Improving solution times for stable matching problems through preprocessing

Improving solution times for stable matching problems through preprocessing

William Pettersson[a,*], Maxence Delorme[b], Sergio García[b], Jacek Gondzio[b], Joerg Kalcsics[b], David Manlove[a]

[a] *School of Computing Science, University of Glasgow, United Kingdom*
[b] *School of Mathematics, University of Edinburgh, United Kingdom*

Abstract

We present new theory, heuristics, and algorithms for preprocessing instances of the Stable Marriage problem with Ties and Incomplete lists (SMTI) and the Hospitals/Residents problem with Ties (HRT). Instances of these problems can be preprocessed by removing from the preference lists of some agents entries such that the set of stable matchings is not affected. Removing such entries reduces the problem size, creating smaller models that can be more easily solved by integer programming (IP) solvers. The new theorems are the first to describe when preference list entries can be removed from instances of HRT when ties are present on both sides, and also extend existing results on preprocessing instances of SMTI. A number of heuristics, as well as an IP model and a graph-based algorithm, are presented to find and perform this preprocessing. Experimental results show that our new graph-based algorithm achieves a 44% reduction in the average running time to find a maximum weight stable matching in real-world instances of SMTI compared to existing preprocessing techniques, and 80% compared to not using preprocessing. We also show that, when solving MAX-HRT instances with ties on both sides, our new techniques can reduce runtimes by up to 55%.

Keywords: Preprocessing, Stable Marriage problem, Hospital/Residents problem, Ties and Incomplete Lists

1. Introduction

Stable matching problems consist of some set (or sets) of agents where each agent ranks a subset of the other agents in order of preference. The solution to such a problem is a stable pairing of agents, or a *stable matching*. Gale and Shapley introduced the notion of stability, as well as the Stable Marriage (SM) problem, in their seminal paper [9], along with a polynomial-time algorithm to solve SM. An instance of SM consists of $2n$ agents split into equal-sized sets called men and women, where each woman ranks all men (and none of the women) in strict order of preference, and vice-versa. Each man and woman must be paired up (with a woman and man respectively) into a *stable* matching. A matching is *stable* if there are no two people not currently matched together who would prefer to be matched with each other over their current partner. Two such people are said to *block* the matching — they are a *blocking pair*.

In the same paper [9], Gale and Shapley also introduced the College Admissions problem that models the problem of allocating students to positions at colleges. This is a stable matching problem similar to SM, except that while each student can be assigned to at most one college, each college has some positive integral capacity q such that they can be matched to at most q students. Additionally, students may find some colleges completely unacceptable, so instead students only rank their *acceptable* colleges in strict order.

*Corresponding author
Email address: william@ewpettersson.se (William Pettersson)

The Hospital-Residents problem (HR) [27, 29] is a different formalisation of the same College Admissions model in centralised matching schemes for allocating resident doctors to hospitals. In HR colleges are replaced with hospitals and students are replaced with resident doctors, with many applications including the National Resident Matching Program (NRMP) in the US [27], the Canadian Resident Matching Service (CaRMS) in Canada [28], and the Scottish Foundation Allocation Scheme (SFAS) in Scotland [13]. Further applications include School Choice in Boston [2], Hungary [5, 6], Spain [6, 26], and Turkey [4, 6].

Just as in the case of HR, one-to-one variants of SM arise when agents are allowed to have unacceptable partners. These agents will have *incomplete* preference lists, and the corresponding problem is called Stable Marriage with Incomplete lists or SMI. The Gale-Shapley algorithm [9] can be modified slightly to find stable matchings in polynomial time [11], and all stable matchings in a given instance have the same size [10].

A further complication to stable matching problems is encountered when agents are either unable or unwilling to distinguish between two acceptable potential partners, common in real-world applications. This creates *ties* in the preference lists, and the resulting extension of SM is called Stable Marriage with Ties (SMT). The presence of such ties in preference lists creates three possible definitions for stability, called *weak stability*, *strong stability*, and *super-stability* [12]. In this paper, we only discuss weak stability, where a pair is blocking only if both agents strictly prefer each other to their currently assigned partner, as only under weak stability are stable matchings guaranteed to exist [12]. Trivially, weakly stable matchings of instances in SMT can be found by breaking ties arbitrarily and then running the deferred acceptance algorithm of Gale and Shapley [9], and all such stable matchings must, therefore, have the same size.

If a stable matching problem allows both ties and incomplete lists, the resulting problem is called the Stable Marriage with Ties and Incomplete lists problem (SMTI). In instances of SMTI, weakly stable matchings might have different sizes, the problem of finding a maximum cardinality stable matching (called MAX-SMTI) is NP-hard [23], and the associated problem of determining whether a given pair is in any stable matching is NP-complete [23].

A special case of SMTI is SMTI with Globally Ranked Pairs (SMTI-GRP). An instance of SMTI is an instance of SMTI-GRP if each pair of acceptable agents $\{u, v\}$ can be assigned a numeric score $f(u, v)$ such that u prefers v over v' if and only if $f(u, v) > f(u, v')$, and u is indifferent between v and v' if and only if $f(u, v) = f(u, v')$. Whilst showing the existence of such a function can be used to characterise an instance of SMTI as an instance of SMTI-GRP, in many applications said function is instead used to create the SMTI-GRP instance [8]. Despite the restricted nature of this problem, stable matchings in instances of SMTI-GRP can still have varying sizes, and finding a maximum sized matching is NP-hard [3]. In some applications, a maximum weight stable matching is desired, where the weight of a matching is the sum of the weights $f(u, v)$ of each matched pair (u, v). The problem of finding such a matching is called MAX-WT-SMTI-GRP, and has been applied to the pairing of children with adoptive families by the British charity Coram [8]. This problem has also been shown to be NP-hard [7].

The Hospitals/Residents problem with Ties (HRT) is the extension of HR that allows agents to express indifference in their preference lists [16, 17]. It can also be equivalently defined as the extension of SMTI where one set of agents (the hospitals) is allowed to have positive integral capacities. As an extension of SMTI, it also is NP-hard to find a maximum cardinality stable matching in HRT, and this problem is called MAX-HRT. While HRT only allows capacities on one side, the Workers/Firms problem (WF) (also known as the many-to-many stable matching problem, or stable b-matching problem) generalises SMI to allow capacities on either side, and the Workers/Firms with Ties problem (WFT) generalises WF to allow agents to express indifference [22].

MAX-SMTI and MAX-HRT have been solved with constraint programming [24] and integer programming (IP) techniques [8, 19, 20]. Linear programming models for SM were originally studied for their properties [11, 30], and the same models with integrality constraints have since been used (and in the case of HRT, adapted) to solve MAX-SMTI and MAX-HRT [19, 20]. Recently the use of dummy variables, constraint merging and secondary stability constraints has been shown to significantly reduce the time to solve MAX-SMTI, MAX-WT-SMTI-GRP, and MAX-HRT [8].

A more complete introduction to stable matching problems can be found in [22].

1.1. Preprocessing stable matching problems

While Gale and Shapley's original algorithm [9] does find a stable matching in polynomial time, an extended version [11] removes from the problem preference list entries that will not affect the outcome of the algorithm. Even though stable matching problems like MAX-SMTI and MAX-HRT are often solved with IP models, and not the algorithm of Gale and Shapley, the idea of removing preference list entries to simplify such problems can still be advantageous. Removing preference list entries without modifying the set of stable matchings shortens preference lists, in turn reducing the size of the models and thus solution times [8]. This removal of preferences is done before building the IP model, and so it is called preprocessing.

Preprocessing can be performed using variations of the Gale-Shapley algorithm, wherein agents "propose" in rounds, and based upon the outcomes of such proposals certain pairs may be removed. Such a technique is known for instances of SMTI where preferences on one side have length at most two [15]. A similar technique has also been applied to HRT [14] for no restriction on the lengths of the preference lists, namely "Hospitals-offer" and "Residents-apply". However, these require that the preference lists of the residents be strictly ordered (i.e., contain no ties), while the hospitals may have ties in their preference lists. Preprocessing theory for instances with ties on both sides has recently been introduced for SMTI [8], which we expand upon, and also extend to HRT. In this theory, pairs that can be removed are identified by considering a given agent in a stable matching problem, a set of positions they wish to be matched to, and the potential competition the agent may face for these positions. If the number of positions is greater than the number of competitors for these positions, then in any stable matching the given agent cannot be assigned to a position worse than the given set, and so any worse preferences from the agent can be removed. The same paper also defines a simple preprocessing heuristic (which we describe in Section 3.1.1) to detect such sets, and uses this heuristic to gain significant improvements in overall solution times on various families of MAX-SMTI instances, both randomly-generated and application-specific.

1.2. Our contribution

In this paper, we extend the theory behind preprocessing to identify more preference list entries that can be removed from instances of SMTI without affecting any stable matching. This theory is also extended to HRT, where existing techniques only worked when one set of agents (the doctors) had strict preferences, while our new techniques apply when both doctors and hospitals may express ties. A number of new heuristics are given that find a subset of the preference list entries that can be removed, as well as a polynomial-time algorithm that finds all the entries in preference lists that can be removed by preprocessing according to our extended theory. Experimental results show that the average time to solve real-world MAX-WT-SMTI-GRP instances from Coram is reduced from 149 seconds using existing preprocessing techniques to only 83 seconds using our new graph-based algorithms. The number of preference list entries removed according to our new theory is increased by approximately 82%, compared to existing preprocessing techniques, contributing to this reduced runtime. We demonstrate similar results in randomly-generated instances of MAX-WT-SMTI-GRP. This increase in the number of preference list entries removed is also shown in the randomly-generated MAX-WT-SMTI-GRP instances with a similar size, where the average number of entries removed increases from 83093 to 185437.

For MAX-SMTI problems, we investigate the effect that the length of preference lists can have on the usefulness of preprocessing. When all candidates have preference lists with 3 entries, relatively few entries can be removed via preprocessing, and we show that using more complex preprocessing techniques can, at times, be detrimental. If the candidates have preference lists with 5 entries, we show that preprocessing is useful, with a reduction from 478 seconds without preprocessing to 437 with heuristic preprocessing. When the preference list of each candidate has 10 entries, however, we show that it is more useful to introduce dummy variables [8], and once such variables are introduced, preprocessing is again less effective. When these dummy variables are present, preprocessing becomes useful when candidates have preference lists that contain a sizeable proportion of the positions. We show that with 250 agents, preprocessing begins to have an effect when preference lists of the candidates have 100 entries, and becomes more significant as longer preference lists are used. When candidate preference lists have 200 entries, we see that total runtime (preprocessing and solving) is reduced from 138 seconds using no preprocessing, or 128 seconds using existing preprocessing, to 102 seconds using our new heuristics.

We also experimentally study instances of HRT. In instances where only the hospitals may have ties in their preference lists, we report that preprocessing is rarely relevant, and that existing techniques "Hospitals-offer" and "Residents-apply" [14] are still suitable. However, when ties are present in both sides, we are able to show a 24% to 55% improvement in runtime on instances with up to 4000 doctors, as well as solving 118 of 120 instances with 8000 doctors compared to 112 with existing techniques.

1.3. Paper layout

Section 2 introduces some of the background, as well as both existing theory and our new theory to detect more preprocessing opportunities, and the extension of this theory to HRT. An exact algorithm, an IP model, and several heuristics for finding preprocessing opportunities are given in Section 3. Section 4 describes the results of the experiments carried out to compare these approaches, and our conclusion follows in Section 5.

2. Background and preprocessing theory

Section 2.1 gives notation and definitions for the stable matching problems we study. Section 2.2 then describes some of the known preprocessing techniques in the literature. Section 2.3 theoretically identifies more preprocessing that can be performed, and Section 2.4 then extends these results to stable matching problems with capacities.

2.1. Definitions and notation

We begin by defining the *Workers / Firms problem with Ties (WFT)*, as the *Hospital / Residents problem with Ties (HRT)* and the *Stable Marriage problem with Ties and Incomplete Lists (SMTI)* are specialisations of WFT.

An instance I of WFT consists of two sets of agents, which we call *positions* (P) and *candidates* (C). We denote their sizes as $n_p = |P|$ and $n_c = |C|$. We use the terms positions and candidates specifically because our preprocessing is easier to explain when we mentally separate the two sets of agents into distinct types. However, they are still symmetrically equivalent, so any preprocessing technique applied to candidates can also be applied to positions and vice-versa.

Each position (respectively candidate) finds some or all of the candidates (respectively positions) acceptable, and ranks these candidates (respectively positions) in non-increasing order of preference. These preferences may include ties, where an agent is indifferent between two or more options. In this paper we assume that if a position p finds a candidate c acceptable, then candidate c also finds p acceptable, and we will say that the pair (c, p) is acceptable. If a candidate c strictly prefers p_1 over p_2, we will write $p_1 \prec_c p_2$. If c either prefers p_1 over p_2 or is indifferent between the two, we write $p_1 \preceq_c p_2$. Each individual candidate c and position p also has some positive integral capacity (or quota), denoted q_c and q_p, respectively.

A matching M in an instance of WFT is a subset of $C \times P$ such that each candidate c (respectively position p) occurs in at most q_c (respectively q_p) pairs, and every $(c, p) \in M$ is acceptable. We will write $M(p)$ for the set of candidates matched with p in M (i.e., $M(p) = \{c \mid (c, p) \in M\}$), and we will write $M(c)$ for the set of positions matched with c in M (i.e., $M(c) = \{p \mid (c, p) \in M\}$). In a given matching, if a candidate c (respectively position p) occurs in exactly q_c (respectively q_p) pairs, we say they are *full*. Otherwise, we say they are *undersubscribed*. Additionally, if a candidate or position occurs in exactly 0 pairs, we say they are *empty*. A candidate or position who is empty is simultaneously undersubscribed.

Definition 1. *Given an instance I of WFT and a matching M, a pair $(c, p) \notin M$ (i.e., p and c are not matched together) is a blocking pair of I if*

- *(c, p) is acceptable, and*
- *either p is undersubscribed, or there is some $c' \in M(p)$ such that $c \prec_p c'$ (i.e., p strictly prefers c to one of their currently assigned candidates), and*

- *either c is undersubscribed, or there is some $p' \in M(c)$ such that $p \prec_c p'$ (i.e., c strictly prefers p to one of their currently assigned positions).*

A matching with no blocking pairs is called stable*.*

HRT is the restriction of WFT where one set of agents (either all candidates, or all positions) have unitary capacity, and SMTI is the restriction of WFT where all candidates and all positions have unitary capacity.

Example 1 gives a sample of the notation used to describe the preferences of agents in individual instances of SMTI, HRT, and WFT. Each agent is identified, and then followed by its preferences in descending order such that if p_1 comes before p_3, then p_1 is preferred over p_3. A tie in candidate c's preference list is a group of positions that candidate c does not distinguish between. We will say that c *ranks* p_1 at the same level or better than p_2 if both p_1 and p_2 appear in the preference list of c, and if either p_1 and p_2 appear in the same tie group, or c prefers p_1 to p_2. As seen in Example 1, we write $[p_2\ p_3]$ in the preference list of candidate c_1 if candidate c_1 is indifferent between p_2 and p_3. Such a tie has length 2. Note that in a given preference list, if a position is not tied with any other position, such as the position p_1 in the preference list of c_1, we can refer to this as a tie of length 1. We define the *rank* of either an element of a preference list, or a tie within a preference list, as one plus the number of ties that are strictly preferred to it. Adding one allows the natural expression "first tie " to refer to the tie with rank 1.

Example 1. *The following is an example of preference lists as expressed by two candidates and three positions.*

$$
\begin{aligned}
&c_1 : p_1\ [p_2\ p_3]\\
&c_2 : p_2\ p_3\\
&p_1 : c_1\\
&p_2 : c_1\ c_2\\
&p_3 : c_1\ c_2
\end{aligned}
$$

2.2. Existing preprocessing theory for SMTI

An instance I of a stable matching problem is preprocessed by removing entries from preference lists such that p is removed from the preference list of c only if the pair (c, p) appears in no stable matching of I, and removing (c, p) does not create any new stable matchings. The removal of these preference list entries can, as demonstrated in Section 4, have a significant effect on the time taken to find either maximum cardinality stable matchings, or maximum weight stable matchings. Note that this preprocessing is not dependent on the specific stable matching problem being solved. Our preprocessing applies to any problem that searches for one or more stable matchings satisfying some criteria, as it does not remove any existing stable matching nor introduce any new stable matching. This includes the common problem of finding a largest stable matching, but also searches for egalitarian stable matchings, or even the problem of counting the number of stable matchings [22].

Under certain variations of stable matching problems, the existence of a stable matching is not guaranteed (see e.g., [16, 17, 18]). In an instance of a stable matching problem with no stable matchings, each acceptable pair vacuously is in no stable matching, and so removing some preference list entries corresponding to pairs that are not in any stable matching may inadvertently create a new instance that does contain a stable matching. However, even if an instance of a stable matching problem does admit some stable matchings, not all pairs that appear in no stable matching can be removed without creating new stable matchings, as seen in Example 2.

Example 2. *Consider the following instance of SMTI.*

$$\begin{array}{lllll} p_1: & c_2 & c_1 & [c_3 & c_4] \\ p_2: & [c_1 & c_3] & & \\ p_3: & c_1 & c_4 & & \\ p_4: & c_3 & c_4 & c_2 & \end{array}$$

$$\begin{array}{lllll} c_1: & [p_1 & p_2] & p_3 & \\ c_2: & p_4 & p_1 & & \\ c_3: & p_1 & [p_2 & p_4] & \\ c_4: & p_1 & p_4 & p_3 & \end{array}$$

It is routine to check that there is no stable matching that contains (c_1, p_1)*. To see this, if* (c_1, p_1) *is in a stable matching, then the stable matching must also contain* (c_2, p_4) *as otherwise* (c_2, p_1) *forms a blocking pair. Then, to ensure that* (c_4, p_4) *is not blocking, the stable matching must contain* (c_4, p_1)*, a contradiction as the stable matching already contains* (c_1, p_1)*. At the same time, however, the only pair that blocks the matching* $\{(c_1, p_3), (c_2, p_4), (c_3, p_2), (c_4, p_1)\}$ *is* (c_1, p_1)*, so removing* (c_1, p_1) *would create a new stable matching.*

The following theorem is a restatement of Theorem 1 in [8][1].

Theorem 1 (Theorem 1 in [8])**.** *Let I be an instance of SMTI. Consider a candidate c and a non-empty set of positions $\mathcal{P}$ such that for every position $p \in \mathcal{P}$, (c, p) is an acceptable pair. Let $\mathcal{C}$ be the set of candidates that at least one position in $\mathcal{P}$ ranks at the same level or better than c, i.e., $\mathcal{C} = \{c' \mid \exists\, p \in \mathcal{P}$ s.t. $c' \preceq_p c\}$. If $|\mathcal{P}| \geq |\mathcal{C}|$, then in any stable matching M, candidate c will be allocated a position p' such that $p' \preceq_c p$ for at least one $p \in \mathcal{P}$.*

The gist of this theorem is that for candidate c, $\mathcal{P}$ is some set of positions that c finds acceptable and $\mathcal{C}$ contains c's competition for these (i.e., any candidate who could match with a position $p \in \mathcal{P}$ such that (c, p) would not be a blocking pair). If $\mathcal{C}$ is small enough compared to $\mathcal{P}$, then candidates in $\mathcal{C}$ (excluding c) cannot possibly take all positions in $\mathcal{P}$. As a result, in any stable matching c must be matched with some position that is no worse than what c considers to be the worst position in P.

2.3. Identifying additional preprocessing in SMTI

We can extend Theorem 1 as follows:

Theorem 2. *Let I be an instance of SMTI. Let us consider a candidate c, a non-empty set of positions $\mathcal{P}$ such that for every position $p \in \mathcal{P}$ the pair (c, p) is an acceptable pair and a set of positions $\mathcal{P}'$ such that for any $p' \in \mathcal{P}'$*

1. *the pair (c, p') is not an acceptable pair, and*
2. *in any stable matching of I, p' will be assigned to some candidate (i.e., p' will not be unassigned).*

Let $\mathcal{C}$ be the set of all candidates c' that either:

Criterion 1: at least one position in $\mathcal{P}$ ranks c' at the same level or better than c, i.e.,

$$\exists\, p \in \mathcal{P} \text{ s.t. } c' \preceq_p c,$$

or

Criterion 2: at least one position in $\mathcal{P}'$ finds c' acceptable

[1]The original theorem mistakenly does not specify that $\mathcal{P}$ ($\mathcal{F}$ in [8]) be non-empty; we fix this here.

holds. If $|\mathcal{P}| + |\mathcal{P}'| \geq |\mathcal{C}|$, then in any stable matching M, candidate c will be allocated a position $p^\star$ such that $p^\star \preceq_c p$ for at least one $p \in \mathcal{P}$.

This theorem is a specialisation of Theorem 3, and so we skip the proof. Instead, we refer the reader to the upcoming proof of the latter. Note that Criterion 1 in Theorem 2 does mean that $c \in \mathcal{C}$.

In this theorem the set $\mathcal{P}'$ can contain positions that definitely will be filled, but not by c. These positions can, however, "use up" candidates that are in $\mathcal{C}$, meaning those candidates can no longer take positions in $\mathcal{P}$. This might result in there not being enough other candidates to fill positions in $\mathcal{P}$. Therefore, in a stable matching c cannot be matched with any position it considers worse than all positions in $\mathcal{P}$.

We have not yet indicated how to select $\mathcal{P}$ or $\mathcal{P}'$. Indeed, we demonstrate several methods for determining both $\mathcal{P}$ and $\mathcal{P}'$ in Section 3. Given appropriate sets $\mathcal{P}$ and $\mathcal{P}'$, Example 3 demonstrates how $\mathcal{C}$ is determined and used to decide whether preprocessing is possible.

Example 3 (Example of use of Theorem 2)**.** *Consider an instance of SMTI with $n_c = 3$ and $n_p = 4$ (i.e., there are 3 candidates and 4 positions). We will preprocess the preference list for c_3. The relevant preferences are as follows:*

$$
\begin{array}{lllll}
p_1: & c_1 & c_2 & & \\
p_2: & c_1 & c_2 & c_3 & \\
p_3: & c_2 & c_3 & & \\
p_4: & c_2 & c_3 & &
\end{array}
$$

$$
\begin{array}{lllll}
c_1: & p_1 & p_2 & & \\
c_2: & p_1 & p_2 & p_3 & p_4 \\
c_3: & p_2 & p_3 & p_4 &
\end{array}
$$

We begin by noting that the first choice for p_1 is c_1, and vice-versa. So, in any stable matching, p_1 will always be assigned to some candidate (we will use this fact in order to apply Theorem 2). Indeed, we could continue in such a manner to find the unique stable matching in this instance, but we do not proceed in this manner as our aim is to demonstrate the usage of Theorem 2.

Firstly, consider $\mathcal{P} = \{p_2, p_3\}$. Looking at the preferences of p_2 and p_3, we see that $\mathcal{C}$ must contain c_1, c_2, and c_3, as these are the candidates that at least one of p_2 or p_3 consider to be at least as good as c_3. This means that $|\mathcal{C}| > |\mathcal{P}|$, and so we cannot preprocess.

However, we also know that c_3 does not find p_1 acceptable, and that p_1 will always be matched to some candidate. So we can also consider $\mathcal{P}' = \{p_1\}$. Since the preference list of p_1 contains the candidates c_1 and c_2, we must add both of these to $\mathcal{C}$, but they are already present so there is no change to $\mathcal{C}$. Now we have $|\mathcal{C}| = |\mathcal{P}| + |\mathcal{P}'|$, and so we can remove from the preference list of c_3 any preferences worse than the worst position in $\mathcal{P}$. That is, we remove the position p_4 from the preference list of c_3 (and remove c_3 from the preference list of p_4).

We can reason through this process as well as follows. If c_3 is not matched to p_2, then p_2 must be matched to either c_1 or c_2. If, in addition to this, c_3 is also not matched to p_3, then p_3 must be matched to c_2. This means that p_2 must be matched to c_1. However, since p_1 must be matched to some candidate, this is a contradiction, as p_1 only finds c_1 or c_2 acceptable. Therefore, c_3 must be matched to one of p_2 or p_3.

2.4. Extending preprocessing to HRT

We extend the above preprocessing results to HRT now. Recall that in HRT, doctors have unitary capacity while hospitals have positive integral capacities. To preprocess a hospital's preference list, we must consider the hospitals as candidates, and must therefore allow all candidates to have positive integral capacities. However, to preprocess a doctor's preference list we must consider the doctors as candidates, and must therefore allow all positions to have positive integral capacities. Thus we must allow positions to refer to either hospitals or doctors, and similarly for candidates. Our general framework will, therefore, allow for positive integral capacities in all positions and candidates, but with the caveat that either all positions have unitary capacity, or all candidates have unitary capacity. This allows us to prove the correctness of

preprocessing HRT in one theorem. We later give an example demonstrating that this preprocessing does not immediately extended to instances of WFT.

Our theorem for HRT replaces the sizes of the sets $\mathcal{C}$, $\mathcal{P}$, and $\mathcal{P}'$ with the sums of the capacities of each in the criteria for preprocessing. Let $\mathcal{A}$ be some set of agents, and let q_a be the capacity (quota) of agent a. We will then write $|\mathcal{A}|_q$ to mean $\sum_{a\in\mathcal{A}} q_a$.

Theorem 3. *Let I be an instance of HRT (i.e., either the candidates have unitary capacity and positions have positive integral capacities, or vice-versa). Let us consider a candidate c, a non-empty set of positions $\mathcal{P}$ such that for every position $p \in \mathcal{P}$, (c, p) is an acceptable pair, a set of positions $\mathcal{P}'$ such that for any $p' \in \mathcal{P}'$, (c, p') is not an acceptable pair and in any stable matching of I p' will not be undersubscribed. Let $\mathcal{C}$ be the set of all candidates c' that either:*

Criterion 1: at least one position in $\mathcal{P}$ ranks at the same level or better than c, i.e.,

$$\exists\, p \in \mathcal{P} \text{ s.t. } c' \preceq_p c,$$

or

Criterion 2: at least one position in $\mathcal{P}'$ finds acceptable.

If $|\mathcal{P}|_q + |\mathcal{P}'|_q \geq |\mathcal{C}|_q$, then in any stable matching M, candidate c will be full and will only be allocated positions $p^\star$ such that $p^\star \preceq_c p$ for at least one $p \in \mathcal{P}$.

Proof. Let I, M, c, $\mathcal{P}$, $\mathcal{P}'$, and $\mathcal{C}$ be as given in the theorem, and assume towards a contradiction that either c is undersubscribed or c is assigned at least one position p^+ such that $p \prec_c p^+$ for all $p \in \mathcal{P}$.

By the definition, we know that $c \in \mathcal{C}$. Since c is either undersubscribed, or not assigned to only positions $p \in \mathcal{P}$, and since $|\mathcal{P}|_q + |\mathcal{P}'|_q > |\mathcal{C}|_q - 1$, and as every $p' \in \mathcal{P}'$ will be assigned to capacity with only candidates $c' \in \mathcal{C}$, by the pigeon hole principle there must be some $p \in \mathcal{P}$ that is either undersubscribed or full and assigned some candidate $c^+ \notin \mathcal{C}$.

We will now show that (c, p) forms a blocking pair. If p is undersubscribed then p would prefer to be assigned to c, and if p is full, then by construction p prefers c to c^+ (otherwise c^+ would be in $\mathcal{C}$). Similarly, if c is undersubscribed then c would prefer to be assigned p, and if c is full then c is assigned p^+, and, as $p' \prec_c p^+$ for all $p' \in \mathcal{P}$, then c prefers p to p^+.

Lastly, we need to show that $(c, p) \notin M$. To do this, we rely on the fact that either c or p must have unitary capacity. We know that p is either undersubscribed, or assigned some candidate $c^+ \neq c$, so if p has unitary capacity, then $(c, p) \notin M$. Similarly, we know that c is either undersubscribed, or assigned some $p^+ \neq p$, and so if c has unitary capacity, then $(c, p) \notin M$.

Thus, as either p or c must have unitary capacity, $(c, p) \notin M$, and so (c, p) will form a blocking pair. □

As in Theorem 2, Criterion 1 in Theorem 3 means that $c \in \mathcal{C}$.

Note that if we allow both candidates and positions in Theorem 3 to have positive integral capacities, we may reach a scenario in which both c and p are undersubscribed or prefer each other to one of their currently assigned partners, even though (c, p) is already in the matching, and so would not form a blocking pair. Example 4 demonstrates such a scenario.

Example 4. *Consider the following instance I of WFT which contains one candidate and two positions, wherein c_1 has a capacity of three, p_1 also has a capacity of three, and p_2 has a capacity of one.*

$$\begin{array}{lcll} c_1 & : & p_1 & p_2 \\ p_1 & : & c_1 & \\ p_2 & : & c_1 & \end{array}$$

Clearly the only stable matching is $\{(c_1, p_1), (c_1, p_2)\}$, but if we preprocess the preference list of p_1 with $\mathcal{P} = \{p_1\}$, then $\mathcal{C} = \{c_1\}$, and we determine that $|\mathcal{P}|_q = |\mathcal{C}|_q$. Thus, if the restriction in Theorem 3 that ensures either all candidates or all positions have unitary capacity were to be dropped, the pair (c_1, p_2) would incorrectly be removed from I. We note that if a pair can appear multiple times in a "matching", then this preprocessing theory would apply in an obvious manner.

Given an instance I of HRT, a candidate c and sets $\mathcal{P}$ and $\mathcal{P}'$ that satisfy Theorem 2, our preprocessing creates a new instance I' by removing from the preference list of c any positions that c considers strictly worse than all of those in $\mathcal{P}$. It is trivial to see that any stable matching that exists in I must also exist and be stable in I', but the opposite is not as clear. We will show such a result for WFT, which is more general than either HRT or SMTI. To prove our result, we first need the following lemma.

Lemma 4. *Let I be an instance of WFT, let c be some candidate such that in any stable matching in I, c is always full, let $\mathcal{P}$ be some set of positions such that if c is assigned to p in some stable matching of I then p is in $\mathcal{P}$, and let I' be the instance of WFT created from I by marking as unacceptable to c any position p^+ that satisfies $p \prec_c p^+$ for all $p \in \mathcal{P}$. Then in any matching M that is stable in I', c is full.*

With this lemma, which we prove in Appendix A, we can now show that our preprocessing is correct.

Theorem 5. *Let I be an instance of WFT, let c be some candidate such that in any stable matching in I, c is always full, let $\mathcal{P}$ be some set of positions such that if c is assigned to p in some stable matching of I then p is in $\mathcal{P}$, and let I' be the instance of WFT created from I by marking as unacceptable to c any position p^+ that satisfies $p \prec_c p^+$ for all $p \in \mathcal{P}$. Then any matching M that is stable in I' is also stable in I.*

Proof. Assume towards a contradiction that there is a matching M that is stable in I' but not stable in I. By Lemma 4 we know that c must be full in M. As M is not stable in I, there must be some pair (c^+, p^+) that blocks M in I but not in I'. Such a pair must be a pair that was marked as unacceptable when constructing I', and therefore $c^+ = c$.

As c is full in M, let $p^\star \in M(c)$ be a position that satisfies $p^+ \prec_c p^\star$. Such a $p^\star$ must exist for (c, p^+) to block M in I. However, as $p^\star \in M(c)$, $p^\star$ was not marked as unacceptable to c when constructing I' and so there must be some $p' \in \mathcal{P}$ such that $p^\star \preceq_c p'$. As $p' \in \mathcal{P}$, we also have $p' \prec_c p^+$, which gives us $p^\star \preceq_c p' \prec_c p^+ \prec_c p^\star$, a contradiction. □

Example 5 demonstrates that Theorem 3 is more powerful than "Hospitals-offer" and "Residents-apply" [14], the existing preprocessing algorithms for HRT.

Example 5. *This example demonstrates how Theorem 3 can detect preprocessing that is possible in an instance of HRT that "Hospitals-offer" and "Residents-apply" [14] are unable to detect. We use p to denote the hospitals and c to denote the resident for consistency with the theory. In this example, p_1 has capacity 1, p_2 has capacity 1, and p_3 has capacity 2.*

$$
\begin{array}{lllll}
c_1: & p_3 & p_2 & p_1 \\
c_2: & p_3 & p_2 & \\
c_3: & p_2 & p_3 & p_1
\end{array}
$$

$$
\begin{array}{lllll}
p_1: & c_3 & c_1 & \\
p_2: & c_2 & [c_1 & c_3] \\
p_3: & [c_1 & c_2 & c_3]
\end{array}
$$

It is routine to check that neither of "Hospitals-offer" and "Residents-apply" [14] can remove any preference list entries. However we can preprocess the preference list of c_1 by letting $\mathcal{P} = \{p_2, p_3\}$. Then $\mathcal{C} = \{c_1, c_2, c_3\}$, and as $q_{p_2} = 1$ and $q_{p_3} = 2$, $|\mathcal{P}|_q = 3 \geq 3 = |\mathcal{C}|_q$. This results in the removal of the entry p_1 from the preference list of c_1. This same result is trivially reasoned by realising that:

- *p_2 and p_3 both find all three doctors acceptable,*
- *together p_2 and p_3 have the capacity to accept all three doctors,*
- *all three doctors find both p_2 or p_3 acceptable, and*
- *no doctor prefers p_1 over either of p_2 or p_3.*

3. Algorithms for carrying out preprocessing

This section describes a number of ways in which instances of stable matching problems can be preprocessed. As a reminder to the reader, preprocessing is not only useful when searching for a largest stable matching: it can be applied to any problem that searches for stable matchings, including different types of optimality conditions such as egalitarian or minimum-regret, and the problem of counting the number of stable matchings [22].

Preprocessing a preference list in a stable matching problem is possible if a set $\mathcal{P}$ can be found that satisfies Theorem 1, and further preprocessing can be carried out by identifying sets $\mathcal{P}$ and $\mathcal{P}'$ that satisfy Theorem 3. In Section 3.1 we give a number of new heuristics for finding such sets $\mathcal{P}$, and Section 3.2 defines an ILP model to find sets $\mathcal{P}$ and $\mathcal{P}'$. Section 3.3 gives a graph-based representation of preprocessing, along with the proofs demonstrating the equivalence between the graph-based representation and the theory in Section 2. This graph-based representation is used in Section 3.4 to construct two polynomial-time algorithms to identify preprocessing. Lastly, Section 3.5 gives a brief discussion on the application of preprocessing techniques on instances of stable matching problems, rather than on a single preference list.

3.1. Heuristics

We will now define a number of heuristics which can identify sets $\mathcal{P}$ that allow preprocessing according to Theorem 1. While the first heuristic defined in Section 3.1.1 was known in the literature [8] for SMTI, Theorem 3 allows us to extend it to HRT. All other heuristics are new contributions.

Our experimental results show that while heuristics are not guaranteed to find a maximal reduction in the lengths of the preference lists, they can still reduce the solution time of the IP models. We show in Section 4 that the trade-off between preprocessing time and solution time means that, for some scenarios, these heuristics are preferable over more complex methods that take longer to compute.

We note that while our new theory involves sets $\mathcal{P}$ and $\mathcal{P}'$, by definition when preprocessing a candidate c only positions that c does not find acceptable might end up in $\mathcal{P}'$. As such, it is not clear how a suitable set $\mathcal{P}'$ could be easily determined, and as a result, we were unable to construct any useful heuristic that also searched for $\mathcal{P}'$.

We now present our three heuristics for preprocessing.

3.1.1. Descending order of preference

Given a candidate c with preference list L, the most obvious heuristic for finding a suitable $\mathcal{P}$ is to simply add to $\mathcal{P}$ positions p from L in descending order of preference. If, at any point in this process Theorem 1 is satisfied, then preprocessing can occur. This was described in [8], and was shown to be implementable in $O((n_c + n_p)n_p)$ time for a given candidate in an instance of SMTI.

In our research into preprocessing, we noted that the implementation used in [8] could be improved by using a data structure with constant time lookup for storing $\mathcal{P}$. This has no effect on the asymptotic analysis, but we found it to have significantly improved performance under certain scenarios, which is reported in Section 4.

3.1.2. Skip larger

When using the heuristic of Section 3.1.1, there are some positions p that, when added to $\mathcal{P}$, cause a large increase to $|\mathcal{C}|_q$. To avoid such issues, these particular positions p that cause a large increase to $|\mathcal{C}|_q$ can simply be skipped. We implement three such heuristics, which do not add a position p to $\mathcal{P}$, but instead skip over it, if adding p to $\mathcal{P}$ would cause $|\mathcal{C}|_q$ to increase by at least 5, 15, or 50 candidates, respectively. Note that with this terminology, the "descending" heuristic introduced in Section 3.1.1 can also be termed Skip-∞. This will prove useful when discussing the results of our computational experiments.

3.1.3. FindBest

As Theorem 3 requires that $|\mathcal{C}|_q$ be smaller than or equal to $|\mathcal{P}|_q + |\mathcal{P}'|_q$, it seems reasonable to add to $\mathcal{P}$ a position p that increases $|\mathcal{C}|_q$ by the smallest amount possible. Finding such a p is possible by calculating the change in $|\mathcal{C}|_q$ for each position p that has not yet been added.

Given a preference list L, a slightly quicker method for finding a "good" position p to add to $\mathcal{P}$ is to define the set B as the most-preferred tie in L that contains at least one position not already in $\mathcal{P}$. A "good" position is then one in B that also increases $|\mathcal{C}|_q$ by the smallest amount over all positions in B.

3.2. IP Model

We now give an IP model for determining if, given an instance I of HRT, sets $\mathcal{P}$ and $\mathcal{P}'$ can be found that satisfy Theorem 3.

Let I be an instance of HRT with n_p positions and n_c candidates, and let c_k be some particular candidate whose preference list is being processed. Let r_i be the rank of position p_i according to candidate c_k if p_i finds c_k acceptable, and 0 otherwise. Let $\mathcal{P}''$ be a set of positions such that for any $p_i \in \mathcal{P}''$, (c_k, p_i) is not an acceptable pair, and for any stable matching in I, p_i is full in I (we discuss how to determine such a set $\mathcal{P}''$ in Section 3.4.2). Let $A = \{i \mid (c_k, p_i) \text{ is acceptable}\}$, let $B = \{i \mid p_i \in \mathcal{P}''\}$, and let $v_{ij} = 1$ if and only if either (a) $p_i \notin \mathcal{P}''$ and position p_i considers candidate c_j to be at least as good as candidate c_k, or (b) $p_i \in \mathcal{P}''$ and p_i finds c_j acceptable.

Define binary variables x_i for $i \in A \cup B$ that are set to 1 if and only if position p_i is in $\mathcal{P} \cup \mathcal{P}'$, and binary variables y_j that are set to 1 if and only if candidate c_j is in $\mathcal{C}$. Note that if $x_i = 1$ for some $i \in A$, then $p_i \in \mathcal{P}$, and if $x_i = 1$ for some $i \in B$, then $p_i \in \mathcal{P}'$. We also define the variable z to denote the worst rank of any position in $\mathcal{P}$. An optimal set of preferences to be removed from a preference list can then be found by solving the following integer programming model:

$$\begin{aligned}
\min \quad & z && (1)\\
\text{s.t.} \quad & z \geq r_i x_i \qquad i \in A, && (2)\\
& \sum_{i \in A} x_i \geq 1, && (3)\\
& \sum_{i \in A \cup B} x_i q_i \geq \sum_{j=1}^{n_c} y_j q_j, && (4)\\
& x_i v_{ij} \leq y_j, \qquad i \in A \cup B,\ j = 1, \ldots, n_c, && (5)\\
& x_i \in \{0, 1\}, \qquad i \in A \cup B, && (6)\\
& y_j \in \{0, 1\}, \qquad j = 1, \ldots, n_c, && (7)
\end{aligned}$$

Constraints (2) ensures that the variable z will denote the worst rank of any position in $\mathcal{P}$. Constraints (3) ensures that the solution $\mathcal{P}$ is nonempty, and constraints (4) ensures that the sets $\mathcal{P}$, $\mathcal{P}'$, and $\mathcal{C}$ satisfy the capacity requirements of Theorem 3. Lastly, constraints (5) ensure that, if a given position p_i is in $\mathcal{P}$, then any competition that c_k has for p_i is included in $\mathcal{C}$, or, if a given position p_i is in $\mathcal{P}'$, then any candidate that p_i finds acceptable is in $\mathcal{C}$.

3.3. Graph-based representation of preprocessing

We now give a graph-based representation of preprocessing, which we rely on in Sections 3.4.1 and 3.4.2 to construct algorithms to detect preprocessing. These new preprocessing algorithms construct a graph iteratively by walking down the preference list, and check the size of a maximum flow of this graph at each iteration. If a maximum flow is sufficiently small (see Theorem 7 below), then any further entries in the preference list can be removed.

The graph will be built based on truncations of the preference list of some particular candidate c whose preference list will be preprocessed. A *truncation* of a preference list L after t ranks, written L_t, contains the first t tie groups in L. An example of such a truncation is given in Example 6.

Definition 2. *Given an instance I of HRT, a candidate c, a truncation L_t of their preference list L, and a set of positions $\mathcal{P}^*$ such that*

1. *(c, p) is not acceptable for any $p \in \mathcal{P}^*$, and*
2. *any $p \in \mathcal{P}^*$ will be full in any stable matching of I,*

we construct the graph $G[L_t]$ as follows: First, create one source and one sink vertex. For each $p \in \mathcal{P}^*$, add a vertex for p to $G[L_t]$, and add an edge from p to the sink with capacity q_p. Then, for each candidate c' that p finds acceptable, add the edge from c' to p with capacity one to $G[L_t]$ (and the vertex c' if it is not yet present). Then add to $G[L_t]$ one vertex for each position p in L_t, along with an edge from p to the sink with capacity q_p. For each such position p, find all candidates $c' \neq c$ such that $c' \preceq_p c$, and add the edge $\{c', p\}$ to $G[L_t]$ (and the vertex c' if it is not yet present). Lastly, for each vertex representing a candidate c' in $G[L_t]$, add an edge from the source to c' with capacity $q_{c'}$. Note that we specifically avoid having a vertex for c in this construction.

We note that there is a similarity between $\mathcal{P}^*$ from Definition 2, and $\mathcal{P}'$ as used in Theorems 2 and 3, but we highlight the fact that $\mathcal{P}^*$ and $\mathcal{P}'$ are not necessarily the same sets. Theorem 7, which we give later, will detail how $\mathcal{P}'$ may be derived from $\mathcal{P}^*$.

An example of the complete construction of a graph $G[L_t]$ is given in Example 6. However to make the graphs simpler to read, we will employ a number of simplification techniques when drawing such graphs. For any truncation L_t drawn as in Example 6, all of the edges are oriented from left to right. For each candidate c there is one edge from the source to c with capacity q_c, for each position p there is one edge from p to the sink with capacity q_p, and there is no other edge incident with either the source or the sink. With that in mind, we will draw such graphs without the source or sink, and we will say the capacity or flow of candidate c (respectively position p) to mean the capacity or flow of the edge from the source to candidate c (respectively the edge from position p to the sink). Unitary capacities of vertices will not be marked, but non-unitary capacities can be written within brackets next to the vertex names. Example 7 demonstrates a simplified drawing of the graph from Example 6.

Example 6. *We will demonstrate the construction of a network $G[L_t]$ now. Let us take an instance I of HRT with five candidates and four positions such that in any stable matching of I, position p_3 will be assigned to capacity. We only give the required details of the instance for constructing $G[L_2]$, where L is the preference list of c_1. Let some of the preferences of the agents be described by the following:*

$$
\begin{array}{lllll}
c_1: & p_1 & p_2 & p_4 & \\
 & & & & \\
p_1: & c_2 & c_1 & c_3 & \\
p_2: & c_3 & c_2 & c_4 & c_1 \\
p_3: & c_4 & c_5 & &
\end{array}
$$

Within I, all candidates have capacity one, p_1 has capacity two, p_2 has capacity three, and p_3 has capacity two. As we know that p_3 will be assigned to capacity in any stable matching of I, we can also let $\mathcal{P}^* = \{p_3\}$. The preference lists not shown (i.e., those of c_2, c_3, c_4, and p_4) and the capacities not given will not be relevant to the construction of $G[L_t]$. Recall that L_2 is the truncation of L after two ranks. Thus, L_2 is the list p_1, p_2, and we construct $G[L_2]$ as shown in Figure 1.

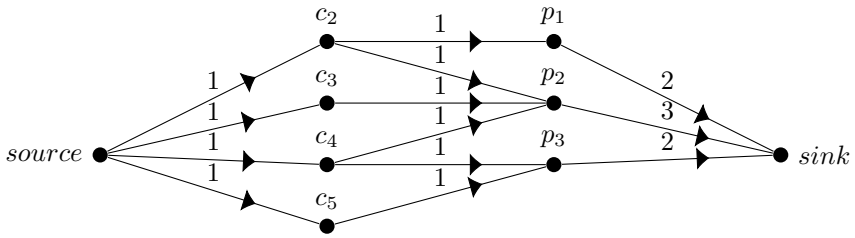


Figure 1: A drawing of $G[L_t]$ from Example 6.

Example 7. *Figure 1 shows an example of $G[L_t]$ drawn. Figure 2 carries the same information as Figure 1, but in a more readable format. For any $G[L_t]$, all edges between a candidate and a position have capacity one, so those capacities are not shown. The edges from the source, or to the sink, have been removed, and the capacities of said edges have instead been either left off (if they are equal to one), or written in parentheses next to the corresponding candidate or position (if they are greater than one).*

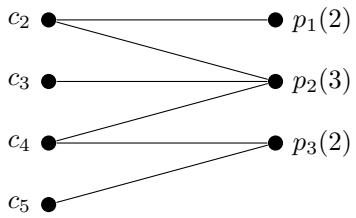


Figure 2: A simplified drawing of $G[L_t]$ from Example 6, with no source or sink as per Example 7.

In the rest of this section, we will use this bipartite representation of $G[L_t]$, and the set U (respectively W) will be used to refer to the set of vertices representing candidates (respectively positions) in this graph.

We begin with the following lemma, which we later use to reason that if there exists in $G[L_t]$ a maximum flow that leaves at least a capacity q_c unused through W, then there must also exist in $G[L_t]$ a (possibly different) maximum flow such that the flow through vertices corresponding to positions in $\mathcal{P}^*$ must be at capacity, meaning that any "unused capacity" is available in vertices that correspond to positions c finds acceptable. This lemma uses concepts from graph theory, specifically maximal flow theory. For a background on this theory, we point the reader to [21].

Lemma 6. *Let I be an instance of HRT, let c be some candidate in I with capacity q_c, let L_t be a truncation of the preference list of c, let $\mathcal{P}^*$ be as defined in Definition 2, let $G[L_t]$ be the graph constructed as per Definition 2, and let W be all the positions represented in $G[L_t]$. If there exists a maximum flow F that satisfies $|F| \leq |W|_q - q_c$, then there exists a flow F' that satisfies $|F'| = |F|$ and the extra criterion that for any $p' \in \mathcal{P}^*$, the flow through p' must equal $q_{p'}$, the capacity of p'.*

Proof. Let F be as given in the lemma. We will write $F(p')$ to denote the flow in F through p', and similarly for flow F' which we will create. Let $p' \in \mathcal{P}'$ be a vertex such that the $F(p') < q_{p'}$, noting that if no such p' exists then the theorem holds.

We will create a new flow F' such that $F'(p') = F(p') + 1$, $F'(p) = F(p) - 1$ for some $p \in W \setminus \mathcal{P}'$, and for any other $p^= \notin \{p, p'\}$, $F(p^=) = F'(p^=)$. The result then holds by repeated iteration of this procedure.

We will build an alternating red-blue path that starts at p' and ends with some $p \in W \setminus \mathcal{P}^*$, at which point swapping along this path gives the new flow F'. We will build this path by colouring the edges between the sets of vertices U and W in $G[L_t]$. In particular, we will use the colours red and blue. Where we say *red-degree* (respectively blue-degree) of a given vertex v, we mean the number of edges incident to v that are red (respectively blue). Note that unlike common graph colourings, we will allow edges to be assigned multiple colours. We will show that edges in our alternating path are mono-coloured, but edges in the graph that are not in our alternating path may be assigned multiple colours.

First, colour all of the edges that carry non-zero flow in F in blue. Note that each edge between U and W has capacity one, and, by construction, any maximal flow of $G[L_t]$ has integral flow through any edge. Thus, an edge between U and W is blue if it is at capacity, and not blue otherwise. Now take a stable matching in I, and colour the edges corresponding to I in red. Again, as any pair can appear at most once in a matching, each edge either is red if the corresponding pair is in the new stable matching, or is not red otherwise.

We now iteratively build an alternating red-blue path, starting at p'. As $p' \in \mathcal{P}^*$, it must have a red-degree of $q_{p'}$. However, it must have a blue-degree less than or equal to $q_{p'} - 1$ as $F(p') < q_{p'}$. Thus, we begin our alternating path by taking an edge that is red, and not blue.

We follow this red edge to some candidate c'. If $F(c') < q_{c'}$, then we can increase the flow in F by augmenting along the current path, contradicting the fact that F is a maximum flow. Thus it must be that $F(c') = q_{c'}$, and so c' must have a blue-degree of $q_{c'}$. Trivially, its red-degree must be at most $q_{c'}$. So, by the existence of the red-and-not-blue edge (c', p'), there must also be a blue-and-not-red edge incident with c'. We then follow this blue-and-not-red edge to some new position p''.

If $p'' \in \mathcal{P}^*$ then p'' must have a red-degree of $q_{p''}$, and a blue-degree of at most $q_{p''}$. By the presence of the blue-and-not-red edge we followed to reach p'', there must be a red-and-not-blue edge incident with p'', and so we can repeat the above procedure. Therefore this process can be repeated until we reach a position $p \in W \setminus \mathcal{P}^*$. At this point, we have an even length alternating red-blue path from p' to p. By the colouring of the red and blue edges, and as p' is not at capacity, we can augment the flow in F by selecting the red edges instead of the blue edges, creating a flow F' such that $F'(p') = F(p') + 1$, $F'(p) = F(p) - 1$, and for any other $p^= \notin \{p, p'\}$, $F(p^=) = F'(p^=)$. □

Theorem 7. *Let L_t be a truncation of the preference list of some candidate c, let $\mathcal{P}^*$ be a set as defined in Definition 2, let $G[L_t]$ be the graph constructed as per Definition 2, and let W be the positions represented in $G[L_t]$. If a maximum flow F of $G[L_t]$ satisfies $|F| \leq |W|_c - q_c$, there exist sets $\mathcal{C}$, $\mathcal{P}$, and $\mathcal{P}'$ that satisfy Theorem 3.*

Proof. By Lemma 6, we can assume without loss of generality that for any $p' \in \mathcal{P}^*$, p' is at capacity in F. We will build up sets U' and W' iteratively, use these to construct $\mathcal{C}$, $\mathcal{P}$, and $\mathcal{P}'$, and then show that these sets satisfy Theorem 3. We build U' and W' in turn by adding positions to W', then candidates to U', and then repeating this process until it terminates.

Let S be the set of positions that are not at capacity in F. Note that the flow in through S in F can be at most $|S|_q - q_c$ by the condition of the theorem. We begin by adding to W' all positions in S, then repeat the following:

1. Add the neighbourhood of any position $p \in W'$ to U'.
2. For any neighbour n of a candidate $c \in U'$, if there is some flow in F from c to n, add n to W'.

First, we note that this process obviously terminates as $G[L_t]$ is a finite graph, and at each step we increase the size of either U' or W'. Let $\{s_1, s_2\}$ be the source and sink in $G[L_t]$ respectively. We will denote by G' the induced subgraph on $\{s_1, s_2\} \cup U' \cup W'$, and we will denote by F' the restriction of F to the subgraph G'. Note that if F' is not a maximum flow, then there is some augmenting path in G' to increase the flow of F'. However, this augmenting path would also exist in $G[L_t]$, which contradicts the fact that F is a maximum flow. Thus, it must be that F' is a maximum flow in G'.

By this construction, if there is a $p \in W'$ and a candidate c' such that c finds p acceptable and $c' \preceq_p c$, then the edge $\{c', p\}$ is in $G[L_t]$, and thus in G'. Similarly, if there is a $p' \in W$ and a candidate c' such that p' is full in any stable matching, (c, p') is not an acceptable pair, but (c', p') is an acceptable pair, then the edge $\{c', p'\}$ is in $G[L_t]$ and thus in G'. Additionally, by the construction of $G[L_t]$, p' is full in any stable matching of I. We make use of these facts at the end of the proof.

Next we show that for any $c' \in U'$, $F'(c') = q_{c'}$. We show this by demonstrating that if a candidate $c' \in U'$ is not at capacity in F', then there is an augmenting path from c' to some undersubscribed p, which contradicts the fact that F' is a maximum flow.

Assume that some $c' \in U'$ is undersubscribed. Without loss of generality, take $c' \in U'$ that is not at capacity such that $(p, c_1, p_1, c_2, \ldots, c_k, p_k, c')$ is a shortest path in G' where:

1. p is not at capacity;
2. c_i is at capacity for $i \in \{1, \ldots, k\}$; and
3. there is flow from c_i to p_i in F' for $i \in \{1, \ldots, k\}$.

Such a path is guaranteed to exist by construction as any position is only ever added to G' if it either is not at capacity, or if there is flow from some c_i already in G'. However, then this path is also an augmenting path for F' in $G[L_t]$, which contradicts the fact that F' is a maximum flow.

As each candidate $c' \in U'$ is at capacity, it is clear that the flow F' must satisfy $|F'| = |U'|_q$. By construction, we also have a set of positions $S \subseteq W$ such that the flow through S is at most $|S|_q - q_c$. As any other position in W' can at most be at capacity, this gives us $|F'| \leq |W'|_q - q_c$. We lastly define $\mathcal{C} = U' \cup \{c\}$, $\mathcal{P} = \{p \mid p \in W' \text{ and } c \text{ finds } p \text{ acceptable }\}$, and $\mathcal{P}' = W' \setminus \mathcal{P}$. It clearly follows that $|\mathcal{C}|_q = |F'| + q_c \leq |W'|_q = |\mathcal{P}|_q + |\mathcal{P}'|_q$, and by the earlier paragraph these sets satisfy the criteria of Theorem 3, concluding the proof. □

Given a preference list L, we can truncate it to create L_t for any positive t. If a truncation L_t satisfies Theorem 7, then preprocessing according to Theorem 3 is possible with $\mathcal{P} \subseteq L_t$. The following theorem states that the reverse is also true: that if preprocessing with $\mathcal{P}$ according to Theorem 3 is possible, then for any positive t such that $\mathcal{P} \subseteq L_t$, L_t will satisfy the criteria of Theorem 7.

Theorem 8. *Given a candidate c and their truncated preference list L_t, if sets $\mathcal{C}$, $\mathcal{P}$, and $\mathcal{P}'$ exist that satisfy Theorem 3 such that $\mathcal{P} \subseteq L_t$, then the graph $G[L_t]$ has a maximum flow F that satisfies $|F| \leq |W|_q - q_c$, where W is the set of positions represented in $G[L_t]$.*

Proof. Construct $G[L_t]$ as per the definition, recalling that U and W denote the candidates and positions represented in $G[L_t]$ respectively, $W = \mathcal{P} \cup \mathcal{P}'$, and that $U = \mathcal{C} \setminus \{c\}$. The result then holds as any maximum flow F must satisfy $|F| \leq |\mathcal{C}|_q - q_c$ (as $U = \mathcal{C} \setminus \{c\}$, and all flow must pass through U). So, by $|\mathcal{C}|_q \leq |\mathcal{P}|_q + |\mathcal{P}'|_q$ we have $|F| \leq |W|_q - q_c$. □

3.4. Graph-based algorithms

We now give the algorithms that use the construction and theorems from the previous section to identify preprocessing. For a preference list L of some agent, the algorithms will consider all truncations L_t of L in ascending order of length (i.e., t), and construct the graph $G[L_t]$ per Definition 2.

If a maximum flow is too large in a given $G[L_t]$ (i.e. $|F| > |W|_q - q_c$), then Theorem 8 tells us that the preference list cannot be preprocessed at this tie level, and longer truncations of L must be considered. However, Theorem 7 states that if a maximum flow is small enough, preprocessing is possible.

We are also able to take advantage of our particular method of constructing $G[L_t]$. We build graphs $G[L_t]$ iteratively, and as we will have a maximum flow at each step, we only need to attempt to augment this flow at each step, rather than trying to re-build a complete maximum flow. In addition, we also know that any such augmenting path must include the newest vertex added, else the augmenting path would exist in the previous iteration, a contradiction. Exploiting these features in the implementation of our algorithms significantly improved their performance.

Section 3.4.1 gives a basic algorithm that can identify a set $\mathcal{P}$ such that preprocessing can be performed (if such a set exists), but this algorithm will always assume that $\mathcal{P}' = \emptyset$. In Section 3.4.2 we then extend this algorithm to also identify a suitable set $\mathcal{P}'$. In Section 4 we will see that this second algorithm, while identifying more preprocessing, can also take significantly longer to run.

3.4.1. Graph-based algorithm to find $\mathcal{P}$

Algorithm 1 describes the process of building the graphs $G[L_t]$ and finding a maximum flow. If this algorithm returns some $t \geq 1$, then any preference list entries appearing after the t-th tie can be removed. The algorithm also may return the sentinel value -1, if no preprocessing was detected. Note that this algorithm only considers the set $\mathcal{P}$; it does not attempt to find a set $\mathcal{P}'$. The extension to also find $\mathcal{P}'$ is described in Section 3.4.2.

Given an instance of HRT with n_c candidates and n_p positions, we must run Algorithm 1 for each candidate c. Algorithm 1 iterates over each element in the preference list of c exactly once (lines 4-5) for $|L|$ iterations. We then attempt to extend the current flow by finding augmenting paths from each of the newly added vertices that correspond to a position. Each position p is only added to G once, but we need to search for up to q_p augmenting paths. Thus, we need to search for augmenting paths a total of $\sum_{p \in L} q_p$ times. Each such a search could visit each of the $O(n_c|L|)$ edges in G at most once. This gives an asymptotic running time of $O(n_c|L| \cdot \sum_{p \in L} q_p)$ for preprocessing one preference list of length L, and such a preference list has length at most n_p, so preprocessing a candidate takes $O(n_c n_p \sum_{p \in L} q_p)$ time.

Algorithm 1 Preprocess one preference list

Input: Candidate c with preference list L that is to be preprocessed
Output: A rank r such that in a stable matching, c will be allocated a position with rank at most r
Let $W = \emptyset$, let $F = \emptyset$, let $G[L_t]$ be an empty graph
for each rank r in L in descending order of preference **do**
    for each p in the r-th rank of L **do**
        Add a vertex p to $G[L_t]$
        Add p to W
        for each $c_j \neq c$ that p considers at least as good as c **do**
            if $c_j \notin V(G[L_t])$ **then**
                Add a vertex c_j to $G[L_t]$
            end if
            Add the edge $\{c_j, p\}$ to $G[L_t]$
        end for
        Attempt to find an augmenting path from p to increase the size of F
    end for
    if $|F| \leq |W|_q - q_c$ **then**
        return r
    end if
end for
return -1 ▷ To indicate that no guarantee of matching can be given

In an instance of HRT, if some p has non-unitary capacity, then all candidates must have unitary capacity, and so n_c is a trivial upper bound to the set of capacities of the candidates. This results in a running time of $O(n_c^2 n_p^2)$ for one candidate, or $O(n_c^3 n_p^2)$ to complete one pass of preprocessing over all agents.

Example 8 (Example of use of Algorithm 1)**.** *Consider an instance of HRT with $n_c = 4$ and $n_p = 4$ (i.e., there are 4 candidates and 4 positions). Note that as we are demonstrating only Algorithm 1, $\mathcal{P}' = \emptyset$. We will preprocess the preference list for c_1. Let q_{c_1}, the capacity of c_1, be 2, and let $q_{c_2} = 1$. As this is an instance of HRT, and $q_{c_1} > 1$, we know that every position must have unitary capacity, and so $|W|_q = |W|$ for any set of positions W. The relevant preferences are as follows:*

$$
\begin{array}{llllll}
c_1: & p_1 & p_2 & p_3 & p_4 \\
\\
p_1: & [c_1 & c_2] & c_4 & c_3 \\
p_2: & c_2 & c_1 & c_3 & c_4 \\
p_3: & c_2 & c_1 & c_3 & c_4 \\
p_4: & c_2 & c_3 & c_1 & c_4.
\end{array}
$$

Recall that we will be building a graph $G[L_t]$ based on truncations of the preference list of c, and that the source and sink in $G[L_t]$ are not drawn to simplify the diagrams. At each step, we will look for a maximum flow F, and, if $|F| \leq |W| - 2$, we stop and perform the actual preprocessing.

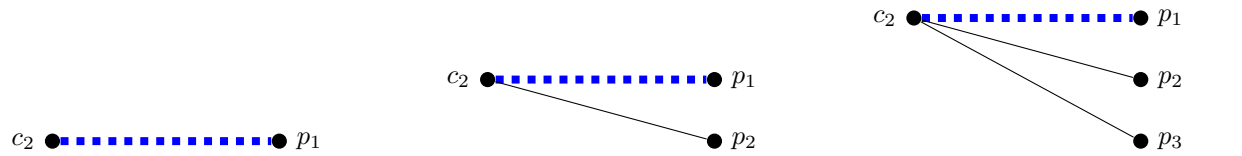
We start with an empty graph $G[L_t]$. First, we add a vertex for p_1 to the right side of $G[L_t]$ (as p_1 is the most preferred choice of c_1), and then we add all candidates who can compete with c_1 for position p_1 to the left side of $G[L_t]$. In our example, this is only c_2. We also add an edge from p_1 to c_2 to indicate that, if c_2 is paired with p_1, then (c_1, p_1) will not form a blocking pair. See Figure 3a.

The maximum flow, indicated in dashed blue in Figure 3a, has size 1. As $|W| = 1$, $|F| > |W| - 2 = -1$ and so we cannot preprocess.

On the next step, we add a vertex for p_2 to the right side of $G[L_t]$. As there is already a vertex for c_2 in $G[L_t]$, we do not need to add a new vertex, but only a new edge from p_2 to c_2. In this new graph, a maximum

flow (indicated in dashed blue in Figure 3b) still has size 1, and now $|W| = 2$*. However,* $|F| \not\leq 2 - 2 = 0$*, so we must continue the algorithm.*

The algorithm continues with the next step, adding a vertex for p_3 *to the right side of* $G[L_t]$*. Again, there is already a vertex for* c_2 *in* $G[L_t]$*, so we only need to add a new edge from* p_3 *to* c_2*. In this graph, a maximum flow (indicated in dashed blue in Figure 3c) still has size 1, but now* $|W| = 3$*. Then* $|F| \leq |W| - 1$*, and we can preprocess. This means we can remove from the preference list of* c_1 *any positions that* c_1 *considers worse than* p_3*, the last position we added. In this case, this means removing* p_4 *as an option for* c_1*, and removing* c_1 *as an option for* p_4*. Note that we also now know that* c_1 *will definitely be matched with some position.*



(a) The graph $G[L_1]$ after one step of Algorithm 1. The edge marked in dashed blue is the maximum matching.

(b) The graph $G[L_2]$ after two steps of Algorithm 1. The dashed blue edge is one of two maximum flows, both with size one.

(c) The graph $G[L_3]$ after three steps of Algorithm 1. The dashed blue edge is one of three maximum flows, all with size one.

Figure 3: Two steps in preprocessing for Example 8.

Example 9 (Example of Algorithm 1 when ties are present in list being processed)**.** *Consider an instance of SMTI with* $n_c = 4$ *and* $n_p = 3$ *(i.e., there are 4 candidates and 3 positions). Note that as we are demonstrating only Algorithm 1,* $\mathcal{P}' = \emptyset$*, and as we have an instance of SMTI, not HRT, we know that every agent has unitary capacity, and thus* $|W|_q = |W|$*. We will preprocess the preference list for* p_1*. Note that this means we swap the nomenclature for positions and candidates in this example. The relevant preferences are as follows:*

$$\begin{array}{llllll} c_1: & p_1 & p_2 & & \\ c_2: & p_1 & p_2 & p_3 & \\ c_3: & p_2 & p_3 & p_1 & \\ c_4: & p_2 & p_1 & p_3 & \end{array}$$

$$p_1: \quad [c_1 \quad c_2] \quad c_4 \quad c_3$$

Again starting with an empty graph $G[L_t]$*, we first add vertices for both* c_2 *and* c_1 *to* $G[L_t]$*. We add vertices for both as they are tied for most preferred candidates by* p_1 *of the candidates who have not yet been added. However, for both* c_1 *and* c_2*,* p_1 *is the most preferred position, and so* p_1 *has no potential competition. This leaves us with a graph of two vertices on the right, but no edges. Therefore* $|F| \leq |W| - 1$ *and so we can preprocess the preference list of* p_1 *and remove candidates appearing after the tie* $[c_1 c_2]$*. That is, we remove* c_4 *and* c_3 *as options for* p_1*.*

3.4.2. Extended graph-based algorithm to find $\mathcal{P}$ *and* $\mathcal{P}'$

Algorithm 1 does not attempt to find sets $\mathcal{P}'$ as defined by Theorem 3. To extend Algorithm 1 to support such sets, we need to know some set $\mathcal{P}''$ of positions that are guaranteed to be full in any stable matching. Note that $\mathcal{P}''$ and $\mathcal{P}'$ are not the same — for a candidate c, $\mathcal{P}'$ is a subset of $\mathcal{P}''$ containing only positions that c does not find acceptable. A position p can be added to $\mathcal{P}''$ if preprocessing the preference list of p results in the removal of some entries from p's preference list. We can also add a position p to $\mathcal{P}''$ if, when preprocessing the preference list L of p, L has a length of t and preprocessing indicates that L can be truncated after t ranks. This would result in a truncation of zero preferences from the preference list of p, but this still means that p is always full in any stable matching. Given this set $\mathcal{P}''$, and a candidate c, we can determine the set $\mathcal{P}'$. We use this technique to "pre-fill" the graph $G[L_t]$ by replacing line 3 in Algorithm 1 with Algorithm 2.

Using Algorithm 2 in conjunction with Algorithm 1 in this manner results in a complexity of $O(n_c^3 n_p^2)$, the same as for Algorithm 1 by itself, but we show in Section 4 that using Algorithm 2 can under some circumstances significantly increase the runtime for preprocessing, and is not a guaranteed overall performance improvement.

Algorithm 2 Prefill a graph-based on one preference list

Input: Candidate c, and a set $\mathcal{P}''$ containing positions guaranteed to be filled in any stable matching
Output: A graph G, a set W of positions added to G, and a matching M of G
Let $W = \emptyset$, let $F = \emptyset$, let G be an empty graph
for each position $p \in \mathcal{P}''$ **do**
    if c does not find p acceptable **then**
        Add a vertex p to G
        Add P to W
        for each c' that p finds acceptable **do**
            if $c' \notin V(G[L_t])$ **then**
                Add a vertex for c' to $G[L_t]$
            end if
            Add the edge $\{c', p\}$ to $G[L_t]$
            for $k = 1, \ldots, q_p$ **do**
                Attempt to find an augmenting path from p to increase the size of F
            end for
        end for
    end if
end for
return W, M, $G[L_t]$

Example 10 (Example of Algorithm 2)**.** *Consider an instance of SMTI with $n_c = n_p = 4$ (i.e., there are 4 candidates and 4 positions). As we have an instance of SMTI, not HRT, we know that every agent has unitary capacity, and thus $|W|_q = |W|$. We will preprocess the preference list for c_3. The relevant preferences are as follows:*

$$
\begin{array}{llllll}
p_1: & c_1 & c_2 & & & \\
p_2: & c_1 & c_2 & c_3 & & \\
p_3: & c_2 & c_3 & & & \\
p_4: & c_2 & c_3 & & &
\end{array}
$$

$$
\begin{array}{llllll}
c_1: & p_1 & p_2 & & & \\
c_2: & p_1 & p_2 & p_3 & p_4 & \\
c_3: & p_2 & p_3 & p_4 & &
\end{array}
$$

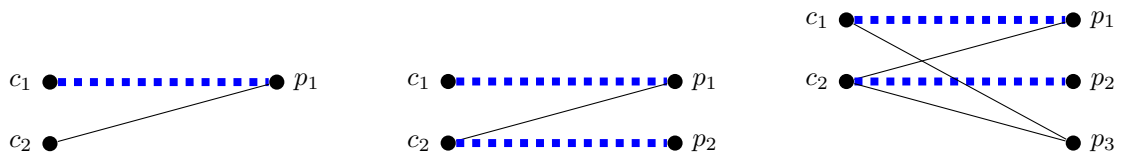
We also know that $p_1 \in \mathcal{P}'$. That is, in any stable matching, position p_1 will always be matched to some candidate. See the first paragraph of Section 3.4.2 for details on how $\mathcal{P}'$ may be determined.

Starting with an empty graph $G[L_t]$, we first add a vertex for p_1 to the right of $G[L_t]$, and then one vertex for each acceptable candidate for p_1 (i.e., c_1 and c_2) to the left side of $G[L_t]$. We also add an edge from each of c_1 and c_2 to p_1. See Figure 4a.

We then start adding vertices by looking at the descending preferences of c_3. First is p_2, which we add to the right side of $G[L_t]$, along with edges from p_2 to both c_1 and c_2, so the graph looks like Figure 4b with a maximum flow in dashed blue. At this point, $|F| > |W| - 1$, and so we cannot preprocess.

The next position to consider is p_3, which is handled similarly to p_2 and leaves us with Figure 4c, along with a maximum flow in dashed blue. We can see that as $|F| = 2$, and $|W| = 3$, we have $|F| \leq |W| - 1$, so we can preprocess. This means removing the position p_4 from the preference list of c_3.

Note that without using the fact that p_1 is always matched to one of either c_1 or c_2, we would not be able to determine that c_3 will never be matched with p_4 in a stable matching. This holds even though c_3 does not find p_1 acceptable.



(a) The graph $G[L_1]$ after one step of Algorithm 2. The edge marked in dashed blue is a maximum flow.

(b) The graph $G[L_2]$ after two steps of Algorithm 2. The dashed blue edges make up the maximum flow.

(c) The graph $G[L_3]$ after three steps of Algorithm 2. The dashed blue edges make up one of three maximum flow.

Figure 4: Three steps for preprocessing for Example 10.

3.5. Iteration of preprocessing

All of the preprocessing described earlier in this section explains how to preprocess the preference list of one agent. This must obviously be repeated for each agent, but in addition to this, it is possible that the removal of preferences from the list of agent c_i may make preprocessing possible for agent p_j. There is also a second consideration if Algorithm 2 is used (i.e., if $\mathcal{P}'$ is non-empty). In such cases, any changes to $\mathcal{P}'$ also justify the re-running of preprocessing over all agents.

Our implementation of all preprocessing methods is iterative — each agent has its preference list preprocessed and if any preprocessing is found, the procedure is repeated for all agents. It is possible to only repeat the preprocessing algorithm on the agents whose preferences lists were modified as a result of earlier preprocessing, but initial investigations showed that this had minimal effect on the total runtime of the preprocessing.

For any iteration method, there is at worst a polynomial number of iterations as each iteration must either remove a preference (of which there are at most $O(n_c n_p)$), or mark an agent as always being full who had not yet so marked (and there are $n_c + n_p$ agents in total).

It is plausible to assume that if the preprocessing of the first $x\%$ of agents results in no changes (for various values of x), then there will be no preprocessing to complete for any remaining agents either. This assumption was tested for $x \in \{5\%, 10\%, 25\%, 50\%\}$, and we report none of these percentages resulted in a noticeable effect on the running time of the preprocessing. This agreed with the observation that often the earlier iterations of preprocessing both took longer, and removed more preference list entries. In contrast, the final iterations of preprocessing that removed few or no preferences also ran relatively quickly.

4. Computational results

This section describes our computational experiments on the effectiveness of preprocessing on a number of different classes of stable matching problems. These include MAX-SMTI, MAX-WT-SMTI-GRP, and MAX-HRT, and are described in Section 4.1. Section 4.2 details the particular IP models used to solve these problems, and Section 4.3 describes the experimental setup that we used. Section 4.4 demonstrates that it is rare for any stable matching problem to contain acceptable pairs that do not affect the set of stable matchings, but that our theory (and thus Algorithm 1 combined with Algorithm 2) is unable to detect. Section 4.5 describes the format of our results, including the presentation of resulting data. We show in Section 4.6 that our new preprocessing techniques improve solution times on MAX-SMTI-GRP by up to 44%. In Section 4.7 we look closely at the impact of preference list lengths on the usefulness of preprocessing when solving MAX-SMTI, showing that preprocessing is more effective when agents deem at least 40% of all potential partners as acceptable. Section 4.8 demonstrates that while our new approaches can detect more preprocessing, doing so is not significant when solving MAX-HRT1S. However Section 4.9 shows that when solving MAX-HRT2s, preprocessing can reduce the overall time to find a solution by over 50%. We conclude this section with a discussion in Section 4.10.

4.1. Problem instances

We tested three different types of problems: SMTI, SMTI-GRP, and HRT. New instances for this experiment are available from `http://dx.doi.org/10.5525/gla.researchdata.904`, and were generated using the same code as [8]. All other instances were taken from the literature [8], and are available at `https://researchdata.gla.ac.uk/664/`.

For SMTI-GRP, we solved MAX-WT-SMTI-GRP on 22 real-world instances of the Coram application, each with 550 agents on one side and 894 agents on the other. We also solved MAX-WT-SMTI-GRP on two sets of 220 instances, generated to resemble the real-world instances. Each of these two correspond to values of a parameter $\kappa \in \{1, 2\}$, where the number of agents on one side is $\kappa \times 550$ and the number of agents on the other side is $\kappa \times 894$.

For SMTI we first solved MAX-SMTI on a set of 270 instances: 10 each of the 27 possible ways of combining each possible set of parameters taken from the following:

- $n_c = n_p$ (number of candidates or positions): $\{10000, 15000, 50000\}$,
- p (preference list length): $\{3, 5, 10\}$, and
- d (tie density): $\{0.75, 0.85, 0.95\}$.

The number of candidates (n_c) and positions (n_p) were equal for each instance of SMTI. The parameter "preference list length" is the length of the preference list of each candidate, with each position p ranking exactly those candidates who find p acceptable. Preference list entries were ordered uniformly at random. The tie density is the probability of a preference list entry being tied with its successor.

Our results show that preprocessing has significantly more impact when solving MAX-WT-SMTI-GRP compared to MAX-SMTI. Three factors differ between the MAX-WT-SMTI-GRP instances and the MAX-SMTI instances: the MAX-WT-SMTI-GRP instances had

- longer preference lists,
- a different objective (maximise weight rather than maximise size), and
- some candidates (respectively positions) being more popular amongst all positions (respectively candidates).

Some agents being more popular is not rare in stable matching problems, and this can be modelled using the *skew* parameter from the generator of Irving and Manlove [14]. A more complete investigation of the skew in our instances of SMTI-GRP is given in [8]. We also can model the weight of a pairing (c, p) by summing together the respective Borda score they assign each other[2]. We generate an additional set of instances to investigate the effect of each of these parameters. These additional instances all have $n_c = n_p = 250$, with preference lengths in $\{100, 150, 200\}$, and skew values in $\{5, 10, 15, 20\}$. For each possible selection of values, we randomly generated 10 different instances.

The HRT instances were split into two distinct groups. The first only allowed ties in the preference lists of the hospitals, to allow the comparison with "Hospitals offer" and "Residents apply" [11], while the second set also allows the doctors to have ties in their preference lists.

The first set of instances are denoted HRT-1S (for one-sided ties), and include 3 proprietary instances taken from the Scottish Foundation Allocation Scheme (SFAS), as well as a number of randomly-generated instances intended to simulate this historical real-world application taken from the literature. The randomly-generated instances are again based on all combinations of the following parameter values:

- κ (size): $\{1, 2, 3, 5, 10\}$, and
- μ (master list): $\{0, 5, 15, 25\}$.

[2]The Borda score that c assigns to p is equal to the number of ranks in c's preference list minus the rank of p according to c.

Size is an indicator of the number of doctors and hospitals in the instance, where a value of κ indicates that the instance has $\kappa \times 759$ doctors and $\kappa \times 53$ hospitals with a combined $\kappa \times 775$ available posts. The trio of numbers $(759, 53, 775)$ is taken as an average of the corresponding parameters in the SFAS instances. The second parameter, μ, is used here to replicate the presence of master lists of doctors, a common theme in real-world applications. A value of $\mu = 0$ indicates that no master list is present (and hospital preferences are assigned randomly). Otherwise each doctor is randomly given a score in the range $\{1, \ldots, \mu\}$ and hospital preferences are derived from these scores. Doctor preferences are randomly assigned in each instance of HRT.

The second set of instances, which we have generated, are denoted HRT-2S (for two-sided ties) as they allow doctors to also express ties in their preference lists. These instances have a tie density of 0.75, and all combinations of size taken from $\{1500, 3000, 4000, 8000\}$, and preference length taken from $\{5, 10, 25, 50\}$. This set of randomly-generated instances are intended to test the scalability of our techniques on instances of HRT with two-sided ties and their applicability to the current UK-wide scheme for assigning residents to hospitals, the UK Foundation Programme. The UK Foundation Programme currently has a size of approximately 8000, and doctors are allowed to express up to 20 preferences [1], but further specifics on tie density and actual distribution of preference lengths are not known as they are not published.

4.2. Models for solving stable matching problems

The effectiveness of preprocessing is best determined by how much it reduces the time taken to solve the problem, but this requires a suitable benchmark method for solving each problem. As MAX-SMTI, and by extension, MAX-HRT, are NP-hard, IP models are commonly used to find optimal solutions [8, 19, 20, 25]. In [8], the authors introduce and compare a number of different modifications of IP models for stable matching problems. We use the best model (or set of models, if the best model is not clear) according to [8] as the chosen model(s) for these problems. The three important modifications that can be made are as follows:

- Stability constraint merging — the merging of stability constraints for a set of agents who are all tied in some preference list.
- Dummy variables — the introduction of dummy variables that denote whether an agent is matched with any other agent at a given rank or better.
- Double stability constraints — the use of stability constraints derived from both sets of agents.

For SMTI, we test with two IP models (models M3 and M4 from [8]). Although both of these models utilise stability constraint merging, they differ in that M4 uses dummy variables and M3 does not. For SMTI-GRP, we again test with two IP models (models M4 and M6 from [8]). Whilst both utilise dummy variables, M4 uses stability constraints derived from the preferences of one set of agents while M6 uses double stability constraints. For HRT, we use two different models. For HRT-1S, we use model N8 from [8], which uses both merged stability constraints and dummy variables. However, N8 uses specific techniques that rely on a lack of ties from the doctors, and as such is unsuitable for HRT-2S. Instead, we use model N4 for HRT-2S. The models were selected as they were reported to be the best choice(s) for each given problem type. For more complete details on these models, we refer the reader to [8].

4.3. Experimental setup

The specific code for both the preprocessing and the IP models for finding optimal solutions is available from `https://dx.doi.org/10.5281/zenodo.3956390`, and was compiled with GCC 7.2.0 using -O2 with Gurobi 7.5.1 as the IP library. All tests were carried out on a computing cluster with two Intel Xeon E5-2687W v3 CPUs per node, each running at 2.60 gigahertz and with 512 gigabytes of memory. At any one time, up to 32 instances were running on a single node (corresponding to the 32 physical cores on each node), and each instance was limited to 15 gigabytes of memory, with the exception of the SMTI instances with 50000 agents, in which case only 8 instances at a time were run with 63 gigabytes of memory each. Time limits were set to one hour combined for both preprocessing and IP solving.

4.4. Completeness of preprocessing

Theorem 3 proves that certain pairs in an instance of a stable matching problem can be removed without affecting any solution, and Theorem 8 shows that Algorithm 1 combined with Algorithm 2 is guaranteed to detect if such preprocessing is possible. However, as demonstrated in Example 3, there exist pairs in instances of stable matching problems that never appear in a stable matching, but that Theorem 3 will not be able to detect as never appearing in a stable matching. To test the completeness of our preprocessing, we took each instance I of SMTI with 10000 agents per side and ran Algorithm 1 combined with Algorithm 2 to produce a preprocessed instance I'. For each acceptable pair (c, p) in I', we tested whether a stable matching exists that contains (c, p). Results showed that for each of the 90 instances, each acceptable pair (c, p) in I' appears in some stable matching of I'. This suggests that instances of stable matching problems containing pairs that are not in any stable matching, but are not able to be preprocessed by our theory, are rare. We did find one such example, Example 3, but finding this example involved a computational search through thousands of instances of stable matching problems with only four candidates and four positions.

4.5. Presentation of results

The results on the following pages omit some methods of preprocessing. Initial testing on the real-world MAX-WT-SMTI-GRP instances (see below) showed that the two methods from Section 3.1.3 were orders of magnitude slower than Algorithm 1, while finding only a subset of the preferences that Algorithm 1 removed. In addition, in these tests using the IP model from Section 3.2 for preprocessing proved to be too computationally expensive to run any thorough tests on. The implementation required that we solve $n_c + n_p$ different IPs for each iteration, and whereas most preprocessing techniques took seconds or minutes to complete, the IP-based algorithm would take days to complete for a single instance.

Table 1 describes the various preprocessing methods tested. We tested our new methods against not using any preprocessing, P0, and three existing methods, P1, P1', and P1*. P1 is the descending heuristic [8], defined in Section 3.1.1, and P1' is our improved implementation of P1. Both P1 and P1' were only used for SMTI, SMTI-GRP, and HRT2S. For HRT1S, we replaced both of P1 and P1' with P1*, the "Hospitals-offer" and "Residents-apply" methods [14]. We compared these against P2, P3, and P4, three new heuristic methods, P5, our new graph-based algorithm, and P6, the extended version of our new graph-based algorithm.

Table 1: The different preprocessing methods shown in the results section.

Method	Description
P0	No preprocessing
P1	Descending heuristic (for SMTI, SMTI-GRP, and HRT-2S only) [8]
P1'	Improved implementation of P1 (for SMTI, SMTI-GRP, and HRT-2S only)
P1*	"Hospitals offer" and "Residents apply" (for HRT-1S only) [11]
P2	Skip positions if $\|\mathcal{C}\|_q$ would increase by 5 or more
P3	Skip positions if $\|\mathcal{C}\|_q$ would increase by 15 or more
P4	Skip positions if $\|\mathcal{C}\|_q$ would increase by 50 or more
P5	Algorithm 1 (graph-based)
P6	Algorithm 1 extended with Algorithm 2

Each results table describes various parameters of our experiments as follows. The five left-most columns in each table correspond to the preprocessing method. The "Name" column is a reference to the type of preprocessing used, as per Table 1. The "Preferences removed" column is an average of the number of preference list entries removed from each instance, and the "Prop. removed" column, short for "proportion removed", indicates what proportion of preference list entries from the whole instance have been removed. The "Runtime" is the average running time of the preprocessing step, and the "num. comp." column, short for "number completed", is the number of instances for which the preprocessing step completed in less than 3600 seconds. The remainder of each table is split into two sets (for SMTI), or one set (for HRT) of three columns, one set of three for each IP model tested for that particular problem type (models M4 and M6 for SMTI-GRP, models M3 and M4 for SMTI, and either model N4 or model N8 for HRT). As a reminder,

the rationale for the selection of these models is given in Section 4.2. Of these two sets of three columns in the results tables, the "num. opt." column, short for "number optimal", indicates how many instances were solved to optimality, the "IP Solve" column indicates the average time taken to solve the IP model, and the "Total" column is the average of the combined preprocessing and solving time for the given preprocessing method and IP model. Note that preprocessing is run twice per instance in cases where two IP models were tested. We only show running times from one of these to avoid cluttering the table, which explains why the sum of the preprocessing runtime and the IP solve runtime does not always equal the total runtime. All runtimes are given in seconds. A combined time-limit of 3600 seconds was set for all experiments, and if an optimal solution was not found (including when preprocessing was not completed in under 3600 seconds), the IP solution time was set to 3600. This was done so that the calculations of averages (which include all instances, even those not completed to optimality) did not favour scenarios where the preprocessing took longer (and hence less time was given for the IP solver).

4.6. Results on instances of SMTI-GRP

Table 2 shows results from the real-world instances, indicating that (just as in [8]) preprocessing can reduce total running time significantly. P5 and P6 both remove significantly more preferences than P1, which would, in turn reduce the model size, at the cost of an increase in the running time of the preprocessing step. The best performance is achieved by preprocessing method P5, combined with IP model M6, reducing total running time by approximately 44% compared to using existing preprocessing techniques, or 80% compared to not using preprocessing. We note that with preprocessing method P1, model M6 significantly outperformed model M4, but when using preprocessing method P6, M4 was solved in only 45 seconds. This was the fastest IP solution time for any choice of model or preprocessing technique by a significant margin.

Comparing preprocessing methods P5 with P6, we see that P6 takes almost four times as long to run (38 seconds compared to 11 seconds), while only removing approximately 4% more preference list entries. However, the removal of this extra 4% of preference list entries almost halves the time taken to solve the IP using M4, showing that the number of preference list entries removed is not necessarily a good indicator for the reduction in model solution times.

Table 3 shows that for the augmented instances of a similar size to the real-world data, the new graph-based algorithms again remove significantly more preferences than pre-existing preprocessing methods. However, the reduction in running times is not as impressive, showing only an approximate 19% reduction in total run time compared to existing preprocessing techniques, or 56% compared to not using preprocessing. When we look at augmented instances twice as large as the real-world data (Table 4), we see that no combined method is yet able to solve all 220 instances. Still, our new preprocessing methods are an improvement over existing methods, with P6 and M4 solving 129 instances compared to the 120 solved using P1 and M4, or only 103 solved using M4 without preprocessing.

Table 2: Comparison of preprocessing methods on 22 real-world instances.

Preprocessing method					M4			M6		
Name	Preferences removed	Prop. removed	Runtime (s)	num. comp.	num. opt.	IP Solve (s)	Total (s)	num. opt.	IP Solve (s)	Total (s)
P0	—	—	—	22	22	404	404	22	465	465
P1	94606	35.5%	25	22	22	222	246	22	150	174
P1'	94606	35.5%	3	22	22	209	212	22	146	149
P2	90682	36.5%	2	22	22	350	352	22	402	404
P3	76892	32.2%	8	22	22	266	274	22	157	165
P4	90005	34.5%	33	22	22	190	223	22	159	192
P5	165613	61.4%	11	22	22	79	90	22	72	83
P6	172524	63.3%	38	22	22	46	84	22	90	128

4.7. Results on instances of SMTI

For SMTI, we break down results by the length of preference lists rather than by size, as this highlights when preprocessing is useful. Note that P6 could not even finish preprocessing some of these instances

Table 3: Comparison of preprocessing methods on 220 augmented instances with $\kappa = 1$.

Preprocessing method					M4			M6		
Name	Preferences removed	Prop. removed	Runtime (s)	num. comp.	num. opt.	IP Solve (s)	Total	num. opt.	IP Solve (s)	Total (s)
P0	—	—	—	220	220	151	151	220	164	164
P1	83093	25.2%	25	220	220	82	107	220	79	104
P1'	83093	25.2%	3	220	220	82	85	220	79	81
P2	72851	27.2%	4	220	220	129	133	220	212	215
P3	64475	21.7%	11	220	220	139	150	220	184	195
P4	78385	24.3%	40	220	220	105	145	220	99	139
P5	180888	67.1%	10	220	220	77	87	220	56	65
P6	185438	68.2%	27	220	220	55	83	220	50	76

Table 4: Comparison of preprocessing methods on 220 augmented instances with $\kappa = 2$.

Preprocessing method					M4			M6		
Name	Preferences removed	Prop. removed	Runtime (s)	num. comp.	num. opt.	IP Solve (s)	Total (s)	num. opt.	IP Solve (s)	Total (s)
P0	—	—	—	220	103	2334	2334	100	2326	2326
P1	276752	18.1%	267	220	110	2091	2358	112	2049	2319
P1'	276752	18.1%	22	220	120	2252	2274	119	2234	2257
P2	210545	19.4%	22	220	106	2274	2296	101	2301	2322
P3	178358	14.0%	79	220	102	2289	2368	98	2320	2395
P4	231256	16.1%	329	220	105	2040	2369	100	2055	2382
P5	625791	56.9%	112	220	128	2105	2217	118	2165	2280
P6	646492	58.1%	315	220	129	1900	2215	120	1987	2308

in under 3600 seconds. For such instances, the IP solution time was set to 3600 seconds, as discussed in Section 4.5, even though without preprocessing, all the instances could be solved. The reader should be aware that this means the average figures in the "IP Solve" column may be misleading if P6 is able to preprocess fewer instances than other methods. As a reminder, the "num. comp." column lists the number of instances on which preprocessing successfully completed within the 3600 second time limit, and each of Tables 5, 6, and 7 lists details on running 90 different instances.

Table 5 shows that when preferences have length 3, heuristic methods run in under a second on average, while P5 and P6 take 50 seconds and almost 30 minutes on average, respectively. The preprocessing appears to have minimal effect on the IP solution time, so when preferences are very short preprocessing is not useful. This is consistent across both IP models tested.

Table 5: Comparison of preprocessing methods on 90 MAX-SMTI instances with lists of length 3.

Preprocessing method					M3			M4		
Name	Preferences removed	Prop. removed	Runtime (s)	num. comp.	num. opt.	IP Solve (s)	Total (s)	num. opt.	IP Solve (s)	Total (s)
P0	—	—	—	90	90	19	19	90	19	19
P1	2538	3.0%	1	90	90	20	21	90	19	20
P1'	2538	3.0%	1	90	90	19	20	90	19	20
P2	2589	3.0%	1	90	90	19	20	90	19	20
P3	2538	3.0%	1	90	90	19	20	90	19	20
P4	2538	3.0%	1	90	90	19	20	90	19	20
P5	3731	4.4%	50	90	90	19	68	90	18	68
P6	3731	4.4%	1686	70	70	812	2201	70	811	2161

Table 6 shows experimental results when preferences have a length of 5. Again we see that P5 and P6 both take significantly longer to run than the heuristic preprocessing methods, with P6 only completing the preprocessing step in under an hour on 69 of the 90 instances. We also start to see a difference between the IP models, as M3 can solve all 90 instances in anywhere from 397 to 478 seconds, while M4 can only solve 80

of the 90 and takes upwards of 736 seconds. Looking more closely at the times for M3, we see that without any preprocessing the model takes almost 480 seconds to solve, while with any of the heuristics it only takes around 440 seconds to solve. Even P5 is competitive, as while the preprocessing step takes approximately 50 seconds to run (compared to 1 to 2 seconds for the heuristics), this preprocessing reduces the solution time to below 400 seconds on average.

Table 6: Comparison of preprocessing methods on 90 MAX-SMTI instances with lists of length 5.

Preprocessing method					M3			M4		
Name	Preferences removed	Prop. removed	Runtime (s)	num. comp.	num. opt.	IP Solve (s)	Total (s)	num. opt.	IP Solve (s)	Total (s)
P0	—	—	—	90	90	479	479	80	741	741
P1	3345	2.4%	2	90	90	435	437	80	747	748
P1'	3345	2.4%	1	90	90	440	442	80	748	749
P2	3858	2.7%	1	90	90	441	443	80	744	745
P3	3345	2.4%	2	90	90	452	454	80	751	752
P4	3345	2.4%	2	90	90	446	448	80	750	751
P5	7417	5.2%	51	90	90	397	449	80	737	773
P6	7417	5.2%	1595	69	69	960	2450	60	1276	2479

Table 7 shows experimental results when preferences have a length of 10. We now see that model M4 is outperforming model M3, which is consistent with results in the literature [8]. Again, we also see that P5 takes longer than all the heuristics (approximately 60 seconds compared to 3 to 5 seconds), and that P6 takes so long that it can only preprocess 60 of the 90 instances in under an hour. While there are some differences in the number of instances solved by the different preprocessing methods, we note that the given solution times would indicate that the difference is not that significant. This is based, in part, on both P1 and P1' creating identical models yet P1 is able to solve 4 more instances to optimality than P1'. For example, closer examination shows that the 4 instances solved after preprocessing with P1, but not after preprocessing with P1', were all solved in between 3445 and 3529 seconds, which is within 5% of the timeout of 3600 seconds. Allowing for a 5% variation in individual runtimes would then account for this discrepancy.

Table 7: Comparison of preprocessing methods on 90 MAX-SMTI instances with lists of length 10.

Preprocessing method					M3			M4		
Name	Preferences removed	Prop. removed	Runtime (s)	num. comp.	num. opt.	IP Solve (s)	Total (s)	num. opt.	IP Solve (s)	Total (s)
P0	—	—	—	90	29	2961	2961	43	2505	2505
P1	4757	1.7%	4	90	29	2981	2982	46	2493	2494
P1'	4757	1.7%	4	90	29	2978	2980	42	2499	2500
P2	13100	4.6%	2	90	33	2879	2880	43	2481	2482
P3	4820	1.7%	5	90	28	2943	2944	45	2506	2508
P4	4757	1.7%	5	90	29	2984	2985	44	2497	2498
P5	20540	7.2%	59	90	32	2870	2875	41	2490	2506
P6	20540	7.2%	1838	60	28	2875	2975	37	2533	2741

While Tables 5, 6, and 7 show that preprocessing can be useful for model M3, they do not show a similar result for model M4. Yet our results when testing MAX-WT-SMTI-GRP show that preprocessing can be useful when using model M4 (see, e.g., Table 2). Tables 8, 9, and 10 show the results of using preprocessing to solve MAX-WT-SMTI on skewed[3] instances with fewer agents ($n_c = n_p = 250$) but with longer preference lists. Table 8 shows that when preferences are of length 100, model M4 is faster, but we start to also see some differences in running times. The two graph-based preprocessing algorithms both take a few seconds longer to run, but result in models that can be solved faster. We see that by using one of P1, P1', P5, and

[3]Recall that a skewed instance is one in which certain agents are more popular, and thus ranked more highly across the board by those that find them acceptable, than other less popular agents.

P6, each of the 40 instances can be preprocessed and solved in approximately 37 seconds, whereas the other methods all take approximately 39-40 seconds. This would indicate that, for instances with 250 candidates and 250 positions, preprocessing is likely to not be useful if candidates each rank fewer than 100 positions, and only if each candidate ranks 100 or more positions is preprocessing useful.

Table 9 shows times for instances where candidates rank 150 positions, 60% of the total. Again, P5 and P6 take longer to run, but the benefit is not as evident, with both performing comparably to not using preprocessing. Instead two heuristics, P2 and P3, offer better total performance, with P2 reducing total runtime from approximately 88 seconds without preprocessing down to 68 seconds. Table 10 gives results for instances with preferences of length 200. Here again, the heuristics are the best performers, with P3 solving all instances in an average of 102 seconds compared to 137 seconds without any preprocessing, or 129 seconds using existing preprocessing techniques.

Table 8: Results of solving MAX-WT-SMTI on 40 instances with preferences of length 100.

	Preprocessing method				M3			M4		
Name	Preferences removed	Prop. removed	Runtime (s)	num. comp.	num. opt.	IP Solve (s)	Total (s)	num. opt.	IP Solve (s)	Total (s)
P0	—	—	—	40	40	83	83	40	39	39
P1	137	0.5%	< 1	40	40	84	84	40	37	37
P1'	137	0.5%	< 1	40	40	84	84	40	37	37
P2	3304	13.2%	< 1	40	40	72	72	40	40	40
P3	892	3.6%	< 1	40	40	81	81	40	39	39
P4	235	0.9%	< 1	40	40	83	83	40	40	41
P5	6972	27.9%	4	40	40	55	59	40	32	36
P6	6972	27.9%	4	40	40	55	59	40	32	37

Table 9: Results of solving MAX-WT-SMTI on 40 instances with preferences of length 150.

	Preprocessing method				M3			M4		
Name	Preferences removed	Prop. removed	Runtime (s)	num. comp.	num. opt.	IP Solve (s)	Total (s)	num. opt.	IP Solve (s)	Total (s)
P0	—	—	—	40	40	215	215	40	88	88
P1	140	3.7%	< 1	40	40	213	214	40	86	87
P1'	140	3.7%	< 1	40	40	212	213	40	87	87
P2	5089	13.6%	< 1	40	40	195	195	40	67	68
P3	1472	3.9%	< 1	40	40	202	202	40	78	79
P4	430	1.1%	1	40	40	211	212	40	86	88
P5	10857	29.0%	4	40	40	164	168	40	83	89
P6	10857	29.0%	5	40	40	164	169	40	84	91

Table 10: Results of solving MAX-WT-SMTI on 40 instances with preferences of length 200.

	Preprocessing method				M3			M4		
Name	Preferences removed	Prop. removed	Runtime (s)	num. comp.	num. opt.	IP Solve (s)	Total (s)	num. opt.	IP Solve (s)	Total (s)
P0	—	—	—	40	40	824	824	40	137	137
P1	113	0.2%	< 1	40	40	764	765	40	128	129
P1'	113	0.2%	< 1	40	40	766	766	40	128	128
P2	6409	12.8%	< 1	40	40	557	557	40	105	105
P3	1858	3.7%	1	40	40	689	690	40	100	102
P4	493	1.0%	2	40	40	599	601	40	107	109
P5	13930	27.9%	6	40	39	444	449	40	125	132
P6	13930	27.9%	7	40	39	446	453	40	124	132

4.8. Results on instances of HRT-1S

In these instances, only the hospitals were allowed to have ties in their preference lists, and so P1* was able to be tested. The tables detailing the results of this particular experiment are in Appendix B.1. Here we only report the following summary of the results from the experiments on instances of HRT-1S:

- P1*, P2, P3, and P4 all ran in under a second. Of these, P1* removed the most preference list entries. P5 took 3 seconds on average, while P6 took well over 2000 seconds on average. Despite taking longer than P1*, P5 removed $< 0.1\%$ more preference list entries than P1*, and P6 removed $< 0.1\%$ more preference list entries than P5.
- P5 and P6 both removed slightly more preference list entries than P1* (at an average of 12208 for P5/P6, and 12205 for P1*), demonstrating that P1* does not remove all possible preference list entries even on randomly generated instances.
- The difference in average IP solution time after any of P0, P1*, P2, P3, P4 or P5, ignoring trivial cases, varies by only a few percent, highlighting the fact that for this particular set of instances, where doctors must give strict preferences with no ties, adding our preprocessing to the internal preprocessing of Gurobi is not useful.

4.9. Results on instances of HRT-2S

Recall that instances of HRT-2S allow both hospitals and doctors to express ties in their preference lists, and therefore, P1* is not applicable. Thus, there were no prior techniques for preprocessing HRT-2S instances in the literature, and so we compare all of our new techniques versus not using any preprocessing. Table 11 shows that P3 can cut running times in half, compared to not using preprocessing. Our graph-based algorithms, P5 and P6, both remove slightly more preferences than P3, but take significantly more time to do so. P1 removes very few preferences, and so does not significantly affect the running time of the IP solver. P2 and P4 do remove more preferences than P1, but fewer than P3. We recall that P2 is Skip-5, whereas P3 is Skip-15, P4 is Skip-50, and P1 can be thought of as Skip-∞. Thus, while we see that Skip-15 is the better choice for these instances, we theorise that the specific value 15 may be a parameter that can be tuned for various types of problems. Tables 12 and 13 continue to show similar results, where P3 removes slightly fewer preferences than either P5 or P6, but as both P5 and P6 run significantly slower, P3 still gives the best overall runtime. We note that for instances with 1500, or 4000 doctors, P5 and P6 result in the fastest IP solution times, while with 3000 doctors, P3 results in the fastest IP solution times. Due to the inherent volatility of IP solution times, we do not think any conclusive comparison between P3, and P5 or P6, can be drawn from this.

When we look at instances of HRT-2S with 8000 doctors, we see results that are consistent with results on the smaller instances, with P3 performing the best and managing to solve 118 of the 120 instances in under an hour. Additionally, we see that P5 and P6 are both failing to preprocess many of these larger instances. Even when P5 or P6 do manage to complete preprocessing in under an hour, a close examination of the runtimes shows that P3 still sometimes results in the fastest IP solution time.

A more detailed break-down of experimental results by both size and length of preferences is given in Appendix B.2.

4.10. Discussion

The experimental results showed that our new preprocessing techniques can significantly reduce the time required to find optimal stable matchings, with a 44% reduction in running time on real-world instances with the new improvements demonstrated in this paper. However, we also observed that the best preprocessing technique varies depending on the instance or set of instances being solved.

In instances where preference lists were shorter, the new graph-based algorithms (P5 and P6) were often not worth the extra computational cost, as the solution times were not affected by the addition of preprocessing. However even when instances had longer preference lists, preprocessing was useful in reducing the overall computation time to solve instances. One of the Skip-{5,15,50} preprocessing methods (P2, P3,

Table 11: Results of solving MAX-HRT on 120 HRT-2S instances with 1500 doctors, and 150 hospitals sharing 1500 posts

	Preprocessing method				Times		
Name	Preferences removed	Prop. removed	Runtime (s)	num. comp.	num. opt.	IP Solve (s)	Total (s)
P0	—	—	—	120	120	53	53
P1	831	4.0%	2	120	120	54	56
P2	3862	9.1%	< 1	120	120	43	43
P3	11282	25.7%	< 1	120	120	24	25
P4	4373	11.0%	1	120	120	40	42
P5	12136	27.6%	30	120	120	23	54
P6	12136	27.6%	45	120	120	23	68

Table 12: Results of solving MAX-HRT on 120 HRT-2S instances with 3000 doctors, and 300 hospitals sharing 3000 posts

	Preprocessing method				Times		
Name	Preferences removed	Prop. removed	Runtime (s)	num. comp.	num. opt.	IP Solve (s)	Total (s)
P0	—	—	—	120	120	176	176
P1	1637	3.9%	4	120	120	181	185
P2	7778	9.1%	1	120	120	156	157
P3	22579	25.6%	2	120	120	79	81
P4	8820	11.0%	3	120	120	144	147
P5	24415	27.6%	224	120	120	95	319
P6	24415	27.6%	331	120	120	92	423

or P4) was often one of the better, if not the best choice, for preprocessing. However, we also see that the performance of each of these three does vary, with Skip-15 performing worse than all other preprocessing methods on MAX-WT-SMTI-GRP instances with $\kappa = 2$, but performing better on MAX-HRT-2S instances. We expect that for certain sets of problems, there is some natural number n such that Skip-n is a good choice for preprocessing, but said value of n may require tuning. In comparison, our new graph-based methods (P5 and P6) do not require any tuning, but are not always useful in solving instances generated at random. In comparison, however, they are shown to be useful on the real-world instances, and on instances devised to mimic real-world instances. We hypothesise that there is some structure present within these instances that P5 and P6 are able to take advantage of, but we have been unable to precisely determine said structure.

For our real-world and randomly generated MAX-WT-SMTI-GRP instances, our new graph-based algorithms (P5 and P6) were the best performers. The heuristics also generally improved performance, with some exceptions. The removal of preference list entries can clearly reduce overall running times, however there is not a clear and strong correlation between the number of entries removed, and either the overall running time or the IP solution time.

The results on MAX-SMTI instances where candidates have preference lists of length 3 showed that for these instances, preprocessing was not necessary. Indeed, the time taken to run either method P5 or P6

Table 13: Results of solving MAX-HRT on 120 HRT-2S instances with 4000 doctors, and 400 hospitals sharing 4000 posts

	Preprocessing method				Times		
Name	Preferences removed	Prop. removed	Runtime (s)	num. comp.	num. opt.	IP Solve (s)	Total (s)
P0	—	—	—	120	120	242	242
P1	2167	3.9%	6	120	120	310	316
P2	10242	9.1%	2	120	120	212	214
P3	30116	25.7%	3	120	120	183	186
P4	11662	11.0%	4	120	120	195	199
P5	32558	27.7%	516	120	120	167	683
P6	32558	27.7%	737	120	120	162	900

Table 14: Results of solving MAX-HRT on 120 HRT-2S instances with 8000 doctors, and 800 hospitals sharing 8000 posts

	Preprocessing method				Times		
Name	Preferences removed	Prop. removed	Runtime (s)	num. comp.	num. opt.	IP Solve (s)	Total (s)
P0	—	—	—	120	112	905	905
P1	4292	3.9%	18	120	117	854	873
P2	20644	9.1%	5	120	111	810	815
P3	60295	25.8%	8	120	118	726	734
P4	23341	11.0%	11	120	113	811	822
P5	61587	26.8%	1624	89	89	1138	1832
P6	60868	26.6%	2610	63	63	1779	2679

outstrips the time taken to solve any resulting IP model, with P6, in particular, taking over 40 times longer to complete preprocessing than any IP model took to solve. When preferences are short, then, running preprocessing is not as important. As candidates express more preferences, preprocessing can be useful, as can the introduction of dummy variables in the IP models. There is a narrow band where using preprocessing without dummy variables is important, but if dummy variables are used, candidates must express far longer preferences for preprocessing to become useful. We show that when candidates express 100 or more (of 250 total) positions in their preference lists, using both dummy variables and one of P2 or P3 results in the shortest overall time to find a solution.

When instances of HRT only allowed ties in the hospitals' preferences, preprocessing had minimal impact. However, when ties were present in the preference lists of the doctors, our new preprocessing allowed six additional instances with 8000 doctors and 800 hospitals to be solved. Our preprocessing is shown to consistently improve solution times, with a 53% reduction in average runtime on instances with 1500 doctors, a 55% reduction in average runtime on instances with 3000 doctors, and 24% reduction in average runtime on instances with 4000 doctors.

5. Conclusion

This paper has introduced a number of new preprocessing heuristics and algorithms for SMTI, as well as the first known preprocessing techniques applicable to HRT with ties on both sides. In doing so, we have given theoretical results extending the existing literature on preprocessing instances of both SMTI and HRT. On instances of SMTI-GRP, we experimentally show that our new preprocessing methods are faster by 44% on real-world instances, and continue to show improvement on randomly generated instances. Further experiments show that, as expected, preprocessing has more impact when candidates express longer preference lists. We also show that for instances of HRT with ties only present in the hospitals' preference lists, our new preprocessing techniques do remove more preference list entries than existing methods, but performance is not noticeably affected. However our preprocessing techniques are the first that are applicable when doctors are also allowed to express ties. In such instances, we are able to improve performance by 24–55% depending on the size of the instance, as well as solving 98% of instances with 8000 doctors that mimic the real-world application involving the UK Foundation Programme, compared to 93% solved without preprocessing.

Future work includes finding further real-world applications of HRT that would allow both hospitals and residents to express indifference, and applying these preprocessing techniques to those problems. Preprocessing may also be extended to matching problems with coalitions, the simplest of which is the extension of HRT that allows couples to take part jointly.

Acknowledgements

We would like to thank three anonymous reviewers for their insights and advice that have improved this paper. In particular, we thank them for inspiring questions and comments regarding the completeness

of our preprocessing which drove further investigation in this area. This research was supported by the Engineering and Physical Science Research Council through grant numbers EP/P028306/1 (Pettersson and Manlove) and EP/P029825/1 (Delorme, Garcìa, Gondzio, and Kalcsics).

[1] `https://foundationprogramme.nhs.uk/programmes/2-year-foundation-programme/` (UKFP 2021 Applicants' Presentation). Accessed 12 June 2020.

[2] A. Abdulkadiroğlu, P.A. Pathak, A.E. Roth, and T. Sönmez. Changing the Boston school-choice mechanism. NBER working paper 11965, 2006.

[3] D.J. Abraham, A. Levavi, D.F. Manlove, and G. O'Malley. The stable roommates problem with globally-ranked pairs. *Internet Mathematics*, 5(4):493–515, 2008.

[4] M. Balinski and T. Sönmez. A tale of two mechanisms: student placement. *Journal of Economic Theory*, 84(1):73–94, 1999.

[5] P. Biró and S. Kiselgof. College admissions with stable score-limits. *Central European Journal of Operations Research*, 23(4):727–741, 2015.

[6] P. Biró and F. Klijn. Matching with couples: a multidisciplinary survey. *International Game Theory Review*, 15(2), 2013. article number 1340008.

[7] A. Deligkas, G.B. Mertzios, and P.G. Spirakis. The computational complexity of weighted greedy matching. In *Proceedings of AAAI 2017: the 31st AAAI Conference on Artificial Intelligence*, pages 466–474, 2017.

[8] M. Delorme, S. García, J. Gondzio, J. Kalcsics, D. Manlove, and W. Pettersson. Mathematical models for stable matching problems with ties and incomplete lists. *European Journal of Operational Research*, 277(2):426–441, 2019.

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

[10] D. Gale and M. Sotomayor. Some remarks on the stable matching problem. *Discrete Applied Mathematics*, 11:223–232, 1985.

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

[12] R.W. Irving. Stable marriage and indifference. *Discrete Applied Mathematics*, 48:261–272, 1994.

[13] R.W. Irving. Matching medical students to pairs of hospitals: a new variation on a well-known theme. In *Proceedings of ESA '98: the 6th European Symposium on Algorithms*, volume 1461 of *Lecture Notes in Computer Science*, pages 381–392. Springer, 1998.

[14] R.W. Irving and D.F. Manlove. Finding large stable matchings. *ACM Journal of Experimental Algorithmics*, 14, 2009. Section 1, article 2, 30 pages.

[15] R.W. Irving, D.F. Manlove, and G. O'Malley. Stable marriage with ties and bounded length preference lists. *Journal of Discrete Algorithms*, 7(2):213–219, 2009.

[16] R.W. Irving, D.F. Manlove, and S. Scott. The Hospitals/Residents problem with Ties. In *Proceedings of SWAT '00: the 7th Scandinavian Workshop on Algorithm Theory*, volume 1851 of *Lecture Notes in Computer Science*, pages 259–271. Springer, 2000.

[17] R.W. Irving, D.F. Manlove, and S. Scott. Strong stability in the Hospitals/Residents problem. In *Proceedings of STACS '03: the 20th Annual Symposium on Theoretical Aspects of Computer Science*, volume 2607 of *Lecture Notes in Computer Science*, pages 439–450. Springer, 2003. Full version available as [**?**].

[18] T. Kavitha, K. Mehlhorn, D. Michail, and K.E. Paluch. Strongly stable matchings in time $O(nm)$ and extension to the Hospitals-Residents problem. *ACM Transactions on Algorithms*, 3(2), 2007.

[19] A. Kwanashie. *Efficient Algorithms for Optimal Matching Problems under Preferences*. PhD thesis, University of Glasgow, 2015.

[20] A. Kwanashie and D.F. Manlove. An integer programming approach to the Hospitals / Residents problem with Ties. In *Proceedings of OR 2013: the International Conference on Operations Research*, pages 263–269. Springer, 2014.

[21] E.L. Lawler. *Combinatorial Optimization: Networks and Matroids*. Halt, Rinehart and Winston, 1976.

[22] D.F. Manlove. *Algorithmics of Matching Under Preferences*. World Scientific, 2013.

[23] D.F. Manlove, R.W. Irving, K. Iwama, S. Miyazaki, and Y. Morita. Hard variants of stable marriage. *Theoretical Computer Science*, 276(1-2):261–279, 2002.

[24] D.F. Manlove, G. O'Malley, P. Prosser, and C. Unsworth. A Constraint Programming Approach to the Hospitals / Residents Problem. In *Proceedings of CP-AI-OR '07: the 4th International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization*, volume 4510 of *Lecture Notes in Computer Science*, pages 155–170. Springer, 2007.

[25] A. Podhradský. Stable marriage problem algorithms. Master's thesis, Faculty of Informatics, Masaryk University, 2010. Available from `http://is.muni.cz/th/172646/fi_m` (accessed 25 May 2012).

[26] A. Romero-Medina. Implementation of stable solutions in a restricted matching market. *Review of Economic Design*, 3(2):137–147, 1998.

[27] A.E. Roth. Stability and polarization of interests in job matching. *Econometrica*, 52(1):47–57, 1984.

[28] A.E. Roth and E. Peranson. The redesign of the matching market for American physicians: Some engineering aspects of economic design. *American Economic Review*, 89(4):748–780, 1999.

[29] A.E. Roth and M.A.O. Sotomayor. *Two-Sided Matching: a Study in Game-Theoretic Modeling and Analysis*, volume 18 of *Econometric Society Monographs*. Cambridge University Press, 1990.

[30] J.E. Vande Vate. Linear programming brings marital bliss. *Operations Research Letters*, 8(3):147–153, 1989.

Appendix A. Proofs

We will prove Lemma 4 in this section, but first need an additional definition, the *match profile*. The *match profile* of a position p in a given matching is a measure of how highly p ranks the candidates to which it has been assigned.

Definition 3. *Given an instance I of WFT, a matching M, and a position p, the* match profile *of p in M is a vector $(v_1, v_2, \ldots, v_r)$ where r is the maximum rank of a candidate in p's list and each v_i is a non-negative integer such that $v_i = k$ if and only if $M(p)$ contains exactly k candidates that position p lists in its i-th rank.*

Note that for any match profile $(v_1, v_2, \ldots, v_r)$ for a position p, we have $\sum v_i \leq q_p$ as a position cannot be assigned more candidates than their quota allows. For any position p, the set of possible match profiles for p is therefore finite and can be totally ordered in a lexicographic manner.

Example 11. *Given the following preferences, if $M(p_1) = \{c_1, c_3\}$, then the match profile of p_1 in M is* $(1, 0, 1)$.

$$
\begin{aligned}
p_1 &: c_1\ c_2\ [c_3\ c_4] \\
p_2 &: c_2\ c_3 \\
c_1 &: p_1 \\
c_2 &: p_1\, p_2 \\
c_3 &: p_1\, p_2 \\
c_4 &: p_2\, p_1
\end{aligned}
$$

Next we give an algorithm to *resolve* a blocking pair in a matching where all current blocking pairs involve a common and undersubscribed candidate c_0. This algorithm adds a blocking pair to a given matching, removing a different pair if a capacity is exceeded. This may create new blocking pairs, but all newly-created blocking pairs must involve some common candidate c that is now undersubscribed (but previously wasn't). By repeating this process where necessary, the algorithm resolves all new blocking pairs, either producing a stable matching or strictly increasing the number of positions assigned to c_0.

Lemma 9. *At any point while running Algorithm 3 with inputs I, M, and c_0 that satisfy the requirements of Algorithm 3, no position has fewer candidates assigned in M' than in M, and when the algorithm terminates at least one position has a better match profile in M' than in M.*

Proof. If line 9 is reached in the first iteration, the algorithm terminates after inserting the pair (c, p) and so p has a better match profile and no other position has had any change to its assignments. We can therefore assume without loss of generality that the algorithm reaches line 11. The algorithm then selects c' from $M'(p)$ as one of p's worst assignees in M'. As p is full, and (c, p) is a blocking pair of M', p must strictly prefer c to c'. Therefore, line 12 strictly improves the match profile of p in M'. No other line removes a pair from M', so the match profile of p can never get worse, thus in each iteration the match profile of some position p is strictly improved. □

The following corollary is a consequence of Lemma 9 and the fact that the set of possible match profiles for any position is finite and can be totally ordered.

Corollary 1. *Algorithm 3 terminates.*

Lemma 10. *Running Algorithm 3 with inputs I, M, and c_0 that satisfy the requirements of Algorithm 3 produces a matching M' such that any blocking pair of M' involves c_0.*

Proof. Assume towards a contradiction that M' has a blocking pair (c', p') with $c' \neq c_0$. As all blocking pairs of M involve c_0, it must be the case that (c', p') becomes a blocking pair at some point in the construction of

Algorithm 3 Resolve a blocking pair

1: **Input: An instance I of WFT, a matching M that is not stable, and a candidate c_0 that is undersubscribed in M, where each blocking pair of M involves c_0**
2: **Output: A matching M' such that either (i) M' is stable, or (ii) $|M'(c_0)| > |M(c_0)|$ and each blocking pair of M' involves c_0**
3: Let $M' = M$ and $c = c_0$
4: **loop**
5: Let P_B be the set of positions that occur in a blocking pair with c in M'
6: Let (c, p) be a blocking pair of M' that satisfies $p \preceq_c p_B$ for all $p_B \in P_B$
7: **if** p is undersubscribed **then**
8: Add (c, p) to M'
9: **return** M'
10: **end if**
11: Let $c' \in M'(p)$ satisfy $c'' \preceq_p c'$ for all $c'' \in M'(p)$
12: Remove (c', p) from M', and add (c, p) to M'
13: **if** M' contains no blocking pair involving c' **then**
14: **return** M'
15: **end if**
16: Let $c = c'$
17: **end loop**

M'. Consider the earliest iteration of the main loop of Algorithm 3 for which (c', p') appears as a blocking pair of the matching being constructed such that (c', p') remains a blocking pair until the algorithm terminates. Let M'' be the matching as constructed just as (c', p') appears as a blocking pair.

The pair (c', p') must appear as a blocking pair in line 12, when c' has one assignee removed, and becomes undersubscribed. In particular, this means that c' was full before line 12 (else (c', p') would already be blocking) and thus after line 12 c' is underscribed by exactly one position. Then in the next iteration of the loop, $c = c'$, and a position p is chosen on line 6. As (c', p') has just appeared as a blocking pair, and $c = c'$ in this iteration, we know that $p' \not\prec_{c'} p$, and as (c', p') was not blocking before line 12 in the previous iteration, it must also be that there is no $p'' \in M''(c)$ such that $p' \prec_{c'} p''$. If $p = p'$, then the algorithm adds (c', p') to M'', contradicting the fact that (c', p') remains a blocking pair for the rest of the algorithm. However, if (c', p) is added to M'' where $p \neq p'$, then we know that c' is now full, and there is no $p'' \in M''(c') \cup \{p\}$ such that c' strictly prefers p' to p'', but then (c', p') is no longer a blocking pair, also a contradiction. □

Lemma 11. *Running Algorithm 3 with inputs I, M, and c_0 that satisfy the requirements of Algorithm 3 produces a matching M' such that either M' is stable or $|M'(c_0)| > |M(c_0)|$.*

Proof. We see that the first iteration of Algorithm 3's main loop inserts a pair (c, p) to M', where $c = c_0$, at which point $|M'(c_0)| > |M(c_0)|$ holds. We need to show that when the algorithm terminates, either this still holds or M' is stable. From this point, at any point at which a candidate c' has a position unassigned (line 12) the algorithm either terminates as c' is not involved in any blocking pairs, or assigns to c' to some other position (at either line 8 or line 12 in the next iteration where $c = c'$).

If the algorithm terminates after removing a pair (c_0, p) without assigning some other position to c_0, then M' has no blocking pair involving c_0, and in combination with Lemma 10 this means that M' has no blocking pairs and is stable.

Otherwise, for any position unassigned from c_0 some other position is assigned to c_0 (in addition to the first position p assigned at the start of this proof), and so $|M'(c_0)| > |M(c_0)|$. □

Lemma 4. *Let I be an instance of WFT, let c be some candidate such that in any stable matching in I, c is always full, let $\mathcal{P}$ be some set of positions such that if c is assigned to p in some stable matching of I then*

p is in $\mathcal{P}$, and let I' be the instance of WFT created from I by marking as unacceptable to c any position p^+ that satisfies $p \prec_c p^+$ for all $p \in \mathcal{P}$. Then in any matching M that is stable in I', c is full.

Proof. Assume towards a contradiction that there is a matching M such that M is stable in I' but c is not full in M. By the hypothesis of the lemma, M cannot be stable in I, since c is not full in M, so there must be at least one pair that blocks M in I but not in I'. By the construction of I', any blocking pair of M in I must be a pair that was marked unacceptable when creating I' and therefore any such blocking pair of M must involve c.

The previous paragraph shows that Algorithm 3 can be run on M in I with $c = c_0$. By repeatedly calling Algorithm 3 and applying Lemma 11, we can create a matching M' from M such that M' is stable in I. Consider the last pair of the form (c, p) added to M' such that (c, p) remains in M' until M' is stable, and let M'' be the matching as constructed just before (c, p) is added (i.e., as (c, p) is a blocking pair of M''). We consider the following two cases: p is undersubscribed in M'', and p is full in M''. For either case, we show that (c, p) must have been marked unacceptable in the construction of I'.

If p is undersubscribed in M'', then by Lemma 9, p must also be undersubscribed in M. As M is stable in I', and c is also undersubscribed in M, if (c, p) were not marked as unacceptable in the construction of I' then (c, p) would block M in I'. Therefore, it must be that (c, p) was marked unacceptable to construct I'.

Now assume that p is full in M''. Then, as (c, p) blocks M'', there is some $c' \in M''(p)$ such that p prefers c to c'. By Lemma 9, the match profile of p in M'' must be at least as good as the match profile of p in M, so p must consider c' to be at least as good as some candidate in $M(p)$. That is, there must be a $c'' \in M(p)$ such that $c' \preceq_p c''$, and so it follows that $c \prec_p c''$. Again, we see that for (c, p) to not block M in I', it must be that (c, p) was marked unacceptable to create I'.

We now know that (c, p) must have been marked unacceptable in the construction of I', and $(c, p) \in M'$. However, this means that $p' \prec_c p$ for all $p' \in \mathcal{P}$. Thus, as M' is stable in I, this means that in a stable matching of I, c is assigned a position not in $\mathcal{P}$, which is a contradiction. □

Appendix B. Additional result tables

Appendix B.1. HRT-1S

This section consists of Tables B.15, B.16, B.17, B.18, and B.19. These contain results of the experiments on MAX-HRT1S, as discussed in Section 4.8.

Table B.15: Results of solving MAX-HRT on three real-world instances with approximately 750 doctors.

	Preprocessing method				Times		
Name	Preferences removed	Prop. removed	Runtime (s)	num. comp.	num. opt.	IP Solve (s)	Total (s)
P0	—	—	—	3	3	9	9
P1*	2675	58.5%	< 1	3	3	6	6
P2	129	2.8%	< 1	3	3	9	9
P3	1914	41.8%	< 1	3	3	6	6
P4	1513	33.1%	< 1	3	3	8	8
P5	2698	59.0%	1	3	3	6	7
P6	2700	59.1%	98	3	3	6	104

Table B.16: Results of solving MAX-HRT on 150 randomly generated HRT-1S instances with no master list.

	Preprocessing method				Times		
Name	Preferences removed	Prop. removed	Runtime (s)	num. comp.	num. opt.	IP Solve (s)	Total (s)
P0	—	—	—	150	71	2110	2110
P1*	10737	61.2%	< 1	150	70	2087	2087
P2	1012	5.8%	< 1	150	73	2117	2117
P3	6070	34.5%	< 1	150	73	2053	2053
P4	3908	22.2%	< 1	150	71	2079	2080
P5	10739	61.2%	4	150	71	2071	2076
P6	8017	53.9%	2422	67	55	2178	2608

Table B.17: Results of solving MAX-HRT on 150 randomly generated HRT-1S instances with a master list containing 5 distinct scores.

	Preprocessing method				Times		
Name	Preferences removed	Prop. removed	Runtime (s)	num. comp.	num. opt.	IP Solve (s)	Total (s)
P0	—	—	—	150	131	656	656
P1*	10456	59.7%	< 1	150	130	637	637
P2	637	2.9%	< 1	150	130	659	659
P3	7209	41.7%	< 1	150	131	646	646
P4	6015	35.2%	< 1	150	129	653	654
P5	10464	59.8%	7	150	129	667	674
P6	7536	51.3%	2229	77	72	1787	2289

Appendix B.2. HRT-2S

The following tables (Table B.20 through to Table B.35) contain results of the same experiments as those that produced Tables 11, 12, 13, and 14. These tables, however, are split up based not only on size, but also on lengths of preference lists of the doctors.

Table B.18: Results of solving MAX-HRT on 150 randomly generated HRT-1S instances a master list containing 15 distinct scores.

	Preprocessing method				Times		
Name	Preferences removed	Prop. removed	Runtime (s)	num. comp.	num. opt.	IP Solve (s)	Total (s)
P0	—	—	—	150	150	5	5
P1*	13740	78.3%	< 1	150	150	3	3
P2	3153	17.8%	< 1	150	150	5	5
P3	11705	66.6%	< 1	150	150	4	4
P4	11798	67.2%	< 1	150	150	4	4
P5	13741	78.3%	3	150	150	3	6
P6	10353	68.9%	2377	67	67	1992	2377

Table B.19: Results of solving MAX-HRT on 150 randomly generated HRT-1S instances with a master list containing 25 distinct scores.

	Preprocessing method				Times		
Name	Preferences removed	Prop. removed	Runtime (s)	num. comp.	num. opt.	IP Solve (s)	Total (s)
P0	—	—	—	150	150	1	1
P1*	14080	80.2%	< 1	150	150	< 1	< 1
P2	3705	21.1%	< 1	150	150	1	1
P3	12303	70.1%	< 1	150	150	< 1	< 1
P4	12579	71.7%	< 1	150	150	< 1	< 1
P5	14080	80.3%	2	150	150	< 1	3
P6	10779	71.1%	2314	82	82	1632	2314

Table B.20: Results of solving MAX-HRT on 30 HRT-2S instances with 1500 doctors expressing preferences of length 5, and 150 hospitals sharing 1500 posts

	Preprocessing method				Times		
Name	Preferences removed	Prop. removed	Runtime (s)	num. comp.	num. opt.	IP Solve (s)	Total (s)
P0	—	—	—	30	30	3	3
P1	497	6.6%	< 1	30	30	2	3
P2	342	4.6%	< 1	30	30	2	3
P3	897	12.0%	< 1	30	30	2	2
P4	521	6.9%	< 1	30	30	3	3
P5	952	12.7%	< 1	30	30	3	3
P6	952	12.7%	6	30	30	3	8

Table B.21: Results of solving MAX-HRT on 30 HRT-2S instances with 1500 doctors expressing preferences of length 10, and 150 hospitals sharing 1500 posts

	Preprocessing method				Times		
Name	Preferences removed	Prop. removed	Runtime (s)	num. comp.	num. opt.	IP Solve (s)	Total (s)
P0	—	—	—	30	30	8	8
P1	770	5.1%	< 1	30	30	8	8
P2	1127	7.5%	< 1	30	30	8	8
P3	3002	20.0%	< 1	30	30	6	6
P4	1465	9.8%	< 1	30	30	7	7
P5	3195	21.3%	4	30	30	8	12
P6	3195	21.3%	15	30	30	7	22

Table B.22: Results of solving MAX-HRT on 30 HRT-2S instances with 1500 doctors expressing preferences of length 25, and 150 hospitals sharing 1500 posts

	Preprocessing method				Times		
Name	Preferences removed	Prop. removed	Runtime (s)	num. comp.	num. opt.	IP Solve (s)	Total (s)
P0	—	—	—	30	30	42	42
P1	990	2.6%	2	30	30	43	45
P2	4193	11.2%	< 1	30	30	36	37
P3	11998	32.0%	< 1	30	30	24	25
P4	4876	13.0%	1	30	30	34	35
P5	12867	34.3%	34	30	30	24	58
P6	12867	34.3%	52	30	30	22	74

Table B.23: Results of solving MAX-HRT on 30 HRT-2S instances with 1500 doctors expressing preferences of length 50, and 150 hospitals sharing 1500 posts

	Preprocessing method				Times		
Name	Preferences removed	Prop. removed	Runtime (s)	num. comp.	num. opt.	IP Solve (s)	Total (s)
P0	—	—	—	30	30	159	159
P1	1069	1.4%	5	30	30	162	167
P2	9785	13.0%	2	30	30	125	126
P3	29232	39.0%	3	30	30	63	66
P4	10629	14.2%	4	30	30	118	122
P5	31530	42.0%	81	30	30	60	141
P6	31530	42.0%	109	30	30	59	168

Table B.24: Results of solving MAX-HRT on 30 HRT-2S instances with 3000 doctors expressing preferences of length 5, and 300 hospitals sharing 3000 posts

	Preprocessing method				Times		
Name	Preferences removed	Prop. removed	Runtime (s)	num. comp.	num. opt.	IP Solve (s)	Total (s)
P0	—	—	—	30	30	10	10
P1	1004	6.7%	< 1	30	30	9	10
P2	682	4.5%	< 1	30	30	8	8
P3	1787	11.9%	< 1	30	30	9	9
P4	1054	7.0%	< 1	30	30	9	9
P5	1890	12.6%	3	30	30	13	17
P6	1890	12.6%	52	30	30	13	66

Table B.25: Results of solving MAX-HRT on 30 HRT-2S instances with 3000 doctors expressing preferences of length 10, and 300 hospitals sharing 3000 posts

	Preprocessing method				Times		
Name	Preferences removed	Prop. removed	Runtime (s)	num. comp.	num. opt.	IP Solve (s)	Total (s)
P0	—	—	—	30	30	23	23
P1	1523	5.1%	< 1	30	30	23	24
P2	2216	7.4%	< 1	30	30	23	23
P3	5932	19.8%	< 1	30	30	20	21
P4	2912	9.7%	< 1	30	30	21	22
P5	6291	21.0%	29	30	30	31	60
P6	6291	21.0%	123	30	30	32	155

Table B.26: Results of solving MAX-HRT on 30 HRT-2S instances with 3000 doctors expressing preferences of length 25, and 300 hospitals sharing 3000 posts

	Preprocessing method				Times		
Name	Preferences removed	Prop. removed	Runtime (s)	num. comp.	num. opt.	IP Solve (s)	Total (s)
P0	—	—	—	30	30	109	109
P1	1931	2.6%	4	30	30	108	113
P2	8496	11.3%	1	30	30	101	102
P3	23728	31.6%	2	30	30	68	69
P4	9719	13.0%	3	30	30	96	99
P5	25491	34.0%	222	30	30	94	316
P6	25491	34.0%	370	30	30	91	461

Table B.27: Results of solving MAX-HRT on 30 HRT-2S instances with 3000 doctors expressing preferences of length 50, and 300 hospitals sharing 3000 posts

	Preprocessing method				Times		
Name	Preferences removed	Prop. removed	Runtime (s)	num. comp.	num. opt.	IP Solve (s)	Total (s)
P0	—	—	—	30	30	564	564
P1	2090	1.4%	12	30	30	581	594
P2	19720	13.1%	4	30	30	490	494
P3	58868	39.2%	6	30	30	218	224
P4	21593	14.4%	9	30	30	450	459
P5	63986	42.7%	640	30	30	243	884
P6	63986	42.7%	777	30	30	232	1008

Table B.28: Results of solving MAX-HRT on 30 HRT-2S instances with 4000 doctors expressing preferences of length 5, and 400 hospitals sharing 4000 posts

	Preprocessing method				Times		
Name	Preferences removed	Prop. removed	Runtime (s)	num. comp.	num. opt.	IP Solve (s)	Total (s)
P0	—	—	—	30	30	17	17
P1	1355	6.8%	< 1	30	30	15	15
P2	931	4.7%	< 1	30	30	16	16
P3	2413	12.1%	< 1	30	30	14	14
P4	1422	7.1%	< 1	30	30	16	16
P5	2566	12.8%	8	30	30	21	29
P6	2566	12.8%	142	30	30	21	164

Table B.29: Results of solving MAX-HRT on 30 HRT-2S instances with 4000 doctors expressing preferences of length 10, and 400 hospitals sharing 4000 posts

	Preprocessing method				Times		
Name	Preferences removed	Prop. removed	Runtime (s)	num. comp.	num. opt.	IP Solve (s)	Total (s)
P0	—	—	—	30	30	39	39
P1	2018	5.0%	1	30	30	38	39
P2	2993	7.5%	< 1	30	30	38	38
P3	7982	20.0%	< 1	30	30	34	34
P4	3873	9.7%	< 1	30	30	38	39
P5	8493	21.2%	55	30	30	51	106
P6	8493	21.2%	290	30	30	51	342

Table B.30: Results of solving MAX-HRT on 30 HRT-2S instances with 4000 doctors expressing preferences of length 25, and 400 hospitals sharing 4000 posts

	Preprocessing method				Times		
Name	Preferences removed	Prop. removed	Runtime (s)	num. comp.	num. opt.	IP Solve (s)	Total (s)
P0	—	—	—	30	30	169	169
P1	2542	2.5%	6	30	30	180	186
P2	11149	11.1%	2	30	30	132	134
P3	31797	31.8%	2	30	30	111	113
P4	12983	13.0%	4	30	30	143	147
P5	33994	34.0%	433	30	30	135	568
P6	33994	34.0%	798	30	30	129	927

Table B.31: Results of solving MAX-HRT on 30 HRT-2S instances with 4000 doctors expressing preferences of length 50, and 400 hospitals sharing 4000 posts

	Preprocessing method				Times		
Name	Preferences removed	Prop. removed	Runtime (s)	num. comp.	num. opt.	IP Solve (s)	Total (s)
P0	—	—	—	30	30	740	740
P1	2752	1.4%	18	30	30	1007	1025
P2	25895	12.9%	6	30	30	663	669
P3	78273	39.1%	9	30	30	575	584
P4	28371	14.2%	13	30	30	581	594
P5	85178	42.6%	1566	30	30	461	2027
P6	85178	42.6%	1719	30	30	448	2167

Table B.32: Results of solving MAX-HRT on 30 HRT-2S instances with 8000 doctors expressing preferences of length 5, and 800 hospitals sharing 8000 posts

	Preprocessing method				Times		
Name	Preferences removed	Prop. removed	Runtime (s)	num. comp.	num. opt.	IP Solve (s)	Total (s)
P0	—	—	—	30	30	37	37
P1	2689	6.7%	< 1	30	30	39	40
P2	1862	4.7%	< 1	30	30	35	35
P3	4823	12.1%	< 1	30	30	33	33
P4	2818	7.0%	< 1	30	30	34	35
P5	5114	12.8%	54	30	30	52	106
P6	5114	12.8%	1137	30	30	51	1188

Table B.33: Results of solving MAX-HRT on 30 HRT-2S instances with 8000 doctors expressing preferences of length 10, and 800 hospitals sharing 8000 posts

	Preprocessing method				Times		
Name	Preferences removed	Prop. removed	Runtime (s)	num. comp.	num. opt.	IP Solve (s)	Total (s)
P0	—	—	—	30	30	141	141
P1	4013	5.0%	3	30	30	142	145
P2	6010	7.5%	< 1	30	30	141	142
P3	15945	19.9%	1	30	30	122	123
P4	7733	9.7%	2	30	30	143	145
P5	16911	21.1%	281	30	30	206	487
P6	16911	21.1%	2142	30	30	198	2339

Table B.34: Results of solving MAX-HRT on 30 HRT-2S instances with 8000 doctors expressing preferences of length 25, and 800 hospitals sharing 8000 posts

	Preprocessing method				Times		
Name	Preferences removed	Prop. removed	Runtime (s)	num. comp.	num. opt.	IP Solve (s)	Total (s)
P0	—	—	—	30	30	804	804
P1	5063	2.5%	14	30	30	819	833
P2	22385	11.2%	4	30	30	679	683
P3	63629	31.8%	6	30	30	673	679
P4	25726	12.9%	9	30	30	744	753
P5	68243	34.1%	2562	29	29	693	3135
P6	68063	34.0%	3561	3	3	3267	3588

Table B.35: Results of solving MAX-HRT on 30 HRT-2S instances with 8000 doctors expressing preferences of length 50, and 800 hospitals sharing 8000 posts

	Preprocessing method				Times		
Name	Preferences removed	Prop. removed	Runtime (s)	num. comp.	num. opt.	IP Solve (s)	Total (s)
P0	—	—	—	30	22	2636	2638
P1	5401	1.4%	56	30	27	2418	2475
P2	52318	13.1%	14	30	21	2385	2400
P3	156784	39.2%	24	30	28	2076	2100
P4	57087	14.3%	31	30	23	2324	2356
P5	156077	39.0%	3600	0	0	3600	3600
P6	153381	38.3%	3600	0	0	3600	3600