An Experimental Comparison of Single-Sided Preference Matching Algorithms

Dimitrios Michail
Department of Informatics and Telematics
Harokopio University of Athens
michail@hua.gr

April 20, 2011

Abstract

We experimentally study the problem of assigning applicants to posts. Each applicant provides a preference list, which may contain ties, ranking a subset of the posts. Different optimization criteria may be defined which depend on the desired solution properties. The main focus of this work is to assess the quality of matchings computed by rank-maximal and popular matching algorithms and compare this with the minimum weight matching algorithm which is a standard matching algorithm that is used in practice.

Both rank-maximal and popular matching algorithms use common algorithmic techniques, which makes them excellent candidates for a running time comparison. Since popular matchings do not always exist, we also study the unpopularity of matchings computed by the aforementioned algorithms. Finally, extra criteria like total weight and cardinality are included, due to their importance in practice. All experiments are performed using structured random instances as well as instances created using real-world datasets.

1 Introduction

Consider the scenario where a set of applicants $\mathcal{A}$ has an interest in obtaining a set of posts $\mathcal{P}$ and suppose that associated with each member of $\mathcal{A}$ is a preference list (possibly including ties) comprising a subset of elements of $\mathcal{P}$. A matching of $\mathcal{A}$ to $\mathcal{P}$ is an allocation of each applicant to at most one post such that each post is filled by at most one applicant. Stated differently, it is a matching in the bipartite graph $G = (\mathcal{A} \cup \mathcal{P}, E)$ where E consists of all pairs (a, p) where p belongs in the ordered preference list of a. In order to represent ordered preference lists we assume that each edge $e = (a, p) \in E$ has a *rank* denoting the position of post p in the preference list of a.

In this setting, applicants have preferences over posts but posts are indifferent between applicants. We say that such an instance is a *one-sided preference matching* instance. When both sides of the bipartite instance express preferences, the problem is the *stable marriage* problem. In either case, there has been considerable research in the economics and algorithmics communities concerning these scenarios. The interested reader, for the case of one-sided preference matchings which is the focus of this work, is referred to [25, 12, 28]. The preference matching problem appears in many real-world scenarios such as assigning students to universities, applicants to jobs, papers to referees, etc.

Rank-maximality [13] is one optimization measure that considers applicants collectively. Let us call an edge that is the i-th choice of an applicant, a rank i edge. Rank-maximal matchings first try to maximize the number of rank one edges. Then, given the maximum number of rank one edges, rank two edges are maximized, and so on. Although rank-maximal matchings favor some applicants, which applicants are favored is dependent on the complete setting, not on the individual users' choices.

Popular matchings, first studied by Gärdenfors [7] in the context of the stable marriage problem, try to find matchings which are preferred by the majority of the participants. Given a matching M, an applicant $a \in \mathcal{A}$ is either *unmatched* in M, or *matched* to some post denoted by $M(a)$. We say that an applicant $a \in \mathcal{A}$ *prefers* matching M' to M if (i) a is matched in M' and unmatched in M, or (ii) a is matched in both M and M' and a prefers $M'(a)$ to $M(a)$. M' is *more popular* than M, denoted by $M' \succ M$, if the number of applicants that prefer M' to M exceeds the number of applicants that prefer M to M'. A matching M is *popular* if there is no matching M' that is more popular than M. When only one side has preferences, Abraham et al. [1] gave a polynomial time algorithm to compute a popular matching or to report that none exists. Extensions of the problem have also been studied, see for example [19, 22].

The notion of popular matchings is a specialization of the Condorcet criterion in voting systems [5]. When viewed in such a context, matchings are candidates, applicants vote for matchings and ties are allowed in the preference lists of the voters. The Condorcet criterion announces a winner if that candidate can win a two-candidate election against all other candidates. In voting systems, the Condorcet paradox is a situation where collective preferences can be cyclic. In terms of popular matchings, this paradox translates to the fact that not all instances have a popular matching. For example the complete instance with no ties where each applicant has the exact same preference list, admits no popular matching. The problem resides with the fact that the more popular than relation is not acyclic.

McCutchen [20] defined two quantities in order to measure the unpopularity of a matching. Given any two matchings X, Y in G, let $\Phi(X, Y)$ be the number of applicants who prefer X to Y. Let also $\mathcal{M}$ denote the set of all matchings in G. The *unpopularity margin* of a matching M is defined as

$$g(M) = \max_{M' \in \mathcal{M}} (\Phi(M', M) - \Phi(M, M')).$$

The *unpopularity factor* is defined as

$$u(M) = \max_{M' \in \mathcal{M} \setminus Z(M)} \frac{\Phi(M', M)}{\Phi(M, M')},$$

where $Z(M)$ is the set of all matchings N such that $\Phi(N, M) = \Phi(M, N) = 0$.

In the same paper McCutchen proves that finding a matching that minimizes either of the above two quantities is NP-hard, and also presents two algorithms which given a matching M compute $u(M)$ and $g(M)$ in polynomial time. Huang et al. [11] presented an algorithm which computes a popular matching if one exists or an approximation under certain circumstances. Their algorithm can be viewed as a continuation of the popular matching algorithm of Abraham et al. [1]. However, in the worst case the approximation ratio can be as bad as $\mathcal{O}(n)$, where n is the number of vertices of the instance. It is an open problem to derive an algorithm with a better approximation ratio. On the other hand, one might be willing to compute a mixed matching, that is a probability distribution on the set of all matchings of G. In that case, Kavitha et al.[15] showed that a popular

mixed matching always exists and provided a polynomial time algorithm to find one. The definition of a popular mixed matching extends naturally by using expectation.

The main purpose of this paper is to experimentally study matchings computed by various recent one-sided preference matching algorithms with respect to their unpopularity. On the other hand, since it would be unfair to judge algorithms based solely on the unpopularity, we include additional quality measurements such as cardinality, total rank, maximum rank and running time. We compare several different algorithms for the computation of rank-maximal matchings [13, 23], the algorithm of [1] for the computation of popular matchings, and the algorithm of [11]. While popular matchings seem to be unrelated to rank-maximal matchings, the algorithmic techniques required in order to efficiently compute both types are very much related. Thus, all algorithms are implemented using similar heuristics and graph representations. We briefly describe the algorithms in Section 2. Our code uses LEDA [21] for various data structures and can be found at [17]. In order to have a common point of reference, a minimum weight maximum cardinality matching algorithm is also included in the comparison. This algorithm is a very common choice in practice.

The experimental comparison of the aforementioned algorithm is performed on instances created by three random structured instance generators. All generated problem instances try to mimic different real life situations, while maintaining as few parameters as possible. Section 3 contains a description of the generators. In addition to synthetic datasets we experiment with two real-world datasets: Zillow and NBA. In both cases, the created instances were based on the k-attribute model [4, 3]. Zillow [29] is a website with real estate information, containing 2M records. NBA [24] includes statistics about NBA players since 1973.

Section 4 contains the experiments. Section 4.1 compares the various algorithms based on their running time. Despite the differences between the algorithms, the underlying data structures are very similar. This allows us to identify the bottlenecks of each one. Our main observation here is that the algorithms that are based on a reduction to the maximum weight matching are faster in the case of rank-maximal matchings. Moreover, in several cases it is better to use a simple BFS procedure in order to perform the necessary augmentations instead of using the Hopcroft-Karp algorithm. Section 4.2 goes one step further and compares the matchings computed by the algorithms with respect to their unpopularity factor. Note that different rank-maximal matchings might have different unpopularity factors. We observe that the algorithm of Michail [23] computes matchings with lower unpopularity factors, even when compared to the algorithm of Huang et al. [11] which was specifically designed for this purpose. Finally we compare the various algorithms with respect to the maximum rank, the total rank and the cardinality of the computed solutions.

2 Algorithms

We briefly introduce the algorithms that we use in this work. We try to be short and concise; for more details we refer the reader to the respective papers. Table 1 lists the algorithms together with their running time and space requirements.

2.1 Rank Maximal Algorithms

The algorithm in [13], which we denote as RMM, computes a rank-maximal matching in phases. In the first phase a graph G_1 is constructed from G by using only rank one edges. After computing a maximum matching M_1 in G_1, the algorithm computes the Gallai-Edmonds decomposition [9]. The

Algorithm	Matching Type	Running Time	Space
RMM	Rank-maximal.	$\mathcal{O}(min(r\sqrt{n}, r+n)m)$	$\mathcal{O}(n+m)$
RMM-BFS	Rank-maximal.	$\mathcal{O}((r+n)m)$	$\mathcal{O}(n+m)$
MWM	Rank-maximal.	$\mathcal{O}(min(r\sqrt{n}, r+n)m)$	$\mathcal{O}(n+m)$
MWM-BFS	Rank-maximal.	$\mathcal{O}((r+n)m)$	$\mathcal{O}(n+m)$
EXP	Rank-maximal.	$\mathcal{O}(rn(m+n\log n))$	$\mathcal{O}(r(n+m))$
POP	Popular if exists, nothing otherwise.	$\mathcal{O}(\sqrt{n}m)$	$\mathcal{O}(n+m)$
BND	Popular if exists, bounded unpopularity otherwise.	$\mathcal{O}(min(r\sqrt{n}, r+n)m)$	$\mathcal{O}(n+m)$
MINW	Maximum-cardinality minimum-weight.	$\mathcal{O}(n(m+n\log n))$	$\mathcal{O}(n+m)$

Table 1: Algorithms. n is the number of vertices, m the number of the edges and r the maximum rank of any edge in the input.

Gallai-Edmonds decomposition partitions the vertices of the graph into three disjoint sets, based on whether they are reachable or not by even or odd length alternating paths from free vertices. This decomposition provides important information such as (a) which vertices are matched in all maximum matchings of G_1, and (b) which edges of rank one are never used in any maximum matching of G_1. Based on this information the algorithm prunes from G several higher rank edges as well as several rank one edges. The next step of the algorithm is to augment graph G_1 with all surviving rank two edges and arrive at a graph G_2 where by augmentations a second maximum matching M_2 is computed. Again by computing the decomposition additional edges are pruned and this procedure is iterated until all ranks have been added to the instance, or there are no more edges left. Each phase is therefore one maximum matching computation, one BFS traversal in order to compute the Gallai-Edmonds decomposition and a linear scan of the edges for the pruning step. The maximum matching computation in phase $i+1$ is done using the Hopcroft and Karp [10] algorithm, starting from an already computed partial matching, in time $\mathcal{O}(min(\sqrt{n}, |M_{i+1}| - |M_i| + 1) \cdot m)$ where M_i is the maximum matching at phase i. Here m denotes the total number of edges of the instance.

A considerable improvement in the running time can be achieved by rearranging the graph representation based on the edge ranks. We use an adjacency list representation where each node's edges are sorted based on their ranks in non-decreasing order. This allows us to quickly augment the matching in phase i using all remaining edges with rank $\leq i$ without traversing any edges of higher rank. The algorithm's total running time is $\mathcal{O}(min(r\sqrt{n}, r+n) \cdot m)$ where r is the maximum rank of any edge in the input.

The choice of the algorithm that is used for the augmentations is likely to make a significant difference. While the Hopcroft and Karp [10] algorithm has the best worst-case performance, using BFS for augmentations (essentially the Ford-Fulkerson algorithm) can perform better in practice. Setubal [26] suggests to use BFS in the case of repeatedly solving small instances, up to a thousand vertices, as subproblems. The RMM contains several augmentations which are performed in a partially solved instance. Moreover, as the phase number increases the augmenting paths become fewer and longer. The reason is that augmenting paths must preserve the number of matched edges of all previously considered ranks. The version of RMM which uses a simple BFS procedure in order to perform augmentations is denoted as RMM-BFS.

2.1.1 Reduction to Maximum Weight Matching

The rank-maximal matching problem can also be solved by a reduction to the maximum weight matching problem, based on the fact that any matching has cardinality less than n. The idea of the reduction is to assign weights to edges such that no collection of at most n larger rank edges can outweigh one smaller rank edge. Let $r \leq n$ be the maximum rank of any edge. Then one such possible assignment is to give a weight of n^{r-i} to an edge of rank i. The problem with this approach is that edge weights can be as large as n^{r-1} which is not polynomial in the input size. This affects the running time and the space requirements of the algorithm. We have implemented this reduction using LEDA's $\mathcal{O}(n(m+n\log n))$ maximum weight matching [2] implementation. We denote this implementation as EXP. Due to the extra cost of arithmetic, the space requirement is $\mathcal{O}(r(n+m))$ and the running time $\mathcal{O}(rn(m+n\log n))$. In our implementation, as a heuristic, we try to estimate an upper bound on the maximum weight that an edge might require. Depending on this bound we either run the maximum weight matching using a long data type or using arbitrary precision arithmetic.

The maximum weight reduction can also be performed implicitly [23]. The idea is to use the decomposition theorem of Kao et al. [14] and an implicit representation of the weights using dual variables. This implicit representation requires space polynomial in the input size. The algorithm works again in phases and performs only maximum cardinality matching computations. It can also be viewed as a scaling algorithm specialized for exponentially increasing weights. Note, however, that applying a more general scaling algorithm would result in slower running times. For example the scaling algorithm [8, 6] for the weight matching problem can be implemented such that all arithmetic is performed on numbers with $\mathcal{O}(\log n)$ bits, independent of the edge weights. The resulting running time would be $\mathcal{O}(r\sqrt{n}m\log n)$. The algorithm of [23] which we denote as MWM needs $\mathcal{O}(min(r\sqrt{n}, r+n)m)$ running time and linear space. MWM uses the Hopcroft-Karp algorithm for the maximum cardinality matching computation. Let MWM-BFS be the variant of MWM which uses BFS in order to perform the augmentations.

2.1.2 Initialization Heuristics

The implementations of RMM, RMM-BFS, MWM and MWM-BFS use the greedy heuristic before computing the initial matching M_1. While the choice of the heuristic, which selects an initial (not necessarily maximum) matching, has been shown [16] to influence performance of matching algorithms, we expect it to have a lesser effect in this particular setting. The reason is that between phases there is already a precomputed matching which needs to be augmented to a maximum matching. Moreover, implementing stronger heuristics would require special attention from an engineering point of view. The reason is that any data structures used by such a heuristic, should be initialized only once as a whole and then only partially in every phase of the algorithm, something which requires extra bookkeeping. Even if this not the case, this issue deviates significantly from the main focus of this work, and therefore is not explored any further.

2.2 Algorithms for Popular Matchings

Let G_1 denote the graph $G(\mathcal{A}\cup\mathcal{P}, E_1)$ where E_1 is the set of edges of rank one, i.e. all first choices. The popular matching algorithm of Abraham et al. [1] is based on the observation that in any popular matching M the rank one edges must be a maximum cardinality matching in G_1.

Using this observation the set of useful edges is restricted in two categories: (a) for each applicant all her first choices, i.e., rank one edges (b) for each applicant the set of most preferred choices in her preference list that do not interfere with the maximality of the rank one edges. The term *interference* here means that if such an edge is matched, then the rank one edges of the matching would not be a maximum cardinality matching in G_1. A popular matching is an applicant complete rank-maximal matching in this reduced instance; we may enforce this condition by adding last resort posts to the instance. Computing which posts do not interfere with the maximality of rank one edges is done as described in the rank-maximal matching algorithm using the Gallai-Edmonds decomposition. The algorithm has a running time of $\mathcal{O}(\sqrt{n}m)$ since it is a rank-maximal matching computation with edges of only two ranks. We call our implementation POP.

Popular matchings do not always exist. The algorithm in [11], however, is always guaranteed to produce a matching, even if the instance does not admit a popular one. The algorithm can be viewed as a continuation of the popular matching algorithm. If the reduced instance described in the previous paragraph does not admit an applicant complete matching, the algorithm continues by adding a third set of edges which is the next most preferred choices of each applicant which do not interfere with the maximality of the matching computed with only the first two edge sets. This constitutes the 3rd phase and can be again determined by the Gallai-Edmonds decomposition. We continue this process until we compute an applicant complete matching, a guaranteed outcome due to the existence of last resort posts. Assuming the algorithm terminates in round k, the computed matching M has $u(M) \leq k - 1$. This algorithm is designated as BND.

3 Instances

We next describe our structured instance generators as well as the real-world datasets used in our experiments.

3.1 Highly Correlated

Highly correlated instances are problem instances where most of the applicants have a global and consistent knowledge of the posts' reputation. Let n_a be the number of applicants and n_p the number of posts. Posts are already ranked from an outside source, in some particular order $p_1, p_2, \ldots, p_{n_p}$, a ranking which is well known to all applicants and is gladly accepted. The goal is to have preferences which are based on the intuition that if two posts p_i and p_j are far apart from each other in the global ranking then it should be highly likely that they are far apart in each applicant's ranking as well.

We model such instances based on the following parameters: (a) number of applicants n_a and number of posts n_p, (b) a probability d that an edge exists in the instance, (c) a probability t that the rank of an edge $e = (a, p)$ is the same with the rank of its predecessor in a's adjacency list. Instances are created in the following way. For each applicant a we randomly sample with probability d all posts, resulting in a subset $\{p'_1, p'_2, \ldots\}$ which is subsequently sorted based on the total order $p_1, p_2, \ldots, p_{n_p}$. For each post p'_i we create an edge (a, p'_i). Ranks are assigned as follows. Edge (a, p'_1) is first assigned a rank of one. Then with probability t the edge (a, p'_{i+1}) is assigned the same rank as (a, p'_i) and with probability $1 - t$ is given one higher rank than (a, p'_i).

These instances are denoted as HC and are referred to as highly correlated since preferences lists have a high degree of similarity among applicants.

3.2 Variable Size Exponential

In several real-world assignment situations, the number of available positions is relatively low compared to the applicants. Moreover, there exists a total ordering $p_1, p_2, \ldots, p_{n_p}$ of the positions which is known in advance to all applicants (perhaps from an organization responsible for ranking the posts). Since people usually have a very strong opinion about their top choices and relatively weaker opinion about their last choices, this total ordering is strict at the beginning but allows equality towards the end. Based on this observation we divide posts into clusters of exponentially increasing size. We maintain the following invariant. If an applicant assigns rank i to a post of cluster P_j, then the applicant ranks all posts of the cluster P_j as rank i.

The generator which we call VS has the following parameters: (a) the number of applicants n_a and posts n_p, (b) an instance density parameter d, and (c) a parameter λ controlling the distribution of posts into clusters. Our goal is now to add an edge from an applicant $a \in A$ to the first $\lfloor n_p \cdot d \rfloor$ posts as follows. We initially add edge (a, p_1) with rank one. When adding an edge (a, p_i) we decide the rank based on the parameter λ. With probability $e^{-\lambda i}$ the rank is increased by one and with probability $1 - e^{-\lambda i}$ the same rank is used.

3.3 Uniform Random

These instances have (a) the number of applicants n_a and posts n_p, (b) an instance density parameter d, and (c) a probability t that the rank of an edge $e = (a, p)$ is the same with the rank of its predecessor in a's adjacency list. Each applicant selects a preference list of size $\lfloor n_p \cdot d \rfloor$ randomly with uniform probability. Assume that applicant a has the preference list $p_1, p_2, \ldots, p_{\lfloor n_p \cdot d \rfloor}$. Ranks are assigned as follows. Edge (a, p_1) is first assigned a rank of one. Then with probability t the edge (a, p_{i+1}) is assigned the same rank as (a, p_i) and with probability $1 - t$ is given one higher rank. These instances are referred as UNI.

3.4 Real-world Datasets

In addition to the aforementioned instances, we also experimented with artificially constructed instances based on two real world datasets [29, 24]. The same setting as [27] was used, where preference matchings are considered in the context of databases. It is quite common in such practical situations, when trying to describe preference matching instances, to use the k-attribute model [4, 3]. In this model, assuming that posts have k attributes, we associate them with points in $\mathbb{R}^k$. Each applicant's ranking of the posts is defined by a linear function of these attributes, and then her preference list is determined by projecting the post's points onto some line. From a computational point of view, such a model is quite attractive, due to its compactness in size.

Zillow is a website with real estate information, containing 2M records with attributes. NBA includes statistics about 12278 NBA players since 1973. In the following we use the term *real estate* dataset in order to refer to this dataset. In the case of the real estate dataset we used three attributes, namely, number of bathrooms, number of bedrooms, and living area. We selected three important offensive attributes in NBA: points, offensive rebounds, and assists.

The parameters that were used is (a) number of applicants n_a, (b) number of posts n_p and (c) length d of the preference list of each applicant. The instances were created by sampling n_p applicants uniformly at random from the appropriate dataset. Then n_a applicants were constructed by choosing the coefficients again uniformly at random, with the additional restriction that they are normalized in order to sum up to one. For each of these applicants the first d choices were

added as edges. As an example, assume that we have a 3 dimensional space with three posts $p_1 = (1, 3, 4)$, $p_2 = (4, 1, 0)$, $p_3 = (0, 0, 10)$ and two applicants $a_1 = (0.5, 0.5, 0)$, $a_2 = (0.1, 0.8, 0.1)$. We can easily see that a_1 has the following preference list p_2, p_1, p_3 while a_2 has p_1, p_2, p_3. Ties are resolved arbitrarily.

4 Experiments

All experiments were performed on an AMD Opteron^{TM} Processor 1214 running at 2.2Ghz with 1MB cache and 4GB of main memory. The system was running Linux while both LEDA and our code were compiled with a 64-bit version of GCC 4.1. In all experiments we generated 100 instances for each set of parameters and averaged the results. Note that the goals of this section are twofold: (a) to observe the different behavior of the algorithms on the same instances, and (b) to assess the difficulty of each instance category. Some preliminary experiments, with HC and random instances, were also reported in [11].

4.1 Running Time

In this section we compare the performance of the different rank-maximal matching algorithms (RMM, RMM-BFS, MWM, MWM-BFS, EXP), the algorithms for popular matchings (POP, BND) and the minimum weight matching algorithm (MINW). The algorithms POP and MINW are included as reference points. By performing only a constant number of rounds, the first one achieves a worst-case running time of $\mathcal{O}(\sqrt{n}m)$, but might not return a matching in the case that the instance does not admit a popular matching. The second one is used very often in practice and has excellent performance. The default implementation of MINW in LEDA, used in this work, has a worst-case running time of $\mathcal{O}(n \cdot (m + n\ log\ n))$ [2].

The execution times in seconds can be found in Table 2 and Table 3, categorized by the different instances. The first column denotes the type of instance by presenting the values of the various parameters. Recall that we use n_a and n_p to denote the number of applicants and posts respectively, i.e., $n = n_a + n_p$. Our first step is to categorize the different instances based on their difficulty with respect to the running times. HC instances are the most difficult due to the fact that all applicants prefer the same posts in the same order. Houses in the real estate dataset are all from one particular region and thus have similar characteristics. This correlation causes real estate instances to resemble HC instances. On the other hand, NBA players' performance does not relate easily across different teams and thus the NBA dataset is more evenly distributed. Subsequently, NBA instances are easier than the corresponding instances in the real estate case. VS instances are much easier to solve. This is due to the fact that applicants are willing to settle between different posts, recall that preferences are partitioned in clusters of exponentially increasing sizes. Finally, UNI instances are the easiest of them all.

For HC, VS and UNI instances we performed experiments for $n_a = n_p = 250$ and $n_a = n_p = 500$. Both sparse and dense instances are used by setting the parameter d controlling edge density to 0.02, 0.5 and 1.0. Parameter t, denoting the probability of ties, is restricted to $t = 0.0$, $t = 0.25$ and $t = 0.5$. Larger values result in easier instances, with the extreme case of $t = 1.0$ where all ranks are the same and thus instances are solvable by any maximum cardinality matching computation. In the case of the VS instances the existence of ties depends on the cluster sizes which are in turn dependent on the parameter λ where we use values 0.05, 0.1 and 0.25. The parameter d in the

	RMM	RMM-BFS	MWM	MWM-BFS	EXP	POP	BND	MINW
$(n_a = n_p, d, t)$	Highly Correlated Random (HC) Instances [sec]							
$(250, 0.02, 0.0)$	0.02322	0.00358	0.00566	0.00544	0.00092	0.00827	0.02477	0.00064
$(250, 0.02, 0.25)$	0.00414	0.00326	0.00340	0.00321	0.00092	0.00615	0.00636	0.00064
$(250, 0.02, 0.5)$	0.00380	0.00311	0.00372	0.00355	0.00091	0.00547	0.00544	0.00063
$(250, 0.5, 0.0)$	47.082	7.090	0.858	0.802	0.403	0.103	30.007	0.096
$(250, 0.5, 0.25)$	0.374	0.294	0.109	0.069	0.601	0.103	0.710	0.094
$(250, 0.5, 0.5)$	0.295	0.227	0.095	0.064	0.526	0.102	0.361	0.088
$(250, 1.0, 0.0)$	112.263	39.972	3.182	2.979	5.448	0.225	88.832	0.573
$(250, 1.0, 0.25)$	0.885	0.863	0.285	0.151	4.959	0.220	2.215	0.508
$(250, 1.0, 0.5)$	0.696	0.629	0.236	0.137	4.474	0.219	0.947	0.472
$(500, 0.02, 0.0)$	0.44654	0.02175	0.03100	0.02942	0.00376	0.03401	0.38385	0.00281
$(500, 0.02, 0.25)$	0.04751	0.01535	0.01133	0.01019	0.00379	0.03615	0.04159	0.00278
$(500, 0.02, 0.5)$	0.02552	0.01429	0.01222	0.01108	0.00367	0.03246	0.03226	0.00271
$(500, 0.5, 0.0)$	918.548	223.601	6.874	6.386	5.622	0.510	693.710	0.713
$(500, 0.5, 0.25)$	3.142	2.030	0.718	0.323	9.217	0.501	24.043	0.751
$(500, 0.5, 0.5)$	2.433	1.456	0.596	0.299	8.357	0.494	9.075	0.727
$(500, 1.0, 0.0)$	2058.951	762.307	24.643	23.232	61.487	1.051	1735.360	4.374
$(500, 1.0, 0.25)$	7.046	5.950	1.955	0.662	47.446	1.006	56.567	3.908
$(500, 1.0, 0.5)$	5.352	4.152	1.529	0.603	42.164	0.988	22.130	3.738
$(n_a = n_p, d, \lambda)$	Variable Exponential (VS) Instances [sec]							
$(250, 0.02, 0.05)$	0.00637	0.00348	0.00335	0.00317	0.00092	0.00785	0.00780	0.00064
$(250, 0.02, 0.1)$	0.00498	0.00335	0.00337	0.00318	0.00093	0.00745	0.00749	0.00063
$(250, 0.02, 0.25)$	0.00326	0.00316	0.00341	0.00326	0.00092	0.00510	0.00507	0.00063
$(250, 0.5, 0.05)$	0.174	0.160	0.074	0.062	0.204	0.090	0.182	0.057
$(250, 0.5, 0.1)$	0.145	0.132	0.105	0.096	0.138	0.090	0.221	0.051
$(250, 0.5, 0.25)$	0.137	0.114	0.133	0.123	0.064	0.151	0.275	0.047
$(250, 1.0, 0.05)$	0.984	0.336	0.294	0.257	0.328	0.181	0.343	0.126
$(250, 1.0, 0.1)$	1.567	0.272	0.410	0.374	0.165	0.179	0.448	0.096
$(250, 1.0, 0.25)$	1.619	0.222	0.416	0.389	0.104	0.545	0.549	0.078
$(500, 0.02, 0.05)$	0.05045	0.01547	0.01106	0.00994	0.00378	0.03462	0.05266	0.00276
$(500, 0.02, 0.1)$	0.03557	0.01479	0.01101	0.00980	0.00375	0.03596	0.03621	0.00278
$(500, 0.02, 0.25)$	0.01652	0.01371	0.01053	0.00959	0.00359	0.02299	0.02300	0.00266
$(500, 0.5, 0.05)$	1.174	0.707	0.611	0.575	1.005	0.396	2.239	0.316
$(500, 0.5, 0.1)$	1.384	0.617	0.733	0.702	0.708	0.395	3.130	0.276
$(500, 0.5, 0.25)$	1.252	0.558	0.707	0.675	0.499	1.536	2.115	0.245
$(500, 1.0, 0.05)$	19.138	1.436	1.935	1.723	1.133	0.781	2.881	0.558
$(500, 1.0, 0.1)$	20.868	1.163	2.112	1.945	0.635	0.781	2.965	0.425
$(500, 1.0, 0.25)$	17.080	0.954	1.905	1.793	0.461	4.554	4.540	0.337
$(n_a = n_p, d, t)$	Uniform Random (UNI) Instances [sec]							
$(250, 0.02, 0.0)$	0.00275	0.00349	0.00389	0.00352	0.00107	0.00487	0.00488	0.00154
$(250, 0.02, 0.25)$	0.00301	0.00350	0.00419	0.00366	0.00113	0.00502	0.00500	0.00151
$(250, 0.02, 0.5)$	0.00323	0.00350	0.00467	0.00394	0.00135	0.00514	0.00511	0.00151
$(250, 0.5, 0.0)$	0.090	0.134	0.119	0.075	0.103	0.101	0.101	0.052
$(250, 0.5, 0.25)$	0.084	0.128	0.110	0.070	0.085	0.099	0.099	0.049
$(250, 0.5, 0.5)$	0.080	0.123	0.096	0.064	0.091	0.097	0.097	0.047
$(250, 1.0, 0.0)$	0.199	0.286	0.312	0.158	0.245	0.220	0.221	0.109
$(250, 1.0, 0.25)$	0.190	0.275	0.279	0.147	0.256	0.214	0.213	0.105
$(250, 1.0, 0.5)$	0.181	0.264	0.235	0.134	0.282	0.208	0.208	0.101
$(500, 0.02, 0.0)$	0.01226	0.01650	0.01449	0.01157	0.00652	0.01747	0.01757	0.00608
$(500, 0.02, 0.25)$	0.01245	0.01597	0.01559	0.01195	0.00784	0.01835	0.01873	0.00587
$(500, 0.02, 0.5)$	0.01245	0.01555	0.01718	0.01257	0.01080	0.01974	0.01946	0.00575
$(500, 0.5, 0.0)$	0.485	0.629	0.801	0.364	0.556	0.494	0.496	0.264
$(500, 0.5, 0.25)$	0.456	0.610	0.713	0.339	0.623	0.478	0.477	0.252
$(500, 0.5, 0.5)$	0.428	0.581	0.590	0.307	0.735	0.468	0.467	0.241
$(500, 1.0, 0.0)$	1.058	1.313	2.282	0.747	1.301	1.044	1.046	0.552
$(500, 1.0, 0.25)$	0.983	1.264	1.951	0.688	1.402	1.011	1.009	0.523
$(500, 1.0, 0.5)$	0.924	1.219	1.551	0.626	1.621	0.985	0.978	0.503

Table 2: Running Time in Seconds for Various Algorithms. (n_a = number of applicants, n_p = number of posts, d = density, t = tie probability, λ = distribution parameter)

	RMM	RMM-BFS	MWM	MWM-BFS	EXP	POP	BND	MINW
$(n_a = n_p, d)$				NBA Instances [sec]				
$(250, 10)$	0.01644	0.00836	0.00655	0.00583	0.00183	0.00917	0.01303	0.00140
$(250, 125)$	0.348	0.267	0.110	0.073	0.380	0.101	0.249	0.105
$(250, 250)$	0.868	0.831	0.314	0.166	3.579	0.209	0.826	0.432
$(500, 10)$	0.09333	0.01641	0.01406	0.01288	0.00366	0.03556	0.04745	0.00280
$(500, 250)$	3.463	1.555	0.678	0.356	5.198	0.479	2.462	0.840
$(500, 500)$	7.044	4.958	1.878	0.770	30.276	0.985	10.922	3.309
$(n_a = n_p, d)$				Real Estate Instances [sec]				
$(250, 10)$	0.08648	0.01042	0.01160	0.01096	0.00184	0.01095	0.05794	0.00134
$(250, 125)$	11.004	1.171	0.560	0.390	0.346	0.101	4.957	0.090
$(250, 250)$	33.287	6.178	2.584	2.072	4.478	0.219	17.004	0.478
$(500, 10)$	0.35340	0.01969	0.02633	0.02500	0.00361	0.03340	0.26010	0.00274
$(500, 250)$	219.684	12.369	5.367	3.535	4.698	0.498	92.640	0.653
$(500, 500)$	499.709	92.554	22.641	22.881	42.340	1.033	263.262	3.448

Table 3: Running Time in Seconds for Various Algorithms. (n_a = number of applicants, n_p = number of posts, d = degree of each applicant)

case of the NBA and real estate instances denotes the degree of each applicant. We use $d = 10$ for sparse instances, $d = n_p/2$ and $d = n_p$ for dense instances.

We begin by assessing the performance of rank maximal matching algorithms. The first observation is that in almost all cases the variants which use a simple BFS procedure in order to find augmenting paths perform significantly better than the Hopcroft-Karp variant. MWM-BFS is faster than MWM in all cases but one. The exception is the real estate instances with $n_a = n_p = 500$ and degree of each applicant equal to 500. Even in this case, however, the running time is almost the same. RMM-BFS is faster than RMM except in the UNI instances. Comparing RMM and MWM we see that MWM behaves better in the most difficult instances while RMM takes the lead in easier instances like UNI. The situation becomes less clear, when their BFS variants are considered. Nevertheless, the MWM-BFS scales better that the RMM-BFS.

If all rank maximal matching variants are included in a single comparison there is no clear distinction. However, either MWM-BFS or EXP behave better than the rest; there are two exceptions (a) in VS instances for $n_a = n_p = 500, d = 0.5$ and $\lambda = 0.1$ and (b) in real estate instances for $n_a = n_p = 500$ and degree $d = 500$. It seems that using the reduction to the maximum weight matching, either implicitly or explicitly, is the best way to compute rank maximal matchings. The performance of EXP directly depends on the magnitude of the edge weights, which in turn depends on the number of distinct ranks. In practice, the best choice would be to use a hybrid algorithm which depending on the number of distinct ranks would switch from one algorithm to the other. The threshold on the number of different ranks, should be computed experimentally depending on the particular problem instances.

Recall that MWM and MWM-BFS have better worst case complexity than EXP. The reason that EXP in some cases outperforms both MWM and MWM-BFS algorithms, is that they perform a number of rounds which depends on the number of distinct ranks. The dependency on the ranks of EXP is only in the magnitude of the edge weights. Moreover, the extra $\mathcal{O}(n)$ factor in the complexity of the EXP is based on the observation that the algorithm might need to perform arithmetic with numbers that are as large as $\Theta(n^r)$ where $r \leq n$. However, the best case complexity of implementations of arbitrary precision arithmetic depends on the magnitude of the actual numbers involved in each arithmetic operation. In sparse instances, due to the very few distinct ranks, EXP

is much faster than the rest, achieving similar performance to MINW. The same happens in VS instances with large values of λ, where the number of distinct is again small. On the other hand, EXP behaves poorly in dense instances where the best performance with respect to execution times is achieved by MWM-BFS.

The performance of the BND algorithm resembles the performance of RMM. The reason for this is that both algorithms perform the same operations, i.e., (a) compute in each round the Gallai-Edmonds decomposition in order to decide which edges are still relevant or can be pruned from the instance, (b) prune these edges, and (c) augment the matching to a new maximum. The main difference of the two algorithms is the selection of the edges that are going to be considered in each phase. RMM chooses edges of the next rank while BND selects edges is a more dynamic fashion. The high computational cost of the augmentations is responsible for the performance of these algorithms. At the beginning and at the end of the algorithm this computation is relatively inexpensive. During the middle phases, however, it becomes significantly more expensive since there are still a lot of free vertices while augmenting paths are long in length. In general, any variations between RMM and BND are due to the fact that the number of rounds is dependent on the particular preferences. Our implementation of BND uses the Hopcroft-Karp algorithm for the augmentations. Nevertheless, we expect to see a performance similar to RMM-BFS if we replace the augmentations by a BFS procedure.

4.2 Unpopularity and Other Criteria

Using only running times for the comparison would result, as expected, in choosing MINW as the best choice. This section considers additional properties such as the unpopularity factor [20], in order to compare the different preference matching algorithms. The unpopularity factor of a matching M is finite if the matching is pareto efficient, i.e., we cannot change M to make someone better off without making anyone else worse off. Intuitively it captures the largest alternating path that we can find such that all applicants get promoted to better choices at the expense of the last applicant who becomes free. A matching is popular if and only if its unpopularity factor is at most one.

Tables 4 and 5 contain the unpopularity factors of the matchings computed by the various algorithms. Algorithm POP is not included as (a) it may not return a matching and (b) the first phases of BND return a popular matching if one exists. Although the unpopularity factor is always an integer, we compute averages over 100 instances, and thus both tables contain fractional numbers. The results of Tables 4 and 5 suggest that popular matchings existed in very few instances. For a detailed study of the existance of popular matchings in random graphs the interested reader is referred to [18]. The first observation is that MINW is never the algorithm which computes the matching with the lowest unpopularity factor and is several times the one with the worst unpopularity factor. Furthermore, Tables 4 and 5 suggest that the ranking of instances based on their difficulty with respect to the running time is preserved also with respect to the unpopularity factor. Thus, HC instances are the most difficult followed by real estate, NBA, VS and finally UNI instances. This is natural as, at least in the case of BND, the unpopularity factor of the computed matching depends on the number of rounds that the algorithm requires.

Different rank-maximal matchings, while having the same signature (number of edges of each rank) and therefore also the same cardinality, may have different unpopularity factors. Comparing algorithms with their BFS counterparts, we observe that switching from RMM to RMM-BFS and from MWM to MWM-BFS does not result in any significant change in unpopularity. Moreover,

	RMM	RMM-BFS	MWM & MWM-BFS	EXP	BND	MINW
$(n_a = n_p, d, t)$			Highly Correlated Random (HC) Instances			
$(250, 0.02, 0.0)$	5.00	5.00	5.00	5.00	5.00	5.00
$(250, 0.02, 0.25)$	1.38	1.38	1.38	1.38	1.38	1.38
$(250, 0.02, 0.5)$	1.00	1.00	1.00	1.00	1.00	1.00
$(250, 0.5, 0.0)$	125.00	125.00	125.00	125.00	125.00	125.00
$(250, 0.5, 0.25)$	49.28	49.77	12.80	49.12	28.01	54.28
$(250, 0.5, 0.5)$	31.25	31.54	10.38	31.26	16.43	35.45
$(250, 1.0, 0.0)$	249.00	249.00	249.00	249.00	249.00	249.00
$(250, 1.0, 0.25)$	104.93	105.56	29.00	104.30	54.79	114.78
$(250, 1.0, 0.5)$	68.31	68.88	22.65	68.27	31.71	76.76
$(500, 0.02, 0.0)$	10.00	10.00	10.00	10.00	10.00	10.00
$(500, 0.02, 0.25)$	2.92	2.97	2.74	2.98	2.25	3.03
$(500, 0.02, 0.5)$	1.54	1.54	1.54	1.54	1.54	1.54
$(500, 0.5, 0.0)$	250.00	250.00	250.00	250.00	250.00	250.00
$(500, 0.5, 0.25)$	98.59	99.00	18.31	98.12	50.14	108.69
$(500, 0.5, 0.5)$	62.79	63.38	14.69	62.61	28.90	71.52
$(500, 1.0, 0.0)$	499.00	499.00	499.00	499.00	499.00	499.00
$(500, 1.0, 0.25)$	207.18	208.71	41.16	206.07	99.44	226.69
$(500, 1.0, 0.5)$	135.28	135.65	31.70	134.86	56.71	152.85
$(n_a = n_p, d, \lambda)$			Variable Exponential (VS) Instances			
$(250, 0.02, 0.05)$	2.09	2.12	2.15	2.12	2.00	2.14
$(250, 0.02, 0.1)$	1.85	1.85	1.85	1.85	1.85	1.85
$(250, 0.02, 0.25)$	1.01	1.01	1.01	1.01	1.01	1.01
$(250, 0.5, 0.05)$	11.43	11.48	5.41	11.41	6.96	12.98
$(250, 0.5, 0.1)$	6.34	6.31	3.90	6.31	4.08	7.39
$(250, 0.5, 0.25)$	3.11	3.12	3.04	3.13	2.33	3.61
$(250, 1.0, 0.05)$	11.73	11.74	4.66	11.72	6.83	16.72
$(250, 1.0, 0.1)$	6.04	6.08	2.84	6.02	4.00	9.47
$(250, 1.0, 0.25)$	2.70	2.77	1.67	2.71	2.09	4.68
$(500, 0.02, 0.05)$	3.24	3.36	3.07	3.24	2.85	3.38
$(500, 0.02, 0.1)$	2.38	2.45	2.49	2.47	2.00	2.52
$(500, 0.02, 0.25)$	1.09	1.09	1.09	1.09	1.09	1.09
$(500, 0.5, 0.05)$	11.57	11.60	5.20	11.59	6.62	13.65
$(500, 0.5, 0.1)$	6.29	6.30	3.70	6.30	4.01	7.32
$(500, 0.5, 0.25)$	3.08	3.10	3.04	3.07	2.13	3.68
$(500, 1.0, 0.05)$	11.58	11.69	4.26	11.55	6.28	17.52
$(500, 1.0, 0.1)$	5.88	5.87	2.71	5.83	3.88	9.82
$(500, 1.0, 0.25)$	2.66	2.64	1.65	2.61	2.01	4.70
$(n_a = n_p, d, t)$			Uniform Random (UNI) Instances			
$(250, 0.02, 0.0)$	2.63	2.75	2.52	2.87	1.94	4.12
$(250, 0.02, 0.25)$	2.31	2.35	2.28	2.54	1.48	3.69
$(250, 0.02, 0.5)$	2.12	2.19	2.05	2.23	1.14	3.18
$(250, 0.5, 0.0)$	4.36	4.32	3.64	4.36	2.11	4.13
$(250, 0.5, 0.25)$	4.05	4.07	3.30	4.05	2.02	3.76
$(250, 0.5, 0.5)$	3.35	3.45	2.71	3.41	2.01	3.11
$(250, 1.0, 0.0)$	4.58	4.58	3.80	4.58	2.16	4.01
$(250, 1.0, 0.25)$	4.03	4.05	3.45	4.03	2.01	3.73
$(250, 1.0, 0.5)$	3.56	3.55	2.88	3.55	2.03	3.06
$(500, 0.02, 0.0)$	3.21	3.26	3.07	3.30	2.00	4.38
$(500, 0.02, 0.25)$	3.06	3.12	2.90	3.18	1.99	3.93
$(500, 0.02, 0.5)$	2.60	2.63	2.31	2.77	1.77	3.23
$(500, 0.5, 0.0)$	5.00	4.96	4.25	5.04	2.23	4.37
$(500, 0.5, 0.25)$	4.65	4.64	3.93	4.71	2.04	3.90
$(500, 0.5, 0.5)$	4.05	4.07	3.28	4.11	2.01	3.26
$(500, 1.0, 0.0)$	5.15	5.16	4.27	5.16	2.33	4.37
$(500, 1.0, 0.25)$	4.72	4.71	3.90	4.71	2.02	3.96
$(500, 1.0, 0.5)$	4.17	4.17	3.37	4.18	2.03	3.22

Table 4: Unpopularity Factors for Various Algorithms (n_a = number of applicants, n_p = number of posts, d = density, t = tie probability, λ = distribution parameter)

	RMM	RMM-BFS	MWM & MWM-BFS	EXP	BND	MINW
$(n_a = n_p, d)$			Real Estate Instances			
$(250, 10)$	7.98	8.03	7.70	8.00	7.63	8.69
$(250, 125)$	75.26	75.28	59.45	75.30	69.72	82.61
$(250, 250)$	162.53	162.72	136.51	162.63	150.60	164.13
$(500, 10)$	8.10	8.10	8.10	8.10	8.10	8.83
$(500, 250)$	124.41	124.58	90.85	124.64	111.98	142.95
$(500, 500)$	286.27	286.36	230.18	286.44	260.35	295.65
$(n_a = n_p, d)$			NBA Instances			
$(250, 10)$	4.08	4.09	3.79	4.09	4.07	4.52
$(250, 125)$	26.65	26.77	10.85	26.64	18.36	31.67
$(250, 250)$	63.78	63.90	16.36	63.85	49.18	64.10
$(500, 10)$	4.95	4.99	4.16	5.02	4.01	5.12
$(500, 250)$	43.22	43.40	12.44	43.32	26.71	50.56
$(500, 500)$	99.81	99.95	21.11	99.90	67.13	96.04

Table 5: Unpopularity Factors for Various Algorithms. (n_a = number of applicants, n_p = number of posts, d = degree of each applicant)

EXP and RMM exhibit similar unpopularity factors. This is not the case, however, with MWM which shows a different behavior. Rather surprisingly MWM and MWM-BFS compute matchings which have better unpopularity factors even when compared to the matchings computed by BND. More specifically, one of BND, MWM or MWM-BFS always computes the matching with the lowest unpopularity factor. In easier instances such as sparse instances or even dense UNI instances the BND computes better matchings, while MWM and MWM-BFS compute significantly better matchings in all other cases. These issues may be worthy of further investigation. On the negative side, rank-maximal matchings are not always a good solution. As observed in [11], there are instances of the problem which admit a popular matching but any rank-maximal matching of such an instance has an unpopularity factor of $\Theta(n)$.

Apart from the unpopularity factor we also compared the algorithms based on the total, the maximum rank as well as the cardinality of the computed matchings. The results can be found at Tables 6 and 7 using as a point of reference the MINW algorithm. Since rank maximal matchings have the same signature, there is no distinction between RMM, RMM-BFS, MWM, MWM-BFS or EXP. By definition MINW computes matchings with maximum cardinality and minimum total rank. The last four columns of Tables 6 and 7 present total rank as a percentage increase and cardinality as a percentage decrease over the corresponding MINW result. Again, as we compute averages over 100 instances, tables contain fractional numbers.

In dense UNI instances, MINW computes matchings with significantly lower total rank and much smaller maximum rank. In sparse UNI instances, RMM and BND become more competitive, in both maximum as well as in total rank. In some cases even a decrease of total rank can be observed, which is ofcourse accompanied by a cardinality decrease. Whether minimum total rank or maximum cardinality is more desirable is a subject of the actual application at hand. The situation is difference in more difficult instances like HC and VS. The increase in maximum rank or total rank is much smaller and moreover there is no cardinality change. Table 6 suggests that rank-maximal matchings exhibit better properties than matchings computed by BND. In NBA and real estate instances, independently of the edge density, all matchings exhibit very similar maximum ranks. A noteworthy difference is observed in the cardinality of the matchings in the case of sparse instances. This difference is accompanied by a similar change in the total rank. The

	Maximum Rank			Total Rank		Cardinality	
	RMM	BND	MINW	RMM	BND	RMM	BND
$(n_a = n_p, d, t)$			Highly Correlated Random (HC) Instances				
$(250, 0.02, 0)$	5	5	5	0%	0%	0%	0%
$(250, 0.02, 0.25)$	1.38	2.13	1.38	0%	+13.91%	0%	0%
$(250, 0.02, 0.5)$	1	1	1	0%	0%	0%	0%
$(250, 0.5, 0.0)$	125	125	125	0%	0%	0%	0%
$(250, 0.5, 0.25)$	91.25	98.57	86.67	+3.35%	+11.11%	0%	0%
$(250, 0.5, 0.5)$	59.32	69.01	54.69	+5.33%	+19.07%	0%	0%
$(250, 1.0, 0.0)$	250	250	250	0%	0%	0%	0%
$(250, 1.0, 0.25)$	203.4	195.78	182.14	+4.67%	+4.82%	0%	0%
$(250, 1.0, 0.5)$	143.42	136.21	120.28	+7.89%	+8.22%	0%	0%
$(500, 0.02, 0.0)$	10	10	10	0%	0%	0%	0%
$(500, 0.02, 0.25)$	3.93	6.55	3.89	+0.0529942%	+55.54%	0%	0%
$(500, 0.02, 0.5)$	1.54	3.59	1.54	0%	+23.67%	0%	0%
$(500, 0.5, 0)$	250	250	250	0%	0%	0%	0%
$(500, 0.5, 0.25)$	184.27	195.18	177.19	+2.67%	+8.07%	0%	0%
$(500, 0.5, 0.5)$	120.93	135.98	113.75	+4.32%	+13.88%	0%	0%
$(500, 1.0, 0.0)$	500	500	500	0%	0%	0%	0%
$(500, 1.0, 0.25)$	399.63	388.12	367.67	+03.48%	+3.54%	0%	0%
$(500, 1.0, 0.5)$	278.6	267.65	243.03	+5.99%	+6.09%	0%	0%
$(n_a = n_p, d, \lambda)$			Variable Exponential (VS) Instances				
$(250, 0.02, 0.05)$	2.52	3.7	2.52	0%	+33.25%	0%	0%
$(250, 0.02, 0.1)$	1.85	3.41	1.85	0%	+32.47%	0%	0%
$(250, 0.02, 0.25)$	1.01	1.02	1.01	0%	+0.2%	0%	0%
$(250, 0.5, 0.05)$	19.56	26.27	18.82	+3.02%	+16.41%	0%	0%
$(250, 0.5, 0.1)$	10	15.25	9.98	+1.95%	+19.38%	0%	0%
$(250, 0.5, 0.25)$	4.02	7.17	4.02	+0.28%	+19.52%	0%	0%
$(250, 1, 0.05)$	28.33	28.76	25.05	+4.16%	+4.27%	0%	0%
$(250, 1, 0.1)$	16.25	16.87	14.54	+2.77%	+3.25%	0%	0%
$(250, 1, 0.25)$	8.19	8.64	7.56	+1.17%	+1.99%	0%	0%
$(500, 0.02, 0.05)$	4.41	7.24	4.37	+0.08%	+42.36%	0%	0%
$(500, 0.02, 0.1)$	2.97	5.1	2.95	+0.057%	+57.02%	0%	0%
$(500, 0.02, 0.25)$	1.09	1.28	1.09	0%	+2.27%	0%	0%
$(500, 0.5, 0.05)$	20.03	28.13	19.82	+2.31%	+14.47%	0%	0%
$(500, 0.5, 0.1)$	10.1	16.63	10	+1.25%	+18.72%	0%	0%
$(500, 0.5, 0.25)$	4.07	7.58	4.04	+0.2%	+21.46%	0%	0%
$(500, 1.0, 0.05)$	29.24	29.95	26.32	+2.55%	+2.69%	0%	0%
$(500, 1.0, 0.1)$	16.82	17.41	15.39	+1.74%	+1.93%	0%	0%
$(500, 1.0, 0.25)$	8.55	9.02	8.05	+0.69%	+1.19%	0%	0%
$(n_a = n_p, d, t)$			Uniform Random (UNI) Instances				
$(250, 0.02, 0.0)$	5	5	4.99	-20.54%	-20.33%	-7.24%	-9.82%
$(250, 0.02, 0.25)$	4.72	4.89	4.76	-15.98%	-17.91%	-5.59%	-8.76%
$(250, 0.02, 0.5)$	4.01	4.16	3.91	-9.87%	-14.4%	-3.39%	-7.13%
$(250, 0.5, 0.0)$	78.97	42.56	7.31	+51.56%	+53.62%	-0.18%	-0.024%
$(250, 0.5, 0.25)$	55.51	37.48	5.62	+35.26%	+43.73%	-0.16%	-0.008%
$(250, 0.5, 0.5)$	33.83	29.77	4.04	+18.36%	+28.86%	-0.13%	-0.1%
$(250, 1.0, 0.0)$	129.07	55.33	7.47	+69.5%	+56.31%	0%	0%
$(250, 1.0, 0.25)$	84.2	39.86	5.67	+49.2%	+45.96%	0%	0%
$(250, 1.0, 0.5)$	51.53	40.75	3.98	+27.52%	+34.69%	0%	0%
$(500, 0.02, 0.0)$	9.85	9.93	8.05	-6.86%	+0.87%	-3.94%	-5.1%
$(500, 0.02, 0.25)$	8.65	8.83	6.05	-5.63%	+0.56%	-3.05%	-4.08%
$(500, 0.02, 0.5)$	6.71	7.06	4.42	-4.01%	+0.77%	-1.87%	-2.63%
$(500, 0.5, 0.0)$	146.77	63.01	8.29	+64.72%	+56.45%	-0.1%	-0.02%
$(500, 0.5, 0.25)$	116.47	49.75	6.32	+49.66%	+47.04%	-0.09%	-0.006%
$(500, 0.5, 0.5)$	74.43	52.08	4.49	+28.21%	+36.05%	-0.08%	0%
$(500, 1.0, 0.0)$	238.17	118.82	8.42	+84.06%	+64.58%	0%	0%
$(500, 1.0, 0.25)$	174.29	52.76	6.52	+62.31%	+48.21%	0%	0%
$(500, 1.0, 0.5)$	116.51	53.97	4.46	+37.1%	+36.93%	0%	0%

Table 6: Maximum, Total Rank and Cardinality for Various Algorithms.. (Table presenting total rank as a percentage increase and cardinality as a percentage decrease compared to the MINW. The results for RMM are valid for all rank-maximal matching algorithms.)

	Maximum Rank			Total Rank		Cardinality	
	RMM	BND	MINW	RMM	BND	RMM	BND
(n_a, n_p)			Real Estate Instances				
$(250, 10)$	9.85	9.85	9.95	-11.09%	-10.10%	-6.70%	-6.31%
$(250, 125)$	124.72	124.95	125	-9.61%	-9.49%	-5.9%	-5.87%
$(250, 250)$	250	250	250	+1.58%	+1.55%	0%	0%
$(500, 10)$	10	10	10	-13.16%	-13.08%	-7.94%	-7.86%
$(500, 250)$	249.44	250	249.99	-9.54%	-9.41%	-6.03%	-6.01%
$(500, 500)$	500	500	499.97	+1.57%	+1.58%	0%	0%
(n_a, n_p)			NBA Instances				
$(250, 10)$	9	9.24	9	-10.57%	+4.11%	-5.67%	-3.64%
$(250, 125)$	92.89	92.94	92.95	-18.68%	-14.71%	-10.05%	-8.85%
$(250, 250)$	196	196	195.96	+2.91%	+3.35%	0%	0%
$(500, 10)$	7	7	7	-3.03%	+7.83%	-1.57%	-2.1%
$(500, 250)$	155.88	155.89	157.71	-18.21%	-15.02%	-10.27%	-8.36%
$(500, 500)$	327	327	327	+3.43%	+3.23%	0%	0%

Table 7: Maximum, Total Rank and Cardinality for Various Algorithms.. (Table presenting total rank as a percentage increase and cardinality as a percentage decrease compared to the MINW. The results for RMM are valid for all rank-maximal matching algorithms.)

positive observation here is that, in dense instances, both rank-maximal algorithms and the BND algorithm compute matchings with the same cardinality as MINW and a very small total rank increase.

5 Conclusions

We compared several rank-maximal matching algorithms and two algorithms for popular matchings regarding their running time performance, the unpopularity factor of their produced matchings as well as other criteria such as cardinality, total rank and maximum rank. It should be clear that instances where applicants have strong correlation on their preferences are more difficult to solve. The results presented in the paper should help the interested reader decide, whether switching from a maximum cardinality minimum weight matching algorithm to a more preferences tailored algorithm makes sense.

We have also identified that the technique of reducing the problem to the maximum weight matching problem and solving it with the implicit reduction of [23], tends to perform better and to produce matchings with smaller unpopularity factors. Moreover, the degradation in total or maximum rank is only a small percentage.

References

[1] David J. Abraham, Robert W. Irving, Telikepalli Kavitha, and Kurt Mehlhorn. Popular matchings. *SIAM Journal on Computing*, 37(4):1030–1045, 2007.

[2] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, Englewood Cliffs, NJ, 1993.

[3] Nayantara Bhatnagar, Sam Greenberg, and Dana Randall. Sampling stable marriages: why spouse-swapping won't work. In *Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*, SODA '08, pages 1223–1232, Philadelphia, PA, USA, 2008. Society for Industrial and Applied Mathematics.

[4] Anna Bogomolnaia and Jean-Francois Laslier. Euclidean preferences. *Journal of Mathematical Economics*, 43(2):87–98, February 2007.

[5] M. J. Condorcet. Éssai sur l'application de l'analyse à la probabilité des décisions rendues à la pluralité des voix, 1785.

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

[7] Peter Gärdenfors. Match making: Assignments based on bilateral preferences. *Behavioral Science*, 20(3):166–173, 1975.

[8] Andrew V. Goldberg and Robert Kennedy. An efficient cost scaling algorithm for the assignment problem. *Math. Program.*, 71(2):153–177, 1995.

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

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

[11] Chien-Chung Huang, Telikepalli Kavitha, Dimitrios Michail, and Meghana Nasre. Bounded unpopularity matchings. *Algorithmica*, pages 1–20, 2010. 10.1007/s00453-010-9434-9.

[12] Aanund Hylland and Richard Zeckhauser. The efficient allocation of individuals to positions. *Journal of Political Economy*, 87(2):293–314, 1979.

[13] Robert W. Irving, Telikepalli Kavitha, Kurt Mehlhorn, Dimitrios Michail, and Katarzyna E. Paluch. Rank-maximal matchings. *ACM Transactions on Algorithms*, 2(4):602–610, 2006.

[14] M.-Y. Kao, T. W. Lam, W.-K. Sung, and H.-F. Ting. A decomposition theorem for maximum weight bipartite matchings. *SIAM Journal of Computing*, 31(1):18–26, 2001.

[15] Telikepalli Kavitha, Julin Mestre, and Meghana Nasre. Popular mixed matchings. *Theoretical Computer Science*, 412(24):2679 – 2690, 2011. Selected Papers from 36th International Colloquium on Automata, Languages and Programming (ICALP 2009).

[16] Johannes Langguth, Fredrik Manne, and Peter Sanders. Heuristic initialization for bipartite matching problems. *J. Exp. Algorithmics*, 15:1.3:1.1–1.3:1.22, March 2010.

[17] Library for Matchings with One Sided Preferences. `http://www.dit.hua.gr/~michail/subpages/libmosp/doxygen`, 2010.

[18] Mohammad Mahdian. Random popular matchings. In *Proceedings of the 7th ACM conference on Electronic commerce*, EC '06, pages 238–242, New York, NY, USA, 2006. ACM.

[19] David F. Manlove and Colin T. S. Sng. Popular matchings in the capacitated house allocation problem. In *Proceedings of the 14th Annual European Symposium on Algorithms (ESA' 06)*, volume 4168 of *Lecture Notes in Computer Science*, pages 492–503, London, UK, 2006. Springer-Verlag.

[20] Richard Matthew McCutchen. The least-unpopularity-factor and least-unpopularity-margin criteria for matching problems with one-sided preferences. In *Proceedings of the 8th Latin American conference on Theoretical informatics*, LATIN'08, pages 593–604, Berlin, Heidelberg, 2008. Springer-Verlag.

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

[22] Julián Mestre. Weighted popular matchings. In *Proceedings of Automata, Languages and Programming, 33rd International Colloquium*, volume 4051 of *Lecture Notes in Computer Science*, pages 715–726, London, UK, 2006. Springer-Verlag.

[23] Dimitrios Michail. Reducing rank-maximal to maximum weight matching. *Theoretical Computer Science*, 389(1-2):125 – 132, 2007.

[24] Statistics. `www.databasebasketball.com/stats_download.htm`.

[25] Alvin E. Roth and Andrew Postlewaite. Weak versus strong domination in a market with indivisible goods. *Journal of Mathematical Economics*, 4(2):131–137, 1977.

[26] João C. Setubal. Sequential and parallel experimental results with bipartite matching algorithms. Technical Report IC-96-09, Institute of Computing, University of Campinas, September 1996.

[27] Leong Hou U, Nikos Mamoulis, and Kyriakos Mouratidis. A fair assignment algorithm for multiple preference queries. *Proc. VLDB Endow.*, 2:1054–1065, August 2009.

[28] Lin Zhou. On a conjecture by Gale about one-sided matching problems. *Journal of Economic Theory*, 52(1):123–135, 1990.

[29] `www.zillow.com`.