Improving solve times of stable matching problems through preprocessing

William Pettersson *†, Maxence Delorme⋆, Sergio García⋆, Jacek Gondzio⋆, Joerg Kalcsics⋆, and David Manlove†

†*School of Computing Science, University of Glasgow, United Kingdom*
⋆*School of Mathematics, University of Edinburgh, United Kingdom*

March 13, 2020

Abstract

We present new theory, heuristics and algorithms for preprocessing instances of the Stable Marriage with Ties and Incomplete lists (SMTI), the Hospitals/Residents with Ties (HRT), and the Worker-Firms with Ties (WFT) problems. We show that instances of these problems can be preprocessed by removing from the preference lists of some agents entries that correspond to pairs that will not occur in any stable matching. 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 in the literature to describe when preference list entries can be removed from instances of WFT, 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 maximum weight stable matching in real-world instances compared to existing preprocessing techniques, or 80% compared to not using preprocessing. We also show that preprocessing can be useful across a wide range of instance parameters, depending mostly on the lengths of preferences of the agents.

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 [14], 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 [14], 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. The Hospital-Residents problem (HR) [31, 33] 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

*Corresponding author.

applications including the National Resident Matching Program (NRMP) in the US [31], the Canadian Resident Matching Service (CaRMS) in Canada [32], and the Scottish Foundation Allocation Scheme (SFAS) in Scotland [18]. Further applications of HRT include School Choice in Boston [1], Hungary [7, 8], Spain [7, 30], and Turkey [4, 7].

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 [14] can be modified slightly to find stable matchings in polynomial time [16], and all stable matchings in a given instance have the same size [15].

A further complication to stable matching problems is encountered when agents are unable 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* [17]. 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 [17]. 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 [14], and all such stable matchings must therefore have the same size.

While all stable matchings have identical sizes in instances of either SMT or SMI, and can be found in polynomial time, the same is not true if both ties and incomplete preference lists are allowed within the same instance. This problem is called the Stable Marriage with Ties and Incomplete lists problem (SMTI). In instances of SMTI, weakly stable matchings might have different sizes, and the problem of finding a maximum cardinality stable matching (called MAX-SMTI) is NP-hard [27].

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 [11]. 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 [2]. In some applications [11], 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. This problem has also been shown to be NP-hard [10].

The Hospitals/Residents problem with Ties (HRT) is the extension of HR that allows agents to express indifference in their preference lists [21, 22]. It can also be equivalently defined as the extension of SMTI where one set of agents (the hospitals) are 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 HR 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. Many results are known for WF (where preferences are strict) [3, 5, 12, 13], but in the presence of ties results are scarcer. Methods are known for generating Pareto optimal (but not necessarily stable) matchings in WFT instances [9], and finding a maximum cardinality stable matching in WFT is NP-hard, as it is a generalisation of MAX-SMTI, but otherwise no stability results (theoretical or experimental) have been established specifically for the WFT setting.

MAX-SMTI and MAX-HRT have been solved with constraint programming [28] and integer programming (IP) techniques [11, 24, 25]. Linear programming models for SM were originally studied for their properties [16, 34], 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 [24, 25]. 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 [11].

A more complete introduction to stable matching problems can be found in [26].

1.1 Preprocessing stable matching problems

While Gale and Shapley's original algorithm [14] does find a stable matching in polynomial time, an extended version [16] removes from the problem preference list entries that will not affect the outcome of the algorithm. Even though 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 the problem can still be advantageous. Removing preference list entries that are not in any stable matching shortens preference lists, in turn reducing the size of the models and thus solve times [11]. This removal of preferences is done before building the IP model, and so it is called preprocessing.

Two methods of preprocessing HRT instances are known in the literature [19], namely "Hospitals-offer" and "Residents-apply". However both of 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. A different method of preprocessing an instance of SMTI is possible if preferences on one side of an instance of SMTI have length at most two [20]. More generic preprocessing theory has recently been introduced [11], which we expand upon. The same paper also defines a simple preprocessing heuristic (described in Section 4.1.1), and uses this heuristic to gain significant improvements in overall solve times on various families of 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, and to WFT where no existing preprocessing techniques are known. 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 find 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 run-time. 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 [11], 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 which 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 where only the hospitals may have ties in their preference lists. In these instances, we report that preprocessing is rarely relevant, and that existing techniques called "Hospitals-offer" and "Residents-apply" [19] are still suitable. However, our new graph-based algorithms can remove more preference list entries than the combination of "Hospitals-offer" and "Residents-apply", and we give one specific example of this phenomena. We were unable to test our algorithms on instances of HRT with ties on both sides, or on WFT instances, as there is a lack of real-world data for such applications. We hope that by providing preprocessing algorithms for such scenarios, further research in the field will highlight such applications.

1.3 Paper layout

We give a formal definition of the problems we investigate in Section 2 before introducing the existing theory behind preprocessing, our extension to the existing theory that allows the identification of more preprocessing opportunities, and our extension of this new theory to HRT and WFT, in Section 3. An exact algorithm, an IP model, and several heuristics for finding preprocessing opportunities are given in Section 4. Section 5 contains experimental results of these approaches, as well as a discussion of these results, and our conclusion follows in Section 6.

2 Definitions, notation and modelling

In this section we give definitions for the problems investigated, as well as descriptions of IP models that have been used to solve said problems.

2.1 Definitions and notation

We begin by defining WFT, as HRT, SMTI and SMTI-GRP are all 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 $n_c = |C|$. Each position has preferences over some or all of the candidates, and each candidate has preferences over some or all of the positions. We use the terms positions and candidates specifically because our preprocessing is easier to explain when mentally separate the two sets of agents into distinct types. However they are still symmetrically equivalent, so any preprocessing technique applied on one side can always be repeated on the other.

Each agent expresses preferences, which may include ties where an agent is indifferent between two options. 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 or position p also has some positive integral capacity (or quota), denoted q_c or q_p.

A matching in an instance of WFT is a subset of $P \times C$ such that each candidate c (respectively position p) occurs in at most q_c (respectively q_p) pairs. 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 $(p, c) \notin M$ (i.e., p and c are not matched together) is a blocking pair of I if*

- *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 partners), 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 current assigned partners).*

A matching with no blocking pairs is called stable.

HRT is the restriction of WFT where all candidates have unitary capacity ($q_c = 1$ for all candidates c), and SMTI is the restriction of HRT where all positions have unitary capacity ($q_p = 1$ for all positions p).

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. 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.*

$$c_1 : p_1\ [p_2\ p_3]$$
$$c_2 : p_2\ p_3$$
$$p_1 : c_1$$
$$p_2 : c_1\ c_2$$
$$p_3 : c_1\ c_2$$

2.2 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 [11, 24, 25, 29]. In [11], 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 [11] 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 [11]). 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 [11]). 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 test with only one model (model N8 from [11]) which uses both merged stability constraints and dummy variables. 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 [11].

3 Preprocessing theory

This section introduces the existing preprocessing theory for SMTI, our expansion of the theory to uncover more preprocessing, and the extension of these results to both HRT and WFT.

3.1 Existing theory

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 (p, c) appears in no stable matching of I. The removal of these preference list entries can, as demonstrated in Section 5, have a significant effect on the time taken to find either maximum cardinality stable matchings, or maximum weight stable matchings.

Under certain variations of stable matching problems, such as strong or super-strong stability, the existence of a stable matching is not guaranteed (see e.g., [21, 22, 23]). In an instance of a stable matching problem with no stable matchings, 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. As all instances of SMTI and HRT admit a stable matching, however, removing entries from preference

lists that do not correspond to any stable matching simplifies the instance without affecting stability. The following theorem is a restatement of Theorem 1 in [11][1].

Theorem 1 (Theorem 1 in [11])**.** *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., $\mathcal{C} = \{c' \mid \exists\, p \in \mathcal{P} \text{ s.t. } c' \preceq_p c\}$. 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.

3.2 Extending existing results

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, a set of positions $\mathcal{P}'$ such that*

1. *for any $p' \in \mathcal{P}'$ 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),*

and a set of candidates $\mathcal{C}$ such that for each $c' \in \mathcal{C}$, at least one of

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

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

   or

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

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 and instead refer the reader to the 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 $\mathcal{P}$. Note that Criterion 1 in Theorem 2 does not require that $c' \neq c$. In fact, it will always be true that $c \in \mathcal{C}$.

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 4. Given appropriate sets $\mathcal{P}$ and $\mathcal{P}'$, Example 2 demonstrates how $\mathcal{C}$ is determined and used to decide whether preprocessing is possible.

Example 2 (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:*

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

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

[1] The original theorem mistakenly does not specify that $\mathcal{P}$ ($\mathcal{F}$ in [11]) be non-empty; we fix this here.

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.

3.3 Expanding preprocessing to HRT and WFT

We can extend the above preprocessing results to WFT (which includes HRT). We do this by replacing 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 WFT. 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 candidates c' that*

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

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

   or

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 $c^+ \notin \mathcal{C}$.

If p is undersubscribed then p would prefer to be assigned 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 $p' \in \mathcal{P}$, c prefers p to p^+. Thus, in any combination of these cases, (c, p) would form a blocking pair. □

Note that Criterion 1 in Theorem 3, just as in Theorem 2, does mean that $c \in \mathcal{C}$.

Given an instance of WFT I, 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. To prove the opposite case, we first need this 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 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 unstable 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 3 demonstrates that Theorem 3 is more powerful than "Hospitals-offer" and "Residents-apply" [19], the existing preprocessing algorithms for HRT.

Example 3. *This example demonstrates how Theorem 3 can detect preprocessing that is possible in an instance of HRT that "Hospitals-offer" and "Residents-apply" [19] 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.*

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

It is routine to check that neither of "Hospitals-offer" and "Residents-apply" [19] 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 $\mathcal{C} = \{c_2, c_3\}$, and as $q_{p_2} = 1$ and $q_{p_3} = 2$, $||\mathcal{P}||_q = 3 \geq 2 = ||\mathcal{C}||_q$. 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.*

4 Preprocessing problem and approaches

Preprocessing a preference list in a stable matching problem is possible if a set $\mathcal{P}$ can be found that satisfies Theorem 1. We give a number of heuristics, an IP model and a polynomial-time algorithm for finding such sets $\mathcal{P}$. We also show an extension to this algorithm that runs in polynomial-time and finds a pair of sets $\mathcal{P}$ and $\mathcal{P}'$ that satisfy Theorem 2, potentially allowing even more preprocessing. For the preference list L of a given candidate c, a set $\mathcal{P}$ (respectively pair of sets $\mathcal{P}$ and $\mathcal{P}'$) that satisfies Theorem 1 (respectively Theorem 2) and that also minimises the largest rank of any position in $\mathcal{P}$

maximises the number of entries that can be removed from the preference list L. We call such a set (respectively pair of sets) optimal for that preference list.

Given an instance I of a stable matching, we have proven that preprocessing as we describe will produce an instance I' which has the same set of stable matchings as I. However it is not obvious that preprocessing preference lists in this order (that is, preprocessing by finding an optimal set for each preference list in turn) will result in such an instance of smallest size, nor is it obvious that such an instance of smallest size would be easiest to solve.

4.1 Heuristics

Our goal when preprocessing is always to find sets $\mathcal{P}$ that allow preprocessing according to Theorem 1. Our experimental results show that while heuristics are not guaranteed to find optimal preprocessing opportunities, they can still reduce the solve time of the IP models. We show in Section 5 that the trade-off between preprocessing time and solve time means that, for some scenarios, these heuristics are preferable over more complex methods that take longer to compute.

4.1.1 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 [11], and was shown to be implementable in $O((n_c + n_p)n_p)$ time for a given candidate.

In our research into preprocessing, we noted that the implementation used in [11] 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 5.

4.1.2 Skip larger

When using the heuristic of Section 4.1.1, there are some positions p that, when added to $\mathcal{P}$, cause a large number of candidates to be added to $\mathcal{C}$. To avoid such issues, these particular positions p that cause a large increase to $\mathcal{C}$ 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 the size of $\mathcal{C}$ to increase by at least 5, 15, or 50 candidates respectively.

4.1.3 FindBest

As preprocessing requires that $\mathcal{C}$ be smaller than $\mathcal{P}$, it seems reasonable to add to $\mathcal{P}$ a position p that increases the size of $\mathcal{C}$ by the smallest amount possible. Finding such a p is possible by calculating the change in size of $\mathcal{C}$ 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 the size of $\mathcal{C}$ by the smallest amount over all positions in B.

4.2 IP Model

Given an instance of SMTI with n_p positions and n_c candidates, and some particular candidate c_k whose preference list is being processed, let r_i be the rank of position p_i according to candidate c_k, and let $v_{ij} = 1$ if and only if position p_i considers candidate c_j to be at least as good as candidate c_k.

Define binary variables x_i that are set to 1 if and only if position p_i is in $\mathcal{P}$, and binary variables y_j that are set to 1 if and only if candidate c_j is in $\mathcal{C}$. 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:

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

Constraint (2) ensures that the variable z will denote the worst rank of any position in $\mathcal{P}$. Constraint (3) ensures that the solution $\mathcal{P}$ is nonempty, and constraint (4) ensures that the sets $\mathcal{P}$ and $\mathcal{C}$ satisfy the size requirements of Theorem 1. Lastly, constraint (5) ensures that if a given position is in $\mathcal{P}$, then any competition that c has for that position is included in $\mathcal{C}$. This model can be trivially extended to also find sets $\mathcal{P}'$ as required for Theorem 2.

4.3 Graph-based algorithm

This preprocessing algorithm constructs a bipartite graph iteratively by walking down the preference list, and checks the size of a maximum matching of this graph at each iteration. If this maximum matching 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 candidate 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.

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

1. *(p, c) is not acceptable for any $p \in \mathcal{P}'$, and*
2. *any $p \in \mathcal{P}'$ will always be assigned to some candidate in any stable matching of I,*

we construct the graph $G[L_t]$ as follows: For each $p \in \mathcal{P}'$, add a vertex for p to $G[L_t]$, and then for each candidate c' that p finds acceptable, add the edge $\{c', p\}$ 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. 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). Note that we specifically avoid having a vertex for c in this construction.

An example of the construction and use of such graphs will be given in Examples 4, 5, and 6. In the rest of this section, the set U (respectively W) will be used to refer to the set of vertices representing candidates (respectively positions) in the bipartite graph $G[L_t]$.

We begin with the following lemma, which we later use to reason that if a maximum matching is not incident on every position, then there must also be a (possibly different) maximum matching that is not incident on some position that c would find acceptable.

Lemma 6. *Let I be an instance of a stable matching problem, let c be some candidate in I, 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 matching M that satisfies $|M| \leq |W| - 1$, then there exists a matching M' that satisfies $|M'| = |M|$ and the extra criterion that, for some $p \in W \setminus \mathcal{P}'$, p is not incident to M'.*

Proof. Let M be as given in the lemma. If there is some $p \in W \setminus \mathcal{P}'$ such that M is not incident with p, the result holds trivially. So assume that every $p \in W \setminus \mathcal{P}'$ must be incident to an edge in M. Then there must be some $p_1 \in \mathcal{P}'$ that is not incident to an edge in M. We will build an alternating red-blue

path that starts at p_1 and ends with a $p^+ \in W \setminus \mathcal{P}'$ at which point swapping along this path gives a M' that satisfies the theorem. Note however 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 the edges of M in $G[L_t]$ blue. Note that p_1 has a blue-degree of zero, and all vertices in $G[L_t]$ have a blue-degree of at most one. Now take a stable matching in I, and colour the edges of I red. By definition, all $p \in \mathcal{P}'$ have a red-degree of one, and all vertices in $G[L_t]$ have a red-degree of at most one.

We now iteratively build an alternating red-blue path, firstly starting at p_1. As p_1 is in $\mathcal{P}'$, it has a red-degree of one, and as p_1 has a blue-degree of zero this red edge cannot be blue. We follow this red edge to some candidate. This candidate must have a blue-degree of one, we could increase the size of M by augmenting along our alternating red-blue path, which at this point both starts and ends with red edges. As M is maximum, this is not possible, so we can always follow this blue edge (that is simultaneously not red) to a new position.

If we reach a new position in $\mathcal{P}'$ we can repeat this process, so this process can only terminate when we reach a position $p^+ \in W \setminus \mathcal{P}'$. At this point, however, we have an alternating red-blue path of even length, and by removing from M the blue edges in this path and replacing them with the red edges, we obtain a new matching of the same size but which is not incident with p^+, completing the proof. □

Theorem 7. *Let L_t be a truncation of the preference list of some candidate c, let $G[L_t]$ be the graph constructed as above, and let W be the positions represented in $G[L_t]$. If a maximum matching M of $G[L_t]$ satisfies $|M| \leq |W| - 1$, then in any stable matching c will be matched to some position in L_t.*

Proof. Firstly, given the graph $G[L_t]$, let V be the set of vertices in $G[L_t]$, and let $U = V \setminus W$ (i.e., vertices that correspond to candidates). Additionally, let $W' \subseteq W$ be the vertices corresponding to positions that c does not find acceptable.

From M, use Lemma 6 to construct M' such that for some $p^+ \in W \setminus W'$, M' is not incident with p^+. Use this maximum matching M' to create a minimum vertex cover S of size $|M|$ using Kőnig's Theorem (see e.g. page 242 of [6]). Let $I = V \setminus S$ be the corresponding maximum independent set, let $I_U = U \cap I$, and let $I_W = W \cap I$.

Let $\mathcal{P} = \{p \in I_W \mid c \text{ finds } p \text{ acceptable}\}$, let $\mathcal{P}' = \{p \in I_W \mid c \text{ does not find } p\text{acceptable}\}$, and let $\mathcal{C} = U \cup \{c\} \setminus I_U$. That is, $\mathcal{P}$ contains the positions that are in I and that c finds acceptable, $\mathcal{P}'$ contains the positions that are in I and that c does not find acceptable, and $\mathcal{C}$ contains c and all candidates from $G[L_t]$ that are not in I. Note that as p^+ is not incident with M', it cannot be in S and therefore must be in I, meaning that $\mathcal{P} \neq \emptyset$.

We now show that when defined in this manner, $\mathcal{C}$ is precisely the set as defined by the two criteria in Theorem 2. It is easy to check that each element in $\mathcal{C}$ must satisfy at least one of the criteria from Theorem 2. Let $c' \in \mathcal{C}$, and first suppose that $c' = c$. As $\mathcal{P} \neq \emptyset$, Criterion 1 from Theorem 2 holds. Next, suppose that $c' \neq c$. As $c' \notin I_u$ and I is maximum, c' must be adjacent to some $p \in I_W = \mathcal{P} \cup \mathcal{P}'$, and so c' satisfies either Criteria 1 or 2 from Theorem 2.

We now show that any element not in $\mathcal{C}$ cannot satisfy either criteria from Theorem 2. Any candidate $c' \notin \mathcal{C}$ is in the independent set, but so are all vertices in $\mathcal{P}$ and $\mathcal{P}'$. Therefore, there is no edge between c' and any $p \in \mathcal{P}$ or any $p' \in \mathcal{P}'$, and therefore, by construction of $G[L_t]$ each $p \in \mathcal{P}$ must find c' either strictly less preferable than c or unacceptable, and no $p' \in \mathcal{P}'$ can find c' acceptable.

We now have $\mathcal{P}$, $\mathcal{P}'$ and $\mathcal{C}$ defined as per Theorem 2. It remains to be shown that $|\mathcal{C}| \leq |\mathcal{P}| + |\mathcal{P}'|$. By our construction $|\mathcal{C}| = |U| + 1 - |I_U|$, and $|\mathcal{P}| + |\mathcal{P}'| = |I_W|$. Additionally, $|M| \leq |W| - 1$ gives us

$$|I_U| + |I_W| = |I| = |V| - |M| \geq |V| - |W| + 1 = |U| + 1.$$

As $|I_U| + |I_W| \geq |U| + 1$, then $|I_W| \geq |U| + 1 - |I_U|$, and so $|\mathcal{P}| + |\mathcal{P}'| \geq |\mathcal{C}|$. By Theorem 2, c will therefore be matched with some position it considers at least as good as any in L_t, and as L_t is a truncation of the preference list of c, such a position must also be in L_t. □

In the construction of $G[L_t]$, we specifically do not create a vertex for c to avoid a situation where the independent set I contains all candidates (including c), but no positions. This would result in both $\mathcal{C}$ and $\mathcal{P}$ being empty, which will not satisfy the conditions of Theorem 2.

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 2 is possible with $\mathcal{P} \subseteq L_t$. The following theorem

states that the reverse is true: that if preprocessing with $\mathcal{P}$ according to Theorem 2 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 and their truncated preference list L_t, if sets $\mathcal{P}$ and $\mathcal{P}'$ exist that satisfy Theorem 2 such that $\mathcal{P} \subseteq L_t$ then the graph $G[L_t]$ has a maximum matching M that satisfies $|M| \leq |W| - 1$, where W is the set of positions represented in $G[L_t]$.*

Proof. Without loss of generality, choose $\mathcal{P}$ and $\mathcal{P}'$ that satisfy Theorem 2 and that maximise the value of $|\mathcal{P}| + |\mathcal{P}'| - |\mathcal{C}|$. Construct the graph $G[L_t]$ as per Definition 2, with vertex set $V = U \cup W$ where U is the set of candidates represented in $G[L_t]$. Let $I_W = \mathcal{P} \cup \mathcal{P}'$, let $I_U = U \setminus \mathcal{C}$, and let $I = I_U \cup I_W$. Note that $c \in \mathcal{C}$, but $c \notin U$, while for any other $c' \in \mathcal{C}$, $c' \in U$ by construction, so $|I_U| = |U| - |\mathcal{C}| + 1$, or $|\mathcal{C}| = |U| + 1 - |I_U|$. From Theorem 2, we know that for any $u \in U \setminus \mathcal{C}$, no position in $\mathcal{P}$ ranks u at the same level or better than c, and no position in $\mathcal{P}'$ finds u acceptable. Therefore, the vertex set I is an independent set.

We claim that I is a maximum independent set. If not, take a maximum independent set $\overline{I}$ with $|\overline{I}| > |I|$, and construct $\overline{\mathcal{P}}$, $\overline{\mathcal{P}'}$, and $\overline{\mathcal{C}}$ as per the proof of Theorem 7. We deduce from that proof that these sets satisfy the criteria of Theorem 2. It follows from the observations in the previous paragraph, and by the proof of Theorem 7, that

$$|I| = |I_U| + |I_W| = |U| - |\mathcal{C}| + 1 + |\mathcal{P}| + |\mathcal{P}'|$$

and that

$$|\overline{I}| = |\overline{I_U}| + |\overline{I_W}| = |U| - |\overline{\mathcal{C}}| + 1 + |\overline{\mathcal{P}}| + |\overline{\mathcal{P}'}|.$$

Since $|I| < |\overline{I}|$, then

$$|U| - |\mathcal{C}| + 1 + |\mathcal{P}| + |\mathcal{P}'| < |U| - |\overline{\mathcal{C}}| + 1 + |\overline{\mathcal{P}}| + |\overline{\mathcal{P}'}|,$$

and so

$$|\mathcal{P}| + |\mathcal{P}'| - |\mathcal{C}| < |\overline{\mathcal{P}}| + |\overline{\mathcal{P}'}| - |\overline{\mathcal{C}}|$$

contradicting our assumption that we chose $\mathcal{P}$ and $\mathcal{P}'$ that maximise the value of $|\mathcal{P}| + |\mathcal{P}'| - |\mathcal{C}|$.

Thus, I is a maximum independent set, so the complement $V \setminus I$ must be a minimum vertex cover, with size $|V| - |I|$. Via Kőnig's Theorem, we must also have a maximum matching M of size $|M| = |V| - |I|$.

From the earlier paragraphs, and the fact that $\mathcal{P}$, $\mathcal{P}'$, and $\mathcal{C}$ satisfy Theorem 2,

$$|U| + 1 - |I_U| = |\mathcal{C}| \leq |\mathcal{P}| + |\mathcal{P}'| = |I_W|,$$

and hence

$$|U| + 1 \leq |I| = |V| - |M|.$$

It follows that $|M| + 1 \leq |W|$ and this completes the proof. □

Thus, to find an optimal set $\mathcal{P}$ that satisfies the criteria of Theorem 7, we need only consider the truncations L_t of L in ascending order of length. If a maximum matching is too large in a given $G[L_t]$ (i.e. $|M| > |W| - 1$), then Theorem 8 tells us that the preference list cannot be preprocessed at this tie level, and longer truncations of L must be considered.

Algorithm 1 describes the process of building the graphs $G[L_t]$ and finding a maximum matching. 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 $\mathcal{P}$ as per Theorem 1; it does not use Theorem 2 or the set $\mathcal{P}'$. The extension to also find $\mathcal{P}'$ is described in Section 4.3.1.

Given an instance of SMTI 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 matching by finding an augmenting path from each of the newly added vertices that corresponds to a position. Each position is only added to G once, so we need to search for augmenting paths $|L|$ times. 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 $O(n_c|L|^2)$ for preprocessing one preference list of length L, and such a preference list has length at most n_p, so preprocessing a candidate takes $O(n_c n_p^2)$ 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 $M = \emptyset$, let $G[L_t]$ be an empty graph
for each rank r in L **do** ▷ A rank is the index of a tied group of positions
    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 M
    end for
    if $|M| \leq |W| - 1$ **then**
        return r
    end if
end for
return -1 ▷ To indicate that no guarantee of matching can be given

Example 4 (Example of use of Algorithm 1)**.** *Consider an instance of SMTI with $n_c = 4$ and $n_p = 3$ (i.e., there are 4 candidates and 3 positions). We will preprocess the preference list for c_1. The relevant preferences are as follows:*

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

Recall that we will be building a graph $G[L_t]$ based on truncations of the preference list of c. At each step, we will look for a maximum matching M, and if $|M| \leq |W| - 1$ we stop and perform the actual preprocessing.

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), and then we add all candidates who can compete with c 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, p_1) will not form a blocking pair. See Figure 1a.

The maximum matching, indicated in dashed blue in Figure 1a, has cardinality 1. As $|W| = 1$, $|M| > |W| - 1 = 0$ and so we cannot preprocess.

The algorithm continues with the next step, adding 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 matching (indicated in dashed blue in Figure 1b) still has cardinality 1, but now $|W| = 2$ so $|M| \leq |W| - 1$ and so we can preprocess). This means we can remove from the preference list of c_1 any positions that c_1 considers worse than p_2, the last position we added. In this case, this means removing p_3 as an option for c_1, and removing c_1 as an option for p_3. Note that we also now know that c_1 will definitely be matched with some position.

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

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

Figure 1: Two steps in preprocessing for Example 4.

Example 5 (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). 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:*

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

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

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 $|M| \leq |W| - 1$ 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.

4.3.1 Extended graph-based algorithm

Algorithm 1 does not attempt to find sets $\mathcal{P}'$ as defined by Theorem 2. To extend Algorithm 1 to support such sets, we need to know some set $\mathcal{P}''$ of positions that are guaranteed to be matched 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 preprocessing were to result in a truncation of zero preferences from the preference list of p, as this still means that p is always assigned to some candidate in any matching. We then use the set $\mathcal{P}''$ 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 $O(n_c n_p^2)$, the same as for Algorithm 1 by itself, but we show in Section 5 that using Algorithm 2 can under some circumstances significantly increase the runtime for preprocessing, and is not a guaranteed overall performance improvement.

Example 6 (Example of Algorithm 2)**.** *Consider an instance of SMTI with $n_c = n_p = 4$ (i.e., there are 4 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 also know that in any stable matching, position p_1 will always be matched to some candidate.

Algorithm 2 Prefill a graph-based on one preference list

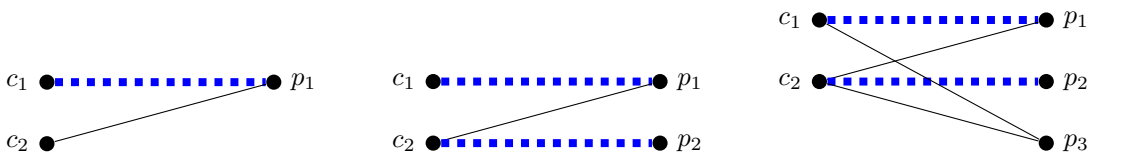
Input: Candidate c, and a set $\mathcal{P}''$ containing positions guaranteed to be filled in any stable matching
Output: A graph G, a set W of positions added to G, and a matching M of G
Let $W = \emptyset$, let $M = \emptyset$, let G be an empty graph
for each position $p \in \mathcal{P}''$ **do**
    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 $G[L_t]$
            Attempt to find an augmenting path from p to increase the size of M
        end for
    end if
end for
return W, M, $G[L_t]$

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 2a.

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 2b with a maximum matching in dashed blue. At this point, $|M| > |W| - 1$, 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 2c, along with a maximum matching in dashed blue. We can see that as $|M| = 2$, and $|W| = 3$, we have $|M| \leq |W| - 1$, 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 $G[L_t]$ after one step of Algorithm 2. The edge marked in dashed blue is a maximum matching.

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

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

Figure 2: Three steps for preprocessing for Example 6.

4.3.2 Extension to many-to-many problems

The heuristics in Section 4.1 preprocess instances of SMTI, but can easily be extended to HRT or WFT. Adapting the graph-based algorithms is slightly more involved, as $\mathcal{P}'$ needs to contain positions that are assigned to capacity, and we involve a capacitated network rather than a graph. We create the network $G'[L_t]$ from a truncated graph $G[L_t]$ as follows. Begin with the graph $G[L_t]$, then:

1. Add a source to the graph, and an edge from the source to each candidate with the capacity of the edge equal to the capacity of the candidate.

2. Add a sink to the graph, and an edge from each position to the sink with the capacity of the edge equal to the capacity of the position.

3. Give each edge between a candidate and a position a capacity of 1.

We can then use this graph and Theorem 9 to decide when to preprocess. Note that while Theorem 3 does also give such a result, it assumes that agents are either full or empty. The following result allows for agents that are neither full nor empty when determining if preprocessing is possible, which makes it more effective.

Theorem 9. *Given an instance of WFT, a candidate c with capacity q_c, and L_t, a truncation of the preference list of c, let $G'[L_t]$ be the network constructed as above. Let W be the positions represented in $G'[L_t]$, and let f be the flow present in a maximum flow through $G'[L_t]$. If $f \leq ||W||_q - q_c$, then in any stable matching c will only be allocated to positions in $\mathcal{P}$.*

The proof of this theorem follows along the same lines as the proof of Theorem 7, and we omit it for brevity.

4.4 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 assigned who had not yet been assigned as such (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.

5 Computational results

We tested our various methods of preprocessing on a wide variety of instances. Each instance was repeatedly preprocessed until no new changes were noticed, as discussed in Section 4.4, before being solved as an IP model.

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 4.1.3 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 4.2 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.

5.1 Problem instances

We tested three different types of problems: SMTI, SMTI-GRP, and HRT. Instances of SMTI with 250 agents are available from `http://dx.doi.org/10.5525/gla.researchdata.904`, and were generated

using the same code as [11]. All other instances were taken from the literature [11], 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 [19]. A more complete investigation of the skew in our instances of SMTI-GRP is given in [11]. We also can model the weight of a pairing (p, c) by summing together the Borda scores of each. 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 include 3 proprietary instances taken from the Scottish Foundation Allocation Scheme (SFAS), as well as a number of randomly-generated instances. 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): $\{0, 5, 15, 25\}$.

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.

As mentioned, the new instances are available from `http://dx.doi.org/10.5525/gla.researchdata.904`, and existing instances are available at `https://researchdata.gla.ac.uk/664/`.

5.2 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.3523174`, 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.

5.3 Presentation of results

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 [11], defined in Section 4.1.1, and P1' is our improved implementation of P1. Both P1 and P1' were only used for SMTI and SMTI-GRP. For HRT, we replaced both of P1 and P1' with P1*, the "Hospitals-offer" and "Residents-apply" methods [19]. 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 and SMTI-GRP only) [11]
P1'	Improved implementation of P1 (for SMTI and SMTI-GRP only)
P1*	"Hospitals offer" and "Residents apply" (for HRT only) [16]
P2	Skip positions if $\|\mathcal{C}\|$ would increase by 5 or more
P3	Skip positions if $\|\mathcal{C}\|$ would increase by 15 or more
P4	Skip positions if $\|\mathcal{C}\|$ 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 four 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 "Run-time" is the average running time of the preprocessing step. 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 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). As a reminder, the rationale for the selection of these models is given in Section 2.2. 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. A combined time-limit of 3600 seconds was set for all experiments. In any cases where a solution was not found (including when preprocessing was not completed in under 3600 seconds), the IP solve time was set to 3600. This was done so that average calculations did not favour scenarios where the preprocessing took longer (and hence less time was given for the IP solver).

5.4 Results on instances of SMTI-GRP

Table 2 shows results from the real-world instances, indicating that (just as in [11]) 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 solve time for combination of model and preprocessing step by a significant margin.

Comparing preprocessing methods P5 with P6, we see that P6 takes almost four times as long to run (approximately 38 seconds compared to approximately 10 seconds), while only removing approximately 4% more preference list entries. However, the removal of these 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 solve 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 as the real-world data (Table 4), we see that no combined method is yet able to solve all 220 instances, but 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 real-world instances.

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

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

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

5.5 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 in under 3600 seconds. For such instances, the IP solve time was set to 3600 seconds as discussed in Section 5.3, 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

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

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

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 or almost 30 minutes on average, respectively. The preprocessing appears to have minimal effect on the IP solve 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 MAX-SMTI instances with lists of length 3

Preprocessing method				M3			M4		
Name	Preferences removed	Run-time	num. comp.	num. opt.	IP Solve	Total	num. opt.	IP Solve	Total
P0	–	–	90	90	19	19	90	19	19
P1	2538	1	90	90	20	21	90	19	20
P1'	2538	1	90	90	19	20	90	19	20
P2	2589	1	90	90	19	20	90	19	20
P3	2538	1	90	90	19	20	90	19	20
P4	2538	1	90	90	19	20	90	19	20
P5	3731	50	90	90	19	68	90	18	68
P6	3731	1686	70	70	812	2201	70	811	2161

Table 6 shows experimental results when preferences have length 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 solve time to below 400 seconds on average.

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

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

Table 7 shows experimental results when preferences have length 10. We now see that model M4 is outperforming model M3, which is consistent with results in the literature [11]. 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 instance 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 solve 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 MAX-SMTI instances with lists of length 10

Preprocessing method				M3			M4		
Name	Preferences removed	Run-time	num. comp.	num. opt.	IP Solve	Total	num. opt.	IP Solve	Total
P0	–	–	90	29	2961	2961	43	2505	2505
P1	4757	4	90	29	2981	2982	46	2493	2494
P1'	4757	4	90	29	2978	2980	42	2499	2500
P2	13100	2	90	33	2879	2880	43	2481	2482
P3	4820	5	90	28	2943	2944	45	2506	2508
P4	4757	5	90	29	2984	2985	44	2497	2498
P5	20540	59	90	32	2870	2875	41	2490	2506
P6	20540	1838	60	28	2875	5174	37	2533	4835

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[2] 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 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.

5.6 Results on instances of HRT

We do not include tables of results for HRT, but instead summarise our results in the following three points:

- P1*, P2, P3, and P4 all ran in under a second, and of these P1* removed the most preference list entries. P5 took on average 3 seconds, 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.

[2]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.

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

Preprocessing method				M3			M4		
Name	Preferences removed	Run-time	num. comp.	num. opt.	IP Solve	Total	num. opt.	IP Solve	Total
P0	–	–	40	40	83	83	40	39	39
P1	137	< 1	40	40	84	84	40	37	37
P1'	137	< 1	40	40	84	84	40	37	37
P2	3304	< 1	40	40	72	72	40	40	40
P3	892	< 1	40	40	81	81	40	39	39
P4	235	< 1	40	40	83	83	40	40	41
P5	6972	4	40	40	55	59	40	32	36
P6	6972	4	40	40	55	59	40	32	37

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

Preprocessing method				M3			M4		
Name	Preferences removed	Run-time	num. comp.	num. opt.	IP Solve	Total	num. opt.	IP Solve	Total
P0	–	–	40	40	215	215	40	88	88
P1	140	< 1	40	40	213	214	40	86	87
P1'	140	< 1	40	40	212	213	40	87	87
P2	5089	< 1	40	40	195	195	40	67	68
P3	1472	< 1	40	40	202	202	40	78	79
P4	430	1	40	40	211	212	40	86	88
P5	10857	4	40	40	164	1681	40	83	89
P6	10857	5	40	40	164	169	40	84	91

- The average IP solve time after any of P0, P1*, P2, P3, P4 or P5 is run is between 672s and 691s, highlighting the fact that for our particular set of instances, where doctors must give strict preferences with no ties, adding our preprocessing to the internal preprocessing of Gurobi is not as useful.

5.7 Discussion

The experimental results showed that our new preprocessing techniques can significantly reduce the time required to find optimal stable matchings. However, we also observed that the best preprocessing technique varies depending on the instance or set of instances being solved. 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 solve 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

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

Preprocessing method				M3			M4		
Name	Preferences removed	Run-time	num. comp.	num. opt.	IP Solve	Total	num. opt.	IP Solve	Total
P0	–	–	40	40	824	824	40	137	137
P1	113	< 1	40	40	764	765	40	128	129
P1'	113	< 1	40	40	766	766	40	128	128
P2	6409	< 1	40	40	557	557	40	105l	105
P3	1858	1	40	40	689	690	40	100	102
P4	493	2	40	40	599	601	40	107	109
P5	13930	6	40	39	444	449	40	125	132
P6	13930	7	40	39	446	453	40	124	132

P6 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 quickest overall time to find a solution.

Preprocessing had minimal impact on our selected suite of test instances for MAX-HRT. This is most likely due to only hospitals being allowed to express indifference. We do however show that our new methods are able to detect more preference list entries that can be removed when compared to existing preprocessing techniques, even though these existing techniques were specialised specifically for HRT where only hospitals can express indifference.

6 Conclusion

We have presented in detail various preprocessing techniques for stable matching problems, which include several heuristics and exact methods. In doing so we have given theoretical results extending the existing literature on preprocessing instances of both SMTI and HRT, and the first known preprocessing results for WFT. On instances of SMTI-GRP, we experimentally show that our new preprocessing methods are faster on smaller instances, and can solve more large instances than existing techniques. 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 on the hospital's side) our new preprocessing techniques do remove more preference list entries than existing methods, but performance is not noticeably affected.

This paper has introduced the first known preprocessing techniques applicable to instances of HRT with ties on both sides, or to WFT. Future work includes finding real-world applications of HRT that would allow both hospitals and residents to express indifference, or real-world applications of WFT, 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.

References

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

[2] 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.

[3] M. Baïou and M. Balinski. Many-to-many matching: stable polyandrous polygamy (or polygamous polyandry). *Discrete Applied Mathematics*, 101:1–12, 2000.

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

[5] V. Bansal, A. Agrawal, and V.S. Malhotra. Polynomial time algorithm for an optimal stable assignment with multiple partners. *Theoretical Computer Science*, 379(3):317–328, 2007.

[6] M. Behzad, G. Chartrand, and L. Lesniak-Foster. *Graphs and Digraphs*. Prinder, Weber and Schmidt International Series, 1979.

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

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

[9] K. Cechlárová, P. Eirinakis, T. Fleiner, D. Magos, D.F. Manlove, I. Mourtos, E. Ocel'áková, and B. Rastegari. Pareto optimal matchings in many-to-many markets with ties. In *Proceedings of SAGT 2015: the 8th International Symposium on Algorithmic Game Theory*, pages 27–39. Springer, 2015.

[10] 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.

[11] 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.

[12] P. Eirinakis, D. Magos, and I. Mourtos. The stable b-matching polytope revisited. *Discrete Applied Mathematics*, 250:186–201, 2018.

[13] P. Eirinakis, D. Magos, I. Mourtos, and P. Miliotis. Finding all stable pairs and solutions to the many-to-many stable matching problem. *INFORMS Journal on Computing*, 24(2):245–259, 2012.

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

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

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

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

[18] 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.

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

[20] 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.

[21] 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.

[22] 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 [**?**].

[23] 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. Article number 15. Preliminary version appeared as [**?**].

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

[25] 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.

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

[27] 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.

[28] 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.

[29] 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).

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

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

[32] 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.

[33] 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.

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

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 7. *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.

Algorithm 3 Resolve a blocking pair

1: **Input: An instance I, an unstable matching M, 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**

Lemma 10. *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 10 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 11. *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 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 12. *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 11 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 12, 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 take 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 10, 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', so it must be that (c, p) were 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 10, 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) were 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 $p' \in \mathcal{P}$, so 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. □