants are given automatically, while reading the input for each found priority list an applicant is created and the ID is incremented. IDs of posts are given implicitly, for each occurring post ID in a preference entry a post is created. A sample input could look as follows: $0.0, 1.1 - 0.0, 1.1 - 2.0, 1.1 - 1$. This scheme can also be simplified by omitting every rank, when the priority list is ordered and does not contain ties. Textwise, the output of the program is either "No Popular Matching" or a string representation of a matching. In it, simply all matching partners are printed in the form ApplicantID− >PostID.
The program can also output instances and matchings in a graphical form: Posts are drawn as rectangles, applicants as circles, the posts on the right side, the applicants left of it. Each group is drawn in a vertical line, the object with ID 0 is located at the top, with increasing IDs towards the bottom. The unique l-posts of the applicants are drawn to their left. Preferences are represented as lines, with decreasing thickness for decreasing rank. For more clarity colors are iterated between black and green, for example we get the following: Rank 1 - Black, Thick; Rank 2 - Green, Middle; Rank 3 - Black, Thin. If an applicant is matched to a post, the line is drawn in red. The types of the nodes, as determined during the algorithm of the solver for instances with ties, is written next to the node. -1 means not yet determined, 0 even, 1 odd and 2 unreachable. The input example from before is then represented as follows:



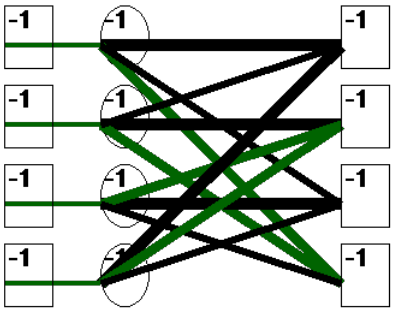
13.1.2 Solver for Instances Without Ties

In this section we will look at the class $PopSolver$, which can solve the popular matching problem for instances without ties. The function $Match()$ is responsible for find-

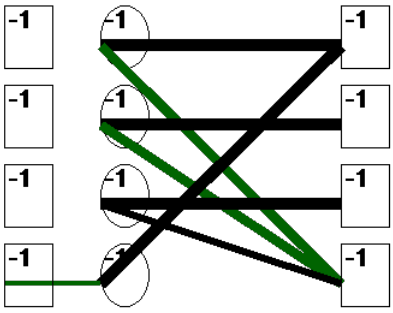
ing a popular matching. In *GetReduced*() first the reduced graph is calculated. In $Inflate()$ the posts are inflated so that each post has only capacity 1. The function *GetApplicantComplete*() then checks for the existence of an applicant-complete matching. For this, first all posts connected to a single applicant are matched and posts with degree 0 are deleted. If then still an applicant-complete matching can be found, the function *GoPath*() is called, which traverses all the disjoint cycles of the graph and matches every second edge. Afterwards, in $Match()$, all f-posts are filled.

The input $0, 3, 2-1, 3, 0-2, 1, 3-0, 1, 2-1$ creates the output below, as you can see there exists a popular matching and the resulting images represent the important steps of the algorithm:

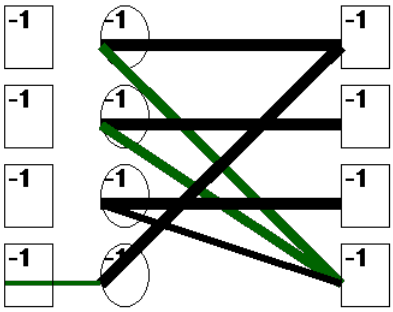
0Start.bmp



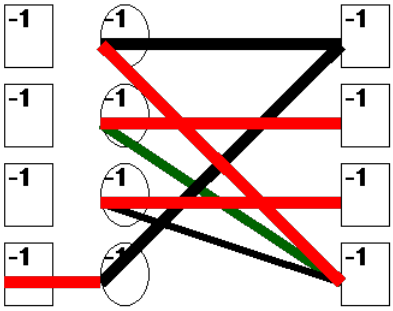
1Reduced.bmp



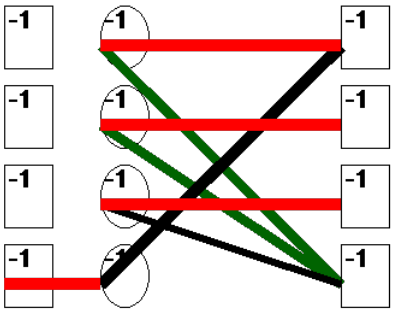
2Inflated.bmp



3ApplicantComplete.bmp



4AllFPostsMatched.bmp

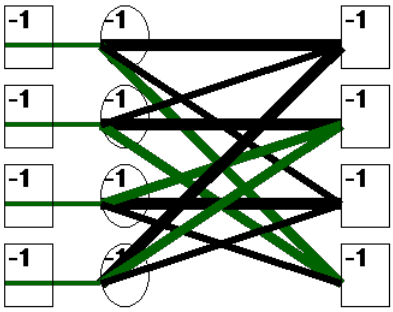


13.1.3 Solver for Instances With Ties

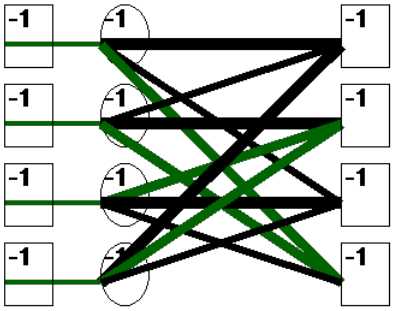
This section covers the class *PopSolver*2, which can solve the popular matching problem for instances with ties. Again, the function $Match()$ is the starting function to find a popular matching. Then, first $Inflate()$ is called to inflate the instance. After that, the function $MaxMatching()$ is called, which tries to find a maximum matching on edges of rank 1. To find such a maximum matching, a simple algorithm based on augmenting paths is used. Via depth-first-search augmenting paths from unmatched nodes are searched, and then matched until no such paths exist. The running time of this is $O(nm)$. The fastest known algorithm for finding maximum-matchings is the Hopcroft-Karp algorithm, which runs in $O(n\sqrt{m})$ though. Thus the procedure used here does not reach the overall running time as described in Section 3.3. But as the implementation is not the main part of this work, the slightly slower running time is acceptable. Furthermore, when implementing the Hopcroft-Karp algorithm one has to be very careful to do it efficiently, to not make it even slower - another reason for this simple algorithm. The function $DetermineTypes()$ then determines the types of all nodes. Since the definition of odd and even is based on the distance from unmatched nodes, from each of these a search is started to find all existing paths, on the way directly the odd and even nodes can be determined. With this information now the reduced graph can be determined, therefor the function $GetReduced()$ is called. In it, according to the characterization in Lemma 8, posts which are odd and unreachable are deleted as candidates for the s-posts. Then the highest ranked even node for each applicant is set as s-post. Thereafter, $DeleteEdges()$ is called, in which edges are deleted connecting two odd nodes are an odd and an unreachable node, which is not possible in a popular matching. Then, again the function $MaxMatching()$ is called to find a maximum matching, but this time edges of any rank are allowed. If the resulting matching is applicant-complete, it is also popular.

The same input as above, $0, 3, 2-1, 3, 0-2, 1, 3-0, 1, 2-1$, is now processed using this solver, which creates the output below. As you can see there exists a popular matching and the resulting images represent the important steps of the algorithm:

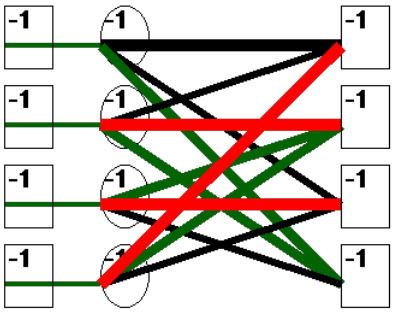
0Start.bmp



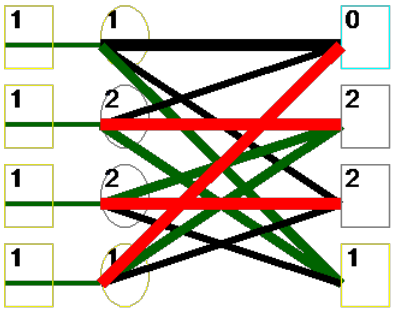
1Inflated.bmp



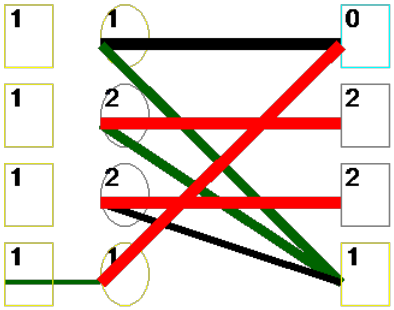
2FirstMax.bmp



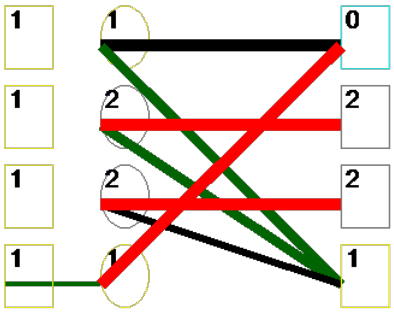
3Types.bmp



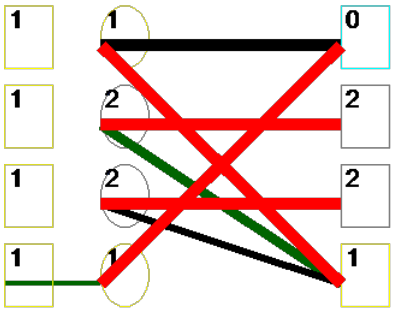
4Reduced.bmp



5EdgesDeleted.bmp

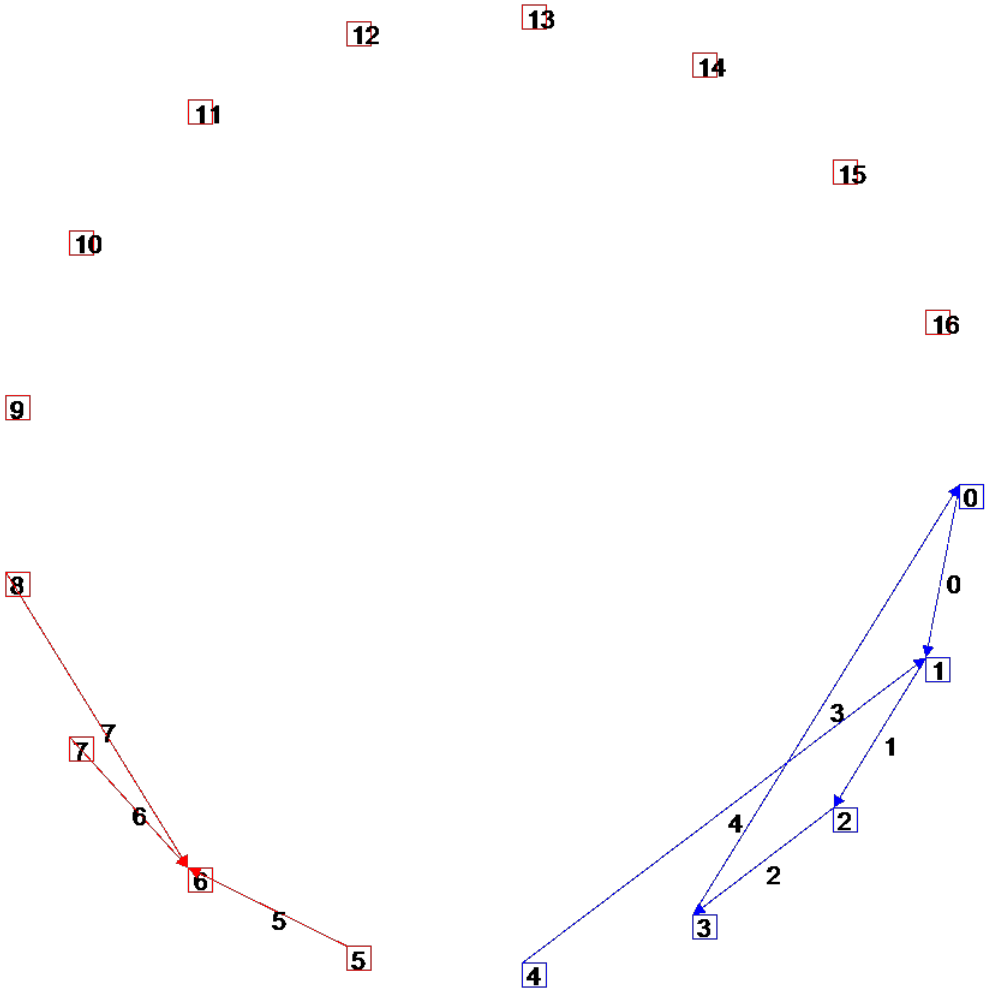


6GeneralMaxMatching.bmp



13.1.4 Creating the Switching Graph

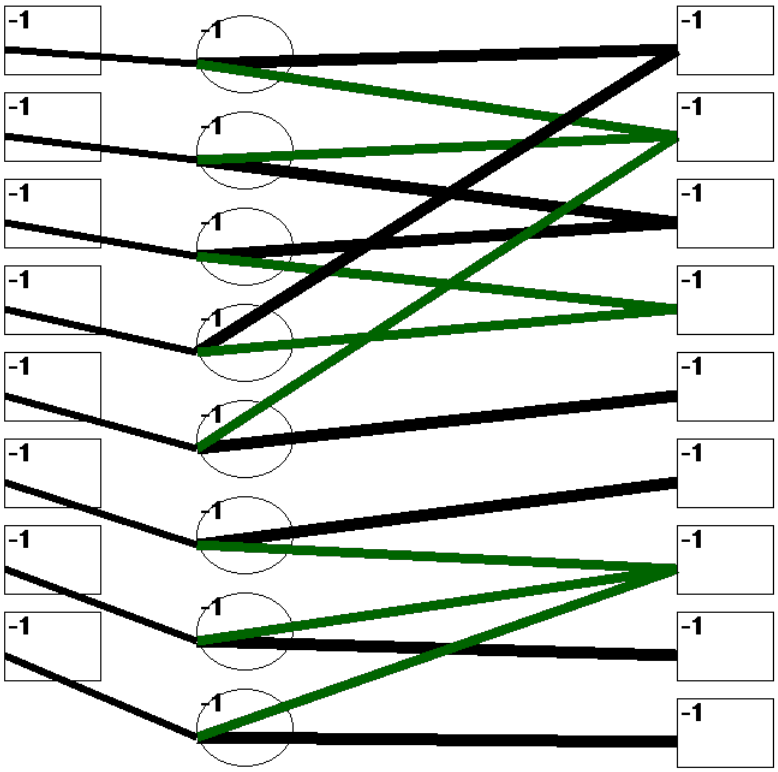
A switching graph is represented as an instance of the class *SwitchingGraph*. It is created in its constructor, to which the underlying instance has to be passed. The posts and edges are created according to the procedure described in Section 4, then $IdentifyComponents()$ is called to identify the components of the switching graph. In it a depth-first-search is used to find the tree and cycle components of the graph. When adding an s-post to a component, a variable is incremented. That way, the function $Count()$ can count the number of popular matchings according to the formula described in Theorem 26.
The example from Figure 1 in Section 4 is programatically solved and displayed, tree components are drawn in red, cycle components in blue:



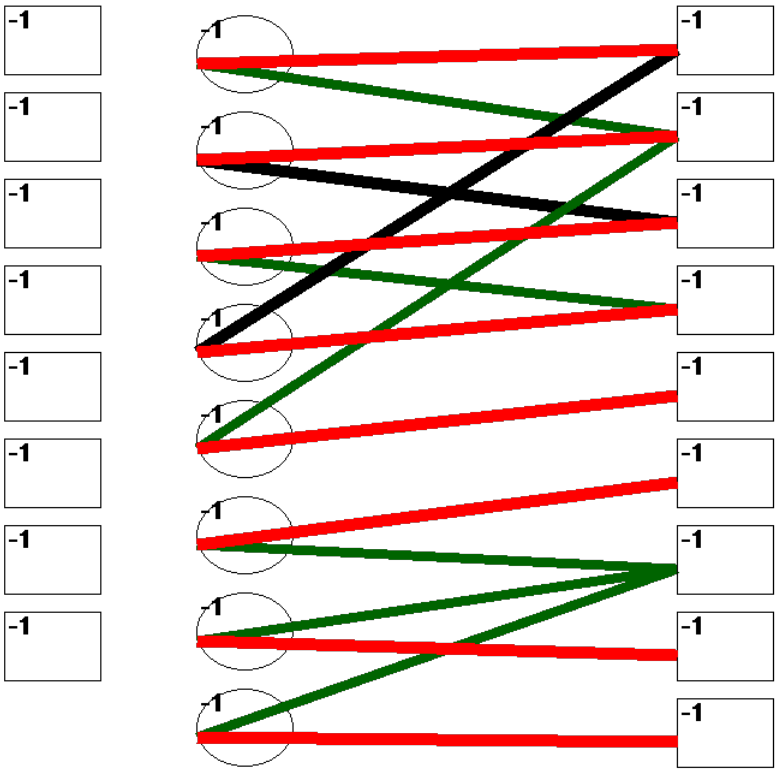
Posts are drawn as rectangles, with their corresponding IDs in the shape. Edges represent the applicants, their IDs are drawn next to them.
In the example in the tree component only post 6 is an s-post, so there exist no switching paths in that component. Since there exists one cycle component, the number of popular matchings for this instance is $2^1 * 1$.

13.1.5 Enumerating Popular Matchings

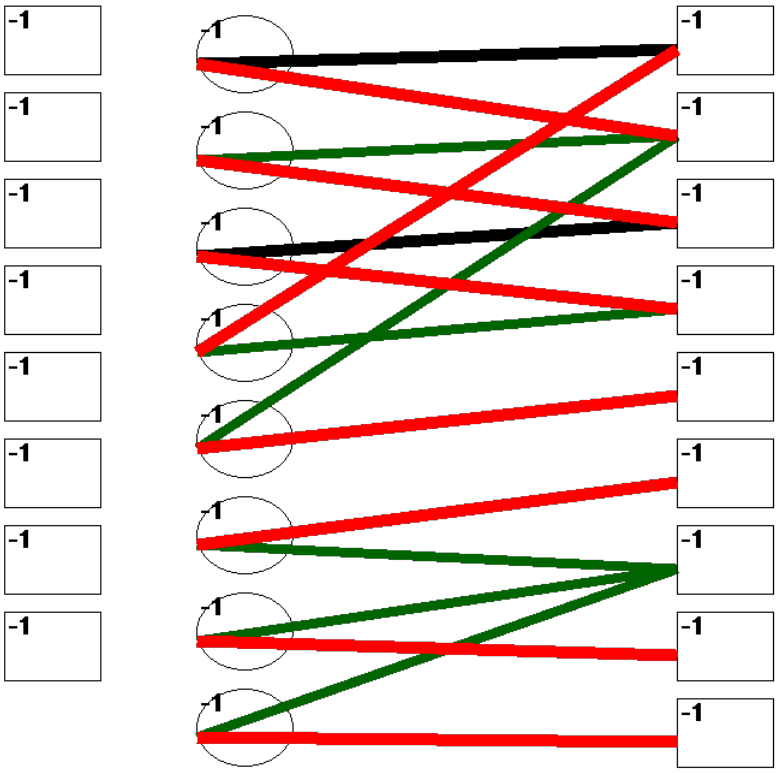
The function $Enumerate()$ enumerates all possible popular matchings of an instance and outputs them graphically in the selected folder. For that, the function $CreateVectors()$ recursively creates all possible vectors of selected switching paths and cycles. Then for each vector, the corresponding switching paths and cycles are applied and the resulting matching is saved.
As mentioned, for the previous example there exist two popular matchings. The instance itself is visualized as:



The matching the solver outputs is this, which is also the first matching which is enumerated ("0.jpg"):



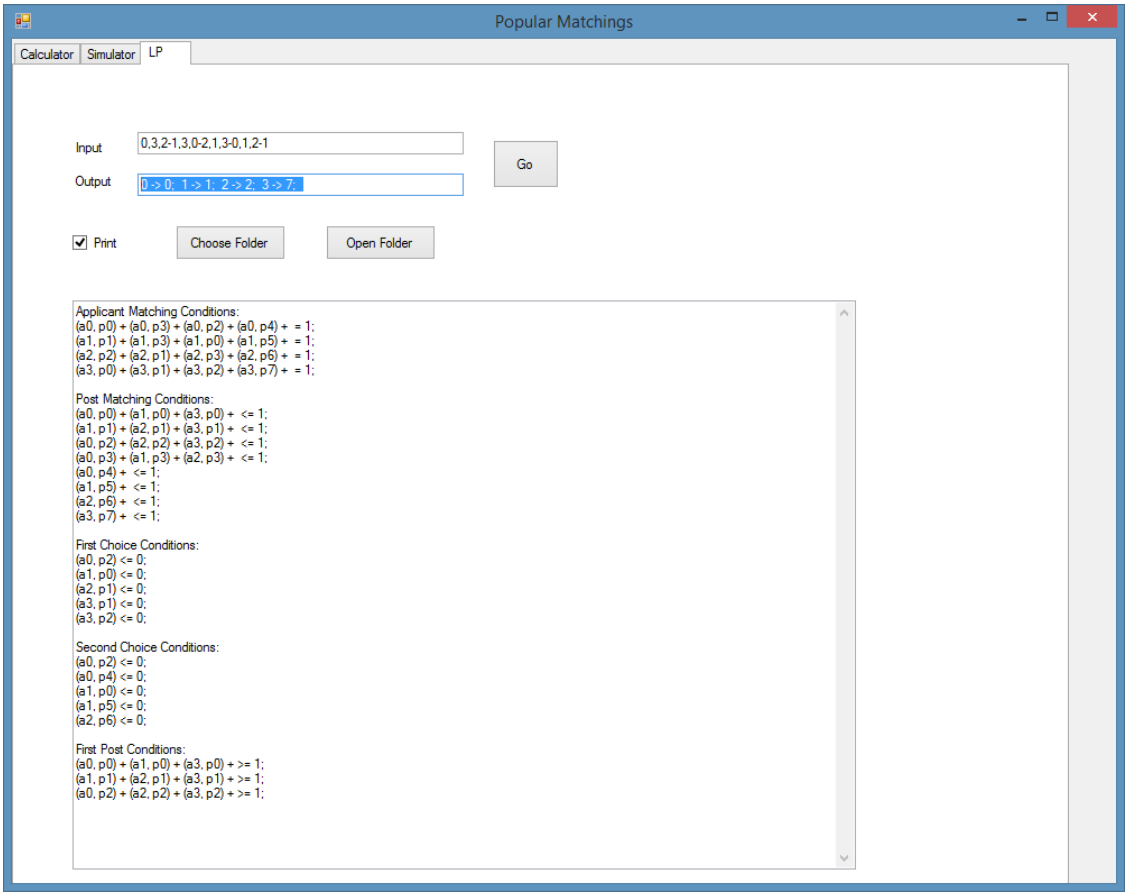
The other popular matching can be created by applying the switching cycle in the one cycle component ("1.jpg"):



13.1.6 LP Solver

This section is about the class *LPSolver*, which is used to solve the popular matching problem using linear programs. Core of it is the Gurobi Optimizer, whose .Net interface is used to solve the actual linear program. As variables serve the edges - they are binary, as not integer numbers have no useful interpretation in the model. Then iteratively the model is created. First for every applicant and post conditions are added, so that the sum of all variables corresponding to edges incident to that applicant or post is less or equal to 1. For applicants we require the sum to be even equal to 1, as we know that all applicants have to be matched. Then for each edge, which is incident to an f-post and has not rank 1, the corresponding variable is restricted to be less or equal to 0, as these edges cannot be taken in a popular matching. Afterwards, all edges incident to an applicant (which essentially are the contents of his priority list) are checked, and only the highest ranked edge connected to a not f-post is allowed, that is, for all variables

corresponding to the other edges a ≤ 0 condition is inserted. Simultaneously, not feasible edges are deleted in the actual instance, so that the instance can be drawn accordingly. Eventually also conditions are inserted to guarantee the matching of all f-posts.
For the example $0,3,2-1,3,0-2,1,3-0,1,2-1$, the output $0->0;1->1;2->2;3->7;$ is generated. If "Print" is enabled, the LP conditions are printed out, they are for this instance:



13.1.7 Enumerate Instances of a Given Size

This section is about the class *Simulator*. With it, all instances of a given size can be created, or a random subset of them.
We here only consider instances with an equal amount of applicants and posts, which we call the instance size, furthermore each post is contained in each applicant's preference list. The function $FindAllMatchings()$ creates and counts all possible instances of a given size n. For this the function $RecPrios()$ is used. It is called recursively with each possible preference list for each applicant except applicant 1. Applicant 1 always gets the same priority list, this way a factor of $n!$ possible permutations of posts in respect

to his preferences does not have to be examined. While creating all instances, they are checked for popularity and the total number is incremented too, thus we eventually get the complete result when paying attention to the permutation factor. Using this approach, counting all instances of size greater 6 will take a very long time. We could reduce the effort by taking out other possible permutations (for example permutations of the posts in general), but this is not implemented, as all this only slows the growths of the effort, but it still remains exponential. This is, because the number of all instances of a size n is given by $(n!)^n$, which is, of course, an extremely quickly growing function. For instance sizes 3 to 5 we get the following result:

n	# not popular instances	# instances
3	6	216
4	24264	331776
5	3211845120	24883200000

Therefore, the program contains functions to estimate the number of instances allowing a popular matching fairly precise, but with a feasible effort.
The function $SimulateRandomInstanes(n, a)$ generates a random instances of size n. Then the factor x/a is calculated, where x is the number of instances not allowing a popular matching. This is then projected to the total number of instances by calculating $(x/a)*n!)^n$. It is unclear though how reliable this number alone is. Therefore the program calculates a 95 % confidence interval for it. This calculation is based on the assumption, that the projected number converges towards a normal distribution with a mean of the actual total number. Thus, a 95 % confidence interval is given by $[\mu - 1.96 * \sigma/\sqrt{N}, \mu - 1.96 * \sigma/\sqrt{N}]$, where μ is the mean, σ the standard deviation and N the number of samples. Since we here only inspect random samples, we have to estimate also μ and σ. For the mean we simply use the average as an estimator $\bar{x}$, for the standard deviation the corrected sample standard deviation $s = \sqrt{\frac{1}{N-1} \sum_{i=1} N(x_i - \bar{x})^2}$. N of course is a in this case, the single x_i we interpret as 0 or 1, depending on whether the generated instance i allows a popular matching or not.
The function $TestCalcConfidenceRow()$ repeats the above for a row of numbers and prints the results in an Excel file. For instance sizes 3 to 18 the results are depicted in Section 7.3.