Stable matching with couples – an empirical study

Péter Biró[1,*,†], Robert W. Irving[2,*] and Ildikó Schlotter[3,‡]

[1] *Institute of Economics, Hungarian Academy of Sciences, H-1112, Budaörsi út 45, Budapest, Hungary*
Email: birop@econ.core.hu.

[2] *School of Computing Science, University of Glasgow, Glasgow G12 8QQ, UK.*
Email: rob.irving@glasgow.ac.uk.

[3] *Budapest University of Technology and Economics, H-1521 Budapest, Hungary*
Email: ildi@cs.bme.hu.

Abstract

In practical applications, algorithms for the classical version of the Hospitals Residents problem (the many-one version of the Stable Marriage problem) may have to be extended to accommodate the needs of couples who wish to be allocated to (geographically) compatible places. Such an extension has been in operation in the NRMP matching scheme in the US for a number of years. In this setting, a stable matching need not exist, and it is an NP-complete problem to decide if one does. However, the only previous empirical study in this context (focused on the NRMP algorithm), together with information from NRMP, suggest that, in practice, stable matchings do exist and that an appropriate heuristic can be used to find such a matching.

The study presented here was motivated by the recent decision to accommodate couples in the Scottish Foundation Allocation Scheme (SFAS), the Scottish equivalent of the NRMP. Here, the problem is a special case, since hospital preferences are derived from a 'master list' of resident scores, but we show that the existence problem remains NP-complete in this case. We describe the algorithm used in SFAS, and contrast it with a version of the algorithm that forms the basis of the NRMP approach. We also propose a third simpler algorithm based on satisfying blocking pairs, and an FPT algorithm when the number of couples is viewed as a parameter. We present an empirical study of the performance of a number of variants of these algorithms using a range of data sets. The results indicate that, not surprisingly, increasing the ratio of couples to single applicants typically makes it harder to find a stable matching (and, by inference, less likely that a stable matching exists). However, the likelihood of finding a stable matching is very high for realistic values of this ratio, and especially so for particular variants of the algorithms.

1 Introduction

Background

The Hospitals Residents problem (HR) is a well-known extension of the classical Stable Marriage problem, introduced (under the alternative name of the College Admissions problem) in the seminal paper of Gale and Shapley [3]. The terminology arises from

*Supported by EPSRC grant EP/E011993/1.

†Supported by OTKA grant K69027 and by the Hungarian Academy of Sciences under its Momemtum Programme (LD-004/2010).

‡Supported by OTKA grant K67651.

the important application to matching schemes that assign applicants to positions in the medical domain. The best known of these schemes is the National Resident Matching Program [12] in the U.S., but there are many others, including the Scottish Foundation Allocation Scheme (SFAS) [19]. Our involvement with this latter scheme has been the main motivation for the study reported in this paper. It is well known that an instance of HR can be solved, i.e., a so-called stable matching can be found, in polynomial time, but a number of variants of the basic problem are more challenging. This includes the case where applicants may form couples, who submit joint, rather than individual, preferences. The Hospitals Residents problem with Couples (HRC) has been the subject of various studies, primarily motivated by developments in NRMP.

We consider a variant of HRC motivated by the decision to accommodate couples in SFAS with effect from 2009. This variant differs in some respects from those that have been studied in the literature, and from the version that is currently part of the NRMP – essentially it can be seen as a special case of these. So we first specify the problem, which we designate as *Special* HRC (or SHRC).

Statement of the problem

An instance of SHRC comprises a set of *applicants* (or residents), a set of *programmes* (or hospitals), and a set of *couples*. Each programme p offers a fixed number $c(p)$ of *places*, the *capacity* of the programme. Each couple consists of a pair of distinct applicants, and no applicant can be in more than one couple. An applicant is either *linked* or *single* depending on whether or not he/she is a member of a couple. If applicants a and b form a couple then each of a and b is the *partner* of the other.

Each applicant, single or linked, has a strictly ordered preference list containing a subset of the programmes. Applicant a is said to *prefer* programme p to programme q if p precedes q in a's preference list. A programme that appears on the preference list of an applicant is *acceptable* to that applicant. Each applicant a has a numerical score $s(a)$. Applicant a is *superior* to applicant b, and b is *inferior* to a, if $s(a) > s(b)$. Two applicants with the same score are said to be *of equal rank*. The preference list of a programme is derived directly from the applicant scores, effectively giving a *master* preference list of applicants [5]. This contrasts with the classical versions of HR (and the NRMP context) in which each hospital has a preference list that is independent of the others. In practice, many applicants may have the same score, leading to the presence of *ties* in the master list and in the programmes' preference lists derived from it, but we primarily consider the case where all of the scores are distinct (which can be realised by breaking all of the ties in some arbitrary way).

Each pair of programmes is designated as either *compatible* or not (primarily reflecting their geographical locations). It is assumed that a programme is compatible with itself. Each couple (a, b) has a joint preference list that contains precisely the compatible pairs of programmes (x, y) where x is acceptable to a and y to b. The precise order of the pairs on this joint preference list is not crucial for our purposes, although we do assume that couples' preferences have the so-called *responsive* property, i.e., if a prefers p to q, and both p and q are compatible with r, then (a, b) prefers (p, r) to (q, r) in all cases. In fact, in the SFAS scheme, a couple's joint preference list is constructed in a particular systematic and transparent way from the two individual preference lists[1]. A compatible pair that appears on the joint preference list of couple (a, b) is said to be *acceptable* to that couple. A couple (a, b) *prefers* a programme pair (p, q) to a programme pair (r, s) if (p, q) precedes (r, s) on

[1]Compatible pair (p, q) precedes compatible pair (r, s) on couple (a, b)'s preference list if (i) $rank_a(p) + rank_b(q) < rank_a(r) + rank_b(s)$, or (ii) $rank_a(p) + rank_b(q) = rank_a(r) + rank_b(s)$ and $\max(rank_a(p), rank_b(q)) < \max(rank_a(r), rank_b(s))$, or (iii) $rank_a(p) = rank_b(s)$, $rank_a(r) = rank_b(q)$ and $s(a) > s(b)$. Here, $rank_x(y)$ is the ranking of programme y on the preference list of applicant x.

(a, b)'s joint preference list[2]. Again this represents a restriction of the general version of the problem, in which each couple has complete freedom to specify their own preference list of programme pairs. We comment further on the relationship between SHRC and the general HRC, and the implications of our work for the more general problem, at the end of Section 8.

A *matching* M is a set of applicant-programme pairs satisfying the following three conditions:

- each applicant a appears in at most one pair, and if $\langle a, p\rangle$ is a pair in M then p is acceptable to a;
- if (a, b) is a couple, then either $\langle a, p\rangle$ and $\langle b, q\rangle$ are in M, where (p, q) is acceptable to (a, b), or there is no pair in M containing a or b;
- the number of pairs in M containing the programme p is at most $c(p)$.

In a matching M, an applicant a is *matched* if there is a pair $\langle a, p\rangle$ in M for some programme p, and is otherwise *unmatched*. A programme p is *full* if there are exactly $c(p)$ pairs of the form $\langle a, p\rangle$ in M, and is otherwise *undersubscribed*. If applicant a is matched in M, we denote by $M(a)$ the programme p such that $\langle a, p\rangle$ is in M, i.e., a's *assigned programme* in M. If a is unmatched in M then $M(a)$ is null. Likewise, for a programme p, we denote by $M(p)$ the set of applicants a such that $\langle a, p\rangle$ is in M, i.e., p's *assignees* in M.

Stability

The stability definition for this context is somewhat more complicated, and perhaps more contentious, than in the case where there are no couples. Crucially, in formulating such a definition, most previous authors appear to have overlooked the additional complication that arises because of the possibility that both members of the couple may be assigned to the same programme, or, as in [6], they have sidestepped the issue by forbidding couples from being assigned to the same programme. Only the recent papers of McDermid and Manlove [11] and Marx and Schlotter [10] have addressed this issue explicitly. We provide some detailed justification for our definition of stability, which differs slightly from that given in [11] and [10], but which we believe is appropriate for our context. We first give our definition, and then a detailed rationale for this choice.

A matching M is *stable* if it is not *blocked* by a pair $\langle a, p\rangle$ consisting of a single applicant a and a programme p, or by a pair $\langle (a, b), (p, q)\rangle$ consisting of a couple (a, b) and distinct programmes p and q, or by a pair $\langle (a, b), p\rangle$ consisting of a couple (a, b) and a programme p.

A single applicant a and a programme p *block* M if

(a) a is unmatched, or prefers p to $M(a)$; and

(b) p is undersubscribed, or a is superior to a member of $M(p)$.

A couple (a, b) and a compatible pair of distinct programmes p and q *block* M if

(c) a and b are unmatched, or (a, b) prefers (p, q) to $(M(a), M(b))$; and

(d) p is undersubscribed, or $p = M(a)$, or a is superior to a member of $M(p)$; and

(e) q is undersubscribed, or $q = M(b)$, or b is superior to a member of $M(q)$.

[2]Note that the SFAS scheme does not permit one member of a couple to be allocated to an acceptable programme and the other to be unallocated. However, in the algorithms that we study, this restriction can easily be relaxed by introducing a dummy programme with infinite capacity.

These first two cases are intuitive, and coincide with the corresponding cases in the definitions given by earlier authors.

However the third case is less immediate. We say that a couple (a, b) and a programme p, acceptable to both a and b, *block* M if

(f) a and b are unmatched, or (a, b) prefers (p, p) to $(M(a), M(b))$; and

(g) either

- (i) p has at least two free places in M; or
- (ii) p has one free place in M, and $p \in \{M(a), M(b)\}$ or *both* a and b are superior to a member of $M(p)$; or
- (iii) p is full in M and
  1. $p \in \{M(a), M(b)\}$ and both a and b are superior to a member of $M(p)$; or
  2. both a and b are superior to a member x of $M(p)$, and x is a linked applicant whose partner is also in $M(p)$; or
  3. both a and b are superior to at least two members of $M(p)$.

Rationale

The rationale for our definition is in terms of fairness to the applicants, and ease of justification, based on our practical experience of the SFAS matching scheme.

Once the outcome of the matching process is known, suppose that a single applicant a queries why he was not assigned to a particular preferred programme p. Then we would like the appropriate response to be that programme p filled all of its places with applicants who are at least as good as a, so there is no applicant whom p could reject in order to accommodate a. This notion of stability corresponds exactly to the one that applies in the classical case where there are no couples.

In order to be able to provide an analogous guarantee to couples, a key requirement is to identify the circumstances in which a couple should take precedence over an applicant, and vice versa. We say that a couple $c = (a, b)$ is *superior* to an applicant x, and x is *inferior* to c, if both a and b are superior to x. An applicant x is *superior* to a couple $c = (a, b)$, and c is *inferior* to x, if x is superior to at least one of a and b. On the face of it this definition may seem surprising, since it amounts to awarding a score to a couple on the basis of the weaker member. However, we can justify this in two different ways.

Firstly, consider a programme p with two places and three applicants a, b and x, where a and b form a couple c, x has a score intermediate between those of a and b, and all three of these applicants have p as their first choice programme. If the two places were to be offered to a and b then it would be impossible to make the above response to x were he to query why he was not assigned to p.

Secondly, if a and b were single applicants rather than a couple, then a and x would be assigned to p's two places. If the places were given to a and b, then applicant b would be seen to have gained an advantage by being part of a couple. Single applicants would have some justifiable cause for complaint if, in certain circumstances, the matching scheme were to bestow an advantage on one or more linked applicants – indeed applicants might be tempted to act strategically by forming "artificial" couples if this were the case.

Our precedence rule involving a couple and an applicant explains why in parts (ii) and (iii) of stability condition (g) we require that *both* members of a couple should satisfy a particular condition.

Next we extend the notions of superiority and inferiority to couples, as follows. Suppose for simplicity that a couple is written so that the first member is superior to the second

member or of equal rank. Then couple (a, b) is *superior* to couple (a', b'), and (a', b') *inferior* to (a, b), if (i) b is superior to b', or (ii) b and b' are of equal rank, and a is superior to a'. Again, we are essentially awarding a score to a couple on the basis of the weaker member. However, we argue that this decision is a necessary consequence of the way we defined precedence between a couple and a single applicant. We now explain.

If there are two couples $c = (a, b)$ and $d = (a', b')$ and all of the individuals have unique scores, then, up to symmetry, there are three ways in which the members of the couples may be ranked, namely

$$\begin{array}{llllll} 1. & a & b & a' & b' \\ 2. & a & a' & b & b' \\ 3. & a' & a & b & b' \end{array}$$

In the first two cases, there seems no doubt that we should regard couple c as being superior to couple d, but the third case seems much less clear cut. However, suppose there is a programme p with two places, and that all four of these applicants, and an additional single applicant x, rank p first among their preferences. Suppose further that the rank ordering of the five applicants is:

$$a' \quad a \quad b \quad x \quad b'.$$

Then it follows that p prefers couple c to x, and, as a consequence of our earlier decision, prefers x to couple d. If preferences are to be transitive, which seems a natural and desirable property, then p must prefer c to d. In the given scenario, the only stable possibility is that p's two places are filled by a and b.

We note that this interpretation of precedence between couples is reflected in part (ii) of stability condition (g).

In addition to the above form of response to a query from a dissatisfied single applicant, we can now formulate analogous responses to queries from couples. Suppose that a couple (a, b) question why they were not assigned to a preferred compatible pair of distinct programmes (p, q). Then the appropriate response would be that either p filled all of its places with applicants who are at least as good as a, or q filled all of its places with applicants who are at least as good as b. So there are no two applicants who can be rejected, one by p and one by q, in order to accommodate a and b.

Finally, suppose that a couple $c = (a, b)$ question why they were not both assigned to a programme p. Then the appropriate response depends on whether one of them, say a, or neither of them, is actually assigned to p. In the first case, the response would be that p is full and has no assignee who is inferior to both a and b. In the second case, it would be either that p has one free place but no assignee who is inferior to both a and b, or that p is full but has no assigned couple inferior to c and no two assignees who are both inferior to a and b.

Example 1, essentially the same as that given by Roth [17] and accredited by him to Klaus and Klijn, illustrates that, as in other variants of the problem, an instance of SHRC need not admit a stable matching.

Example 1. There are three applicants, comprising one single applicant a_2 and one couple (a_1, a_3), and two programmes, each with just one place. The applicants are numbered in decreasing order of score (a_1 highest, a_3 lowest), and the preference lists are as shown in Figure 1.

There are three non-empty matchings for this instance, $M_1 = \{\langle a_1, p_1\rangle, \langle a_3, p_2\rangle\}$, $M_2 = \{\langle a_2, p_2\rangle\}$ and $M_3 = \{\langle a_2, p_1\rangle\}$. It may readily be verified that M_1 is blocked by $\langle a_2, p_2\rangle$, M_2 by $\langle a_2, p_1\rangle$, and M_3 by $\langle (a_1, a_3), (p_1, p_2)\rangle$.

a_1 :	p_1	
a_2 :	p_1	p_2
a_3 :	p_2	
(a_1, a_3) :	(p_1, p_2)	

Figure 1: An SHRC instance with no stable matching

An instance of SHRC that admits a stable matching is said to be *solvable*, and is otherwise *unsolvable*.

Example 2 illustrates an additional possibility that does not seem to have been pointed out before, namely that, even in a case where a stable matching does exist, some couple might wish to exchange their allocation, but doing so would violate stability.

Example 2. There are again three applicants, comprising one single applicant a_2 and one couple (a_1, a_3), and two programmes, each with just one place. Again the applicants are numbered in decreasing order of score (a_1 highest, a_3 lowest). The preference lists are as shown in Figure 2.

a_1 :	p_2	p_1
a_2 :	p_1	
a_3 :	p_1	p_2
(a_1, a_3) :	(p_2, p_1)	(p_1, p_2)

Figure 2: An SHRC instance with a stable matching that is not exchange-stable for a couple

The only stable matching for this instance is $M = \{\langle a_1, p_1\rangle, \langle a_3, p_2\rangle\}$ However, both members of the couple would prefer to exchange their positions.

Related work

Roth [14] first observed that a general instance of HRC need not admit a stable matching and Ronn [13] showed that the problem of deciding whether it does is NP-complete, even if all of the programme capacities are equal to one and there are no single applicants. Of course, in the general HRC problem, each programme has its own individual preference list, and the notion of stability is defined in terms of these preferences, rather than in terms of the global 'superiority' concept. As observed above, an instance of SHRC need not admit a stable matching, but it appears that Ronn's original proof of NP-completeness for the general problem cannot be adapted, at least in a straightforward way, to this special case. Aldershof and Carducci [1] show that, in the HRC context, there is no concept analogous to the resident and hospital optimal stable matchings that are known to exist for any HR instance, and also that stable matchings, when they do exist, can have different sizes.

Roth and Peranson [15] describe the couples algorithm implemented by NRMP, and report on empirical studies, using real NRMP data, undertaken to investigate the effect of varying certain aspects of the implementation. A variant of that algorithm, which is actually very similar to Algorithm C of Section 3, is outlined by Klaus et al. [7], who showed, among other things, that, even in cases where a stable matching exists, there may be no possible execution of the algorithm that finds it.

Klaus and Klijn [6] study a restricted version of HRC where the couples' preferences are 'weakly responsive'; this means that they are derived in a logical way from their individual preferences, much as in our context, but crucially there are no incompatible programmes (see the formal definition in [8]). In this context they show that a stable matching is bound to exist, but Kojima et al. [9] observe that such an assumption would be unrealistic

in practice. McDermid and Manlove [11] consider a version of HRC in which couples' preferences are derived in a similarly consistent way from individual preferences, but where pairs of programmes may be incompatible, and show that the problem of deciding whether a stable matching exists is NP-complete in this case, even when applicants' preference lists have length at most three and programme capacities are at most two, and also even in the very special case when couples are required to be matched to the same hospital. On the other hand, they give a linear-time algorithm that determines, in this context, whether there is a matching that is stable in the classical (Gale-Shapley) sense, and in which assigned couples have compatible programmes. Marx and Schlotter [10] study the HRC problem in the context of parameterized complexity, and show, amongst other things, that the existence problem is W[1]-hard when parameterized by the number of couples. Note, however, that in [6] members of a couple are explicitly forbidden from being assigned to the same hospital, while in [11] and [10], the definition of a blocking pair comprising a couple and a hospital differs slightly from ours, as discussed above. Sethuraman et al. [18] discuss a model related to ours, in which each member of a couple submits an individual preference list, and the couple decides on the compatibilty of programmes based on a partition into 'regions'. They show that linear programming can be used to determine in polynomial time whether there is a matching that is stable in the classical sense, i.e., with respect to the preferences of individuals, and in which the members of each couple are assigned to compatible programmes.

Recently, Kojima et al. [9] have shown that, under certain conditions, including a tight bound on the ratio of couples to single applicants, a stable matching exists with high probability in HRC instances, and they present supporting empirical evidence based on several years data from the US market for clinical psychologists.

The contribution of this paper

In this paper, we first establish that the SHRC problem is NP-complete, even under quite severe restrictions. This is not a consequence of the known hardness results for more general versions of the problem. We then describe an algorithm for the problem, similar to that of Klaus et al. [7], and indicate how certain implementation choices lead to a range of variants, including the one (Algorithm C-RAN described in Section 7) that currently forms the basis of the SFAS matching scheme. This algorithm is contrasted with the algorithm described by Roth and Peranson [15], and then a third, conceptually simpler, algorithm, based on satisfying blocking pairs, is described. Again, for each of the alternative algorithms, several possible variants are identified. In Section 6 we show that, in contrast to the W[1]-hardness result of Marx and Schlotter [10] for the general HRC problem, SHRC becomes fixed-parameter tractable when parameterized by the number of couples. The second part of the paper describes an empirical study designed to investigate the likelihood that a stable matching can be found in various circumstances, depending particularly on the ratio of couples to single applicants, and to compare the performance of a number of variants of the three algorithms. The final section summarises the results of this empirical study, and draws a number of conclusions regarding the relative merits of the algorithms and their variants, the likelihood of solving instances of SHRC, and the relevance of these results for more general versions of the problem.

2 SHRC is NP-complete

To justify our empirical study of heuristics for the SHRC problem, we need to establish that this special case, based on a 'master list' of applicants, remains NP-complete.

Theorem 2.1. *The problem of determining whether a stable matching exists for an instance of SHRC is NP-complete, even if there is a strict master list on both sides and each hospital has capacity one.*

Proof The problem is in NP, obviously. We transform from COMPLETE SMTI-2ML, that is the problem of finding a complete stable matching for an instance of the stable marriage problem with incomplete lists, ties and master lists on both sides. This problem is NP-complete ([5], Theorem 3.2.) even under the following restrictions: there are ties in the master list of men only, they are of length 2, each tie appears in only one individual list and it forms the whole of that individual list. Let I be such an instance. We create an instance I' of SHRC under the restrictions listed above, as follows.

First we construct the so-called *proper part* of I'. Let U and W be the set of men and women in I, respectively. Further, let $U_T \subseteq U$ denote the set of men such that each $m_i \in U_T$ has a single tie in his list, i.e., $m_i : (w_{i,1}, w_{i,2})$. The men and the women of I will correspond to the applicants and the programmes in I', respectively. Each programme in I' has unit quota. Initially, let each man with a strict preference list have the same preference list in I' as in I by keeping also the two master lists. Now, for each $m_i \in U_T$ let us create two couples, $(a_{i,1}, a_{i,4})$ and $(a_{i,2}, a_{i,3})$ in I' together with three new programmes, $p_{i,1}$, $p_{i,2}$ and $p_{i,3}$, with the following individual preference lists.

$$\begin{gathered} a_{i,1} : p_{i,1}\ p_{i,3} \\ a_{i,2} : p_{i,1}\ p_{i,3} \\ a_{i,3} : p_{i,2}\ w_{i,2} \\ a_{i,4} : p_{i,2}\ w_{i,1} \end{gathered}$$

We replace m_i with $a_{i,1}$, $a_{i,2}$, $a_{i,3}$ and $a_{i,4}$ in the master list of the applicants (in this order), whilst the tie $(w_{i,1}, w_{i,2})$ is replaced with $p_{i,1}$, $p_{i,2}$, $p_{i,3}$, $w_{i,1}$ and $w_{i,2}$ in the master list of the programmes (in this order). Furthermore, we suppose that $p_{i,1}$ and $p_{i,2}$ are geographically close to each other, whilst $p_{i,3}$, $w_{i,1}$ and $w_{i,2}$ are also geographically close to each other (but far from $p_{i,1}$ and $p_{i,2}$), therefore the following joint preference lists will be constructed:

$$\begin{gathered} (a_{i,1}, a_{i,4}) : (p_{i,1}, p_{i,2})\ (p_{i,3}, w_{i,1}) \\ (a_{i,2}, a_{i,3}) : (p_{i,1}, p_{i,2})\ (p_{i,3}, w_{i,2}) \end{gathered}$$

This completes the construction of the proper part of I'. We shall verify that we have the following one-to-one correspondence between the complete stable matchings of I and the complete stable matchings of the proper part of I'.

- $\langle m_i, w_j \rangle \in M$ for some $m_i \in U \setminus U_T \iff \langle m_i, w_j \rangle \in M'$
- $\langle m_i, w_{i,1} \rangle \in M$ for some $m_i \in U_T \iff$
  $\{\langle a_{i,1}, p_{i,3} \rangle, \langle a_{i,4}, w_{i,1} \rangle, \langle a_{i,2}, p_{i,1} \rangle, \langle a_{i,3}, p_{i,2} \rangle\} \subseteq M'$
- $\langle m_i, w_{i,2} \rangle \in M$ for some $m_i \in U_T \iff$
  $\{\langle a_{i,2}, p_{i,3} \rangle, \langle a_{i,3}, w_{i,2} \rangle, \langle a_{i,1}, p_{i,1} \rangle, \langle a_{i,4}, p_{i,2} \rangle\} \subseteq M'$

To prove this, first let M be a complete stable matching in I and let M' be the corresponding complete matching in I' as described above. Suppose for a contradiction that M' is not stable. If M' is blocked by a single applicant m_i and a programme w_j then this pair, $\langle m_i, w_j \rangle$ would be blocking for M as well. Suppose now that M' is blocked by a couple $(a_{i,1}, a_{i,4})$. This couple cannot be matched to programmes $p_{i,1}$ and $p_{i,2}$, respectively, since this is their first choice, and therefore, according to our construction of M', these

two programmes must be occupied by the other possible couple, $(a_{i,2}, a_{i,3})$. In this case, $(a_{i,1}, a_{i,4})$ is not blocking with $(p_{i,1}, p_{i,2})$. Similarly, we get a contradiction if we suppose that couple $(a_{i,2}, a_{i,3})$ is blocking for M'.

Now, let us suppose that M' is a complete stable matching in I'. The stability and the completeness of M' implies that either $\{\langle a_{i,1}, p_{i,3}\rangle, \langle a_{i,4}, w_{i,1}\rangle, \langle a_{i,2}, p_{i,1}\rangle, \langle a_{i,3}, p_{i,2}\rangle\} \subseteq M'$ or $\{\langle a_{i,2}, p_{i,3}\rangle, \langle a_{i,3}, w_{i,2}\rangle, \langle a_{i,1}, p_{i,1}\rangle, \langle a_{i,4}, p_{i,2}\rangle\} \subseteq M'$ for each index i, where $m_i \in U_T$. Let M be the corresponding complete matching in I as described. Suppose for a contradiction that M is not stable. Note that M cannot be blocked by $\langle m_i, w_{i,1}\rangle$ for some $m_i \in U_T$, since if $\langle m_i, w_{i,1}\rangle \notin M$ then $\langle m_i, w_{i,2}\rangle \in M$ by the construction. We get a similar contradiction if M is blocked by $\langle m_i, w_{i,2}\rangle$. Furthermore if M is blocked by $\langle m_i, w_j\rangle$ for some $m_i \in U \setminus U_T$ then the copy of this pair would block M' too, so the proof of the statement (i.e. the one-to-one correspondence between the complete stable matchings of I and the complete stable matchings of I') is finished.

We refer to those involved in the proper part as *proper programmes* and *proper applicants*. Now we construct the *additional part* of I'. We extend the set of applicants with seven applicants, $\{a_i^* : 0 \leq i \leq 6\}$ by appending them to the end of the master list of the applicants (a_0^* highest, a_6^* lowest), we also add three new programmes $\{p_i^* : 1 \leq i \leq 3\}$ appended to the end of the master list of the programmes in an arbitrary strict order. Let the applicants have the following individual preference lists.

$$\begin{gathered} a_0^* : \text{ [all proper programmes] } p_1^* \\ a_1^* : p_1^* \\ a_2^* : p_3^* \\ a_3^* : p_3^* \\ a_4^* : p_1^* \\ a_5^* : p_2^* \\ a_6^* : p_2^* \end{gathered}$$

Moreover, six of the seven additional applicants form three couples with the following joint lists.

$$\begin{gathered} (a_1^*, a_6^*) : (p_1^*, p_2^*) \\ (a_2^*, a_4^*) : (p_3^*, p_1^*) \\ (a_3^*, a_5^*) : (p_3^*, p_2^*) \end{gathered}$$

We show that I admits a complete stable matching if and only if I' admits a stable matching. Suppose first that M is a complete stable matching in I. Let M' be the corresponding stable matching in the proper part of I' extended with $\{\langle a_0^*, p_1^*\rangle, \langle a_3^*, p_3^*\rangle, \langle a_5^*, p_2^*\rangle\}$. It is straightforward to show that this matching is stable. In the other direction, if M' is a stable matching then first we shall show that the proper programmes are completely filled with proper applicants. This is because a_0^* cannot be allocated to a proper programme, since otherwise it would not be possible to allocate the three additional couples to the three additional programmes in a stable way. But if a_0^* is not allocated to a proper programme then each proper programme must be filled by a proper applicant (since otherwise a_0^* would form a blocking pair with such an unallocated programme). This means that every applicant is matched to a proper programme in the restriction of M' to the proper part of I', therefore M, the corresponding stable matching in I, is complete. □

Note that Theorem 2.1 obviously remains true for a version of the problem intermediate between SHRC and HRC, in which programmes' preferences are derived from a master list, but couples have complete freedom to form their own joint preference lists. Also, the fact that the result holds when all programmes have capacity 1 means that NP-completeness

```
sort the applicants by decreasing score, breaking ties uniformly at random;
for each applicant a in sorted order
   if (a is a single applicant)
       delete all full programmes x from a's list;
       // a cannot be assigned to x because of superior single applicants
       if (a's list contains at least one programme)
           assign a to the first programme on his preference list;
   else // a is a member of a couple c
       delete all entries (x,x) from c's list where x has just one free place;
       // couple c cannot be assigned to x because of superior single applicants
       if (a is the superior member of c)
           delete all entries (x,y) from c's list where x is full;
           // a cannot be assigned to x because of superior single applicants
       else if (a is the inferior member of c)
           delete all entries (y,x) from c's list where x is full;
           // a cannot be assigned to x for the same reason
```

Figure 3: Phase 1 of Algorithm C

does not depend on the precise formulation of the stability criterion for blocking pairs of the form $\langle (a, b), p \rangle$.

Finally, we remark that it is the current practice of SFAS, as in many practical matching schemes, that the preference lists of the applicants have bounded length (currently ten in the case of SFAS). Also, in our empirical study, described in Section 7, we work with randomly generated instances in which each preference list has length six. So the question arises whether the above NP-completeness result holds also for bounded length preference lists. This is indeed the case, since we can extend the proof of Theorem 2.1 for SHRC with the additional restriction that each applicant's preference list has length at most four. The description of the extended proof can be found in the corresponding technical report [2].

3 The SFAS algorithm

The algorithm that forms the basis of SFAS, which we refer to as Algorithm C, consists of two phases.

Phase 1 of Algorithm C
In Phase 1, some initial simplification is undertaken, whereby single applicants can become (provisionally) assigned to the best available programme, and unattainable entries are deleted from preference lists.

Ties consisting of applicants with identical scores are broken at random to produce a strictly ordered list of applicants. We refine the notion of superiority so that applicant a is now regarded as superior to applicant b, and b inferior to a, if a precedes b in this strictly ordered list. In this first phase, the applicants are processed in the order in which they appear in this strictly ordered list. Henceforth, a couple is always represented as an ordered pair (a, b) such that a is superior to b. (Of course, in general, breaking ties in different ways can be expected to lead to different outcomes. The entire algorithm, including the tie-breaking step, can be executed many times, and the 'best' solution returned, according to whatever optimality criterion may be appropriate.) A pseudocode version of Phase 1 of Algorithm C appears in Figure 3.

The outcome of Phase 1 is a reduced set of preference lists and an initial assignment of (a subset of) the single applicants to programmes.

Lemma 3.1. *(i) If programme p is removed from the single applicant a's preference list during Phase 1, then there is no stable matching in which a is assigned to p.*
(ii) If programme pair (p, q) is removed from the couple (a, b)'s preference list during Phase 1, then there is no stable matching in which a is assigned to p and b to q.

Proof (i) Suppose that, at step x of the algorithm, programme p is removed from applicant a's preference list during Phase 1, and that there is a stable matching M in which a is assigned to p. Suppose further that this was the first such removal. Then at step x, p must have been full with applicants superior to a. Hence at least one of these applicants, say b, is not assigned to p in M. But b cannot be assigned in M to a programme he prefers to p, for such a programme would have to have been removed from his list prior to step x, contrary to the assumption that the first such removal was at step x. Hence b and p block M, a contradiction.
(ii) The proof in this case is analogous to the proof of part (i). □

Phase 2 of Algorithm C
Define an *agent* to be either a single applicant or a couple. In Phase 2 of the algorithm, at any given stage, some agents are matched and some are not. Unmatched agents apply to the *next* entry in their preference list, where the next entry moves sequentially along the list, but may be *reset* to an earlier entry in the list in certain circumstances. Unmatched agents that have a next entry available are represented in a data structure, which we refer to as the *waiting list*. At the end of Phase 1, all the couples who have a non-empty preference list are added to the waiting list.

An application is accepted if it constitutes a blocking pair for the current matching, and is otherwise rejected. An accepted application may lead to the rejection of one or more weakest assignees to avoid programmes becoming over-subscribed. If one member of a matched couple is rejected the other member must *withdraw* from his assigned programme. Note that rejection of an agent advances the 'next' entry in the preference list of that agent (taking care to avoid the repetition of such a step when both members of a couple are simultaneously rejected.)

In addition, each programme that is, or has been, full, maintains a set of rejected applicants – its *reserve list*. So if some applicant withdraws from such a programme p, because of their partner's rejection, or because of the possibility of an improved assignment (see below), each applicant in p's reserve list should, in due course, be allowed to re-apply to the programme if this might improve his assignment. This is achieved by withdrawing the applicant from any programme to which he is currently assigned (and likewise his partner, in the case of a linked applicant), conducting a 'reset' operation on the 'next' preference list position (of the applicant or the couple), and adding him (or the couple containing him) to the waiting list (if not already a member of it). Note that the 'reset' operation is *conditional* – it means moving the 'next' position to the one occupied by p, unless this would imply a move forward in the list (perhaps the review of another programme already caused a reset). Reset for a couple means moving the 'next' position to the first entry containing p for the appropriate member (again, only if this represents a move to a position higher in the list).

A further data structure, which we refer to as the *review list*, holds the programmes that have experienced one or more withdrawals and that have a non-empty reserve list. Whenever a programme p is taken from this review list, each applicant on its reserve list whom p would now accept must be examined – this is referred to as *reviewing* the programme.

Consider first a single such applicant a. The pair $\langle a, p\rangle$ blocks the current matching, so a should withdraw from his currently assigned programme (if any), the next position in a's preference list should be (conditionally) reset to the position occupied by p, a should be added to the waiting list (if not already in that list) and should be removed from p's reserve list.

The situation for a linked applicant is a little more subtle. If a is a linked applicant on p's reserve list, say with partner b, then entries of the form (p, q) that precede the current assignment on (a, b)'s preference list must be examined in turn (potentially all such pairs if a and b are unassigned). The first such (p, q) that blocks the current matching with (a, b) leads to actions similar to the previous case – a and b should withdraw from their currently assigned programmes (if any), the next position in (a, b)'s preference list should be (conditionally) reset to the position occupied by (p, q), (a, b) should be added to the waiting list, and a removed from p's reserve list (but see the additional remark below). No further such pairs (p, q) need then be considered. However, if, during the search for such a blocking pair, a pair (p, q) is encountered that does not block the matching, this must be because q would reject b. But b need not be on q's reserve list, so must be added to it in that case (since a subsequent withdrawal from q might otherwise leave $\langle (a, b), (p, q)\rangle$ as a blocking pair). A final subtlety arises if the pair (p, p) is encountered on (a, b)'s preference list but does not block with (a, b). In this case p would accept a but not both a and b. A subsequent withdrawal from p might change this, so we must ensure that one of these applicants remains on p's reserve list, even if a blocking pair of the form $\langle (a, b), (p, q)\rangle$ is subsequently found.

Phase 2 terminates if the waiting list and review list both become empty.

A pseudocode description of a version of Phase 2 of Algorithm C appears in Figures 4 – 8. Recall that we are assuming that when a couple is represented as an ordered pair (a, b), applicant a is superior to applicant b.

```
place each couple with a non-empty preference list on the waiting list;
set the review list to be empty;
while the waiting list W or the review list R is non-empty
    if W is non-empty
        remove agent x from W;
        x applies to the next entry on its preference list;
    else
        remove programme p from R;
        review programme p;
```

Figure 4: Phase 2 of Algorithm C

Theorem 3.1. *If Phase 2 of Algorithm C terminates then the final matching of applicants to programmes is stable.*

Proof We first note some key consequences of the stability definition:

- if a programme p rejects an assignee a in favour of another single applicant, the new assignee is superior to a;
- if p rejects one or two assignees (who do not themselves form a couple) in favour of a couple, both members of the new couple are superior to the rejected applicant(s);

```
// Single applicant a applies to programme p
if a and p block the current matching
    assign a to p;
    if p is oversubscribed
        p rejects its worst assignee;
else
    p rejects a;
```

```
// Couple (a,b) applies to the programme pair (p,q) (p ≠ q)
if (a,b) and (p,q) block the current matching
   assign a to p and b to q;
   if p is oversubscribed
        p rejects its worst assignee;
   if q is oversubscribed
        q rejects its worst assignee;
else
   p rejects a and/or q rejects b;
```

```
// Couple (a,b) applies to the programme pair (p,p)
if (a,b) and p block the current matching
   assign a and b to p;
   if p is oversubscribed
        p rejects its worst assignee;
   if p is still oversubscribed
        p rejects its worst assignee;
else
   p rejects b;   // no need also to reject a
```

Figure 5: The application steps in Phase 2 of Algorithm C

```
// Programme p rejects applicant a
if a is a single applicant
    advance next position in a's preference list;
    if a has preferences remaining
         add a to waiting list;
else     // a is in a couple c
    advance next position in c's preference list;
    if c has preferences remaining
         add c to waiting list (if not already in it);
add a to p's reserve list (if not already in it);
if a is assigned to p
    unassign a from p;
    if a is a linked applicant
         a's partner withdraws from his assigned programme;
```

Figure 6: The rejection step in Phase 2 of Algorithm C

```
// Applicant a withdraws from programme p;
if p has a non-empty reserve list
    add p to the review list;   // if not already in it
unassign a from p;
```

Figure 7: The withdrawal step in Phase 2 of Algorithm C

```
// Review programme p;
for each applicant a in p's reserve list whom p would now accept
    if a is a single applicant
        if ⟨a, p⟩ blocks the current matching
             a withdraws from assigned programme (if any);
             conditionally reset a;      // to position occupied by p
             add a to waiting list;      // if not already in it
             remove a from p's reserve list;
    else if a is the superior member of a couple (a, b)
        for each pair of the form (p, q) in (a, b)'s preference list, in order
             if ⟨(a, b), (p, q)⟩ is in the current matching
                  break;
             else if ⟨(a, b), (p, q)⟩ blocks the current matching for some q
                  a and b withdraw from assigned programmes (if any);
                  conditionally reset (a, b);   // to position occupied by (p, q)
                  add (a, b) to waiting list;   // if not already in it
                  remove a from p's reserve list unless (a, b) prefers (p, p) to (p, q);   (A)
                  break;
             else if q would reject b
                  add b to q's reserve list;   (B)
    else
        deal analogously with the case where a is the inferior member of a couple (b, a)
```

Figure 8: The review step in Phase 2 of Algorithm C

- if p rejects an assigned couple in favour of a new couple, the weaker member of the rejected couple is inferior to both members of the new couple.

Let M be the matching produced by the algorithm on termination. Suppose first that M is blocked by the pair $\langle a, p\rangle$. Then p must have rejected a, possibly more than once. The last time that p rejected a, say at step x in the execution of the algorithm, p must have been full with applicants superior to a, and a must then have become a member of p's reserve list. Denote by b the weakest assignee of p at that point. There could have been no subsequent withdrawals from p, for this would have caused p to be added to the review list, and thereafter, when p was removed from this list, a, as a member of its reserve list, would have had his preference list position reset, and would have to have applied again to p to finish up with a worse assignment than p (or no assignment at all). Hence, since p had no withdrawals after step x, and since the rejection of an assignee after this step cannot give p an assignee inferior to b, it follows that, on termination, p is full with assignees superior to a, a contradiction.

Suppose now that M is blocked by the pair $\langle (a_1, a_2), (p_1, p_2)\rangle$, where p_1 and p_2 are distinct programmes. Then p_1 must have rejected a_1 or p_2 must have rejected a_2, possibly more than once. Suppose, without loss of generality, that the last time this happened, p_1 was full with applicants superior to a_1, so that a_1 became a member of p_1's reserve list at that point. If there were no subsequent withdrawals from p_1 then it is not possible that (a_1, a_2) and (p_1, p_2) block M. On any subsequent withdrawal from p_1, a_1 would be retrieved from p_1's reserve list. If (a_1, a_2) and (p_1, p_2) block the matching at that moment, then couple (a_1, a_2) have their current position reset, and this ensures that they must apply again to (p_1, p_2) (since they end up with a worse assignment than that), which is a contradiction. Otherwise, if p_1 would not accept a_1 at that point, then a_1 will remain on p_1's reserve list awaiting a possible further withdrawal. Finally, if (a_1, a_2) and (p_1, p_2) fail to block the current matching because p_2 is full of applicants superior to a_2, it may happen that a future withdrawal from p_2 causes (a_1, a_2) and (p_1, p_2) to block the matching at that point. However, in this case a_2 will have been placed on the reserve list of p_2 (at the step labeled (A) in Figure 8), and this will ensure that (a_1, a_2) once again apply to (p_1, p_2), a contradiction.

Finally suppose that M is blocked by the pair $\langle (a_1, a_2), p\rangle$. Then again p must have rejected (a_1, a_2), and recalling our assumption that a_1 is superior to a_2, then the last time this happened, p must have had at least $c(p) - 1$ assignees, excluding a_1, who are superior to a_2. Applicant a_2 would have been placed on p's reserve list at that point. If there is no subsequent withdrawal from p, then $\langle (a_1, a_2), p\rangle$ cannot block M. If, when such a withdrawal takes place, the resulting matching is blocked by $\langle (a_1, a_2), p\rangle$, then provided a_2 is on p's reserve list at that point, (a_1, a_2) would have their preference list position reset to ensure a further application to p. So how can this fail to happen? It may be that, following a withdrawal from p, the resulting matching is not blocked by $\langle (a_1, a_2), p\rangle$ but is blocked by $\langle (a_1, a_2), (r, p)\rangle$ where r is some other programme. In this case we must be careful not to remove a_2 from p's reserve list if the resulting reset is to a point lower in the list than (p, p). This is ensured by the step labeled (B) in Figure 8. □

However, Phase 2 of the algorithm may not terminate; certainly if the problem instance admits no stable matching, this will be the case. Furthermore, even if a stable matching does exist, it may not be found by the algorithm, as is illustrated by Example 3.

Example 3. There are eight applicants, comprising three couples (a_1, a_5), (a_2, a_4) and (a_6, a_8) together with two single applicants a_3 and a_7. There are eight programmes, $p_1, \ldots, p_8$, each with just one place. The applicants are numbered in decreasing order of score (a_1 highest, a_8 lowest), and the individual and joint preference lists are as shown in Figure 9.

$$
\begin{array}{rlll}
a_1: & p_1 & p_3 & \\
a_2: & p_4 & p_1 & p_3 \\
a_3: & p_1 & p_5 & \\
a_4: & p_5 & p_2 & p_7 \\
a_5: & p_2 & p_6 & \\
a_6: & p_6 & & \\
a_7: & p_6 & p_8 & \\
a_8: & p_8 & & \\
(a_1, a_5): & (p_1, p_2) & (p_3, p_6) & \\
(a_2, a_4): & (p_4, p_5) & (p_1, p_2) & (p_3, p_7) \\
(a_6, a_8): & (p_6, p_8) & &
\end{array}
$$

Figure 9: An awkward SHRC instance for Algorithm C

There is a unique stable matching

$$M = \{\langle a_1, p_3\rangle, \langle a_2, p_1\rangle, \langle a_3, p_5\rangle, \langle a_4, p_2\rangle\ \langle a_5, p_6\rangle, \langle a_7, p_8\rangle\}$$

for this instance. However, the algorithm fails to converge to this matching; it will reach the matching $M' = \{\langle a_1, p_1\rangle, \langle a_2, p_3\rangle, \langle a_3, p_5\rangle, \langle a_4, p_7\rangle\ \langle a_5, p_2\rangle\}$, and will then cycle for the unsolvable sub-instance induced by $\{a_6, a_7, a_8\}$.

Variants of Algorithm C

A number of variants of Algorithm C arise as a result of possible implementation choices, such as

- the organisation of the waiting list: agents can be removed from this list randomly, or the list can be handled in some more restricted way, say as a stack;
- the relative priority of single applicants and couples: these can be treated equally, or we might choose to prioritise one or the other when choosing an agent from the waiting list;
- the review list can be given priority over the waiting list, so that at each stage a programme from the former list is reviewed, and only when the review list is empty is an agent taken from the waiting list.

4 An algorithm based on the Roth-Peranson approach

The couples algorithm described by Roth and Peranson [15] as the basis for the NRMP approach, when adapted to our context, has much in common with Algorithm C. The main distinction is that agents, i.e., single applicants or couples, are introduced into the market one at a time, and after each such introduction, the resulting sequence of applications, rejections, withdrawals, etc. is allowed to continue until stability is achieved before the next agent is introduced. This is based on the approach used in the algorithm proposed by Roth and Van de Vate [16] for the classical Stable Marriage problem.

Our implementation is, of course, adapted to the context of applicant scores, and incorporates the stability definition given in Section 1. (It is not clear how Roth and Peranson define a blocking pair comprising a couple and one hospital.) We refer to this adaptation of the Roth-Peranson approach as Algorithm RP. In implementing Algorithm

RP, we have available the same choices as those discussed earlier for Algorithm C, and in addition, the order in which agents are introduced to the market can be varied – for example, singles first, or couples first, or a random choice of agent at each stage.

5 An algorithm based on satisfying blocking pairs

For a given instance I of SHRC, a matching M that is not stable is bound to have one or more blocking pairs, as described in Section 1. A given single applicant a, or couple c, may belong to zero or more blocking pairs. An agent that belongs to one or more blocking pairs is called a *blocking agent*, and for any such agent x, we define the *best blocker* for x to be the blocking pair that involves the highest placed entry on x's preference list. The *best blocker set* for M, denoted $\mathcal{B}_M$, is the set of best blockers. It is immediate that M is stable if and only if $\mathcal{B}_M$ is empty.

The general idea of the algorithm of this section, which we refer to as Algorithm BB, is that we maintain throughout the best blocker set relative to the current matching. At each step we choose a blocking pair from this set, change the matching by satisfying this blocking pair, allowing for any required rejections and/or withdrawals, and update the best blocker set accordingly. The algorithm terminates if this set becomes empty. Several variants are possible depending on how a blocking pair is chosen at each step – for example, it may be chosen at random, or blocking pairs involving singles (or couples) could be prioritised, or priorities could be based on applicant scores, or on how often a best blocker for a particular agent has previously been chosen (which we refer to as *usage*).

It is interesting to note that, for the special case where there are no couples, i.e., an instance of the classical HR problem, if the matching is initially empty, the best blocker set initially contains each agent paired with the first entry on its list, and Algorithm BB reduces to an execution of the standard Gale-Shapley algorithm.

In our setting, it seems reasonable to re-use Phase 1 of Algorithm C here, since this is again applicable and will typically remove some entries from preference lists. The initial matching is the one constructed by Phase 1, and the best blocker set is initialised to contain an element for each couple that has a non-empty list at this point, consisting of the couple paired with the first entry on its (reduced) preference list.

Although conceptually simpler than the other algorithms, Algorithm BB does present some interesting implementation challenges. When a blocking pair is satisfied, resulting rejections and withdrawals can have a substantial knock-on effect on the set of best blockers, and these have to be managed carefully to ensure a correct and efficient implementation.

Notice that Example 3 can be used again here to show that, just like Algorithm C, Algorithm RP and Algorithm BB can also fail to find a stable matching even in cases where one exists. To show this, first we shall observe that, considering any of the variants of Algorithms C, RP and BB, if (a_1, a_5) is matched to (p_1, p_2) at any point of the algorithm then (a_1, a_5) must remain matched there subsequently, so the unique stable matching (where this couple is matched to (p_3, p_6)) cannot be reached. Furthermore, in each variant of Algorithms C and RP, no agent becomes matched to his/their second choice before having been rejected by his/their first choice. Suppose for a contradiction that (a_1, a_5) is matched to (p_3, p_6) on termination of one of these algorithms. It must be the case that (a_1, a_5) has already been rejected by (p_1, p_2), and at that point, say at time t_1, p_1 and p_2 must have been occupied by a_2 and a_4, respectively. Let t_2 denote the time when (a_2, a_4) became matched to (p_1, p_2) for the first time. Since (p_1, p_2) is the second choice of (a_2, a_4) then they must have been rejected by their first choice, (p_4, p_5), at an earlier moment, say at t_3. At that point p_5 must have been occupied by a_3. Since p_5 is a_3's second choice, he must have been rejected by p_1 even earlier, say at t_4, when p_1 must have been occupied

by a_1 (and p_2 by a_5), because these positions could not have been occupied by a_2 and a_4 (according to our assumption that they first get matched there at $t_2 > t_4$), a contradiction. By using a similar argument we can also show that none of the variants of Algorithm BB can reach the unique stable matching, since here no agent becomes matched to his/their second choice if he/they form a blocking pair with his/their first choice.

However, a variant of this approach is to base the algorithm on the *complete* set of blocking pairs at each stage. This variant always has the possibility of finding a stable matching when one exists – it might fortuitously choose precisely the pairs of such a matching in some order. Intuition suggests that choosing anything other than the best blocker for an agent may not be a good strategy, intuition that was borne out in practice – our implementation of a variant of Algorithm BB based on a complete set of blocking pairs did not succeed in finding a stable matching (in reasonable time) for any set of test data used in the empirical study described in Section 7.

6 An FPT Algorithm

It is natural to consider the SHRC problem from the point of view of fixed-parameter tractability, with the number of couples as the obvious parameter. In this section, we describe an FPT algorithm for SHRC in the case of a strict master list of applicants. (Recall that the preference lists of the applicants are also strictly ordered). Such an algorithm is easy to devise in the case that the preference lists of the applicants, and hence also of the couples, are bounded in length by a constant. For the general case, we use a subtle argument to show that the preference lists of the couples can be pruned so that they effectively become of bounded length.

Consider a solvable instance of SHRC with a strict master list of applicants, and let k be the number of couples in the instance. Let M be a stable matching and let M' be the matching produced by executing Phase 1 of Algorithm C. For a given applicant a, we refer to applicants with a higher score than a as *a-superior*.

The first lemma emphasises the preference of applicants for M' rather than M.

Lemma 6.1. *(i) No single applicant a prefers $M(a)$ to $M'(a)$.*
(ii) $M(a) = M'(a)$ for every single applicant a who has a higher score than all of the linked applicants.

Proof (i) This follows at once from Lemma 3.1(i).
(ii) Suppose that the claim is invalid, and that applicant a is the highest scoring single applicant for which it is untrue. Then, by Part (i), a prefers $p = M'(a)$ to $M(a)$. In M, as in M', p cannot be full of a-superior applicants, so it is immediate that a and p block M. □

On the other hand, the next lemma shows that each programme does at least as well in M as in M'.

Lemma 6.2. *Every programme p for which $M(p) \neq M'(p)$ prefers $M(p)$ to $M'(p)$ in the sense that, if a is an applicant in $M'(p) \setminus M(p)$, then $|M(p)| = c(p)$ and every applicant in $M(p)$ is a-superior.*

Proof Suppose, for a contradiction, that a is an applicant in $M'(p) \setminus M(p)$ (so that a is necessarily a single applicant) and that either $|M(p)| < c(p)$ or some applicant in $M(p)$ is inferior to a. Then it follows from Lemma 6.1 that a prefers p to his assignment, if any, in M', and therefore a and p block M. □

Consider the couple $c = (a, b)$ where a is the highest scoring linked applicant. Let $L(c)$ be c's preference list after the execution of Phase 1 of Algorithm C, and let $L'(c)$ be

the reduced list obtained by *pruning* $L(c)$ in the following way. Each entry (p, q) of $L(c)$ is considered in turn. If no previous entry was of the form (r, q), with $r \neq q$, and if the number of *accepted* entries is less than $2k$ then (p, q) is accepted, i.e., it remains in the preference list, otherwise it is deleted.

The next lemma underpins the correctness of our FPT algorithm.

Lemma 6.3. *Let M be a stable matching in which couple $c = (a, b)$ is matched to the programme pair (p, q). Then (p, q) appears as an entry on c's pruned list.*

Proof Suppose that (r, q), with $r \neq q$, precedes (p, q) on $L(c)$ (so that a prefers r to p, since couples' preferences are assumed to be responsive). Then, c and (r, q) must block M. This follows from Lemma 6.1(ii) and the fact that (p, q)'s presence on $L(c)$ implies that not all of p's places are filled in M' (and hence not in M either) by a-superior applicants. Note that the condition $r \neq q$ is needed here, for otherwise it may be that q is full in M with b as its weakest assignee, so that a place would be available for a in q only by excluding b.

On the other hand, suppose that there is no such predecessor (r, q) of (p, q) on $L(c)$ but that $L'(c)$ contains a set of $2k$ predecessors of (p, q) from $L(c)$, say $X = (x_1, y_1), \ldots, (x_{2k}, y_{2k})$. By the same argument as in the previous paragraph, it is not possible that any of the programmes x_i has all of its places filled in M' or in M by a-superior applicants. We consider three cases for the pairs (x_i, y_i).

case (a) $x_i \neq y_i$ and (y_i, y_i) does not appear as an earlier entry on $L(c)$. Then y_i has at most $c(y_i) - 1$ b-superior assignees in M' (otherwise (x_i, y_i) would not appear in $L(c)$) and exactly $c(y_i)$ b-superior assignees in M (otherwise (a, b) and (x_i, y_i) would block M).

case (b) $x_i = y_i$ and $y_i \neq y_j$ for all j. Then y_i has at most $c(y_i) - 2$ b-superior assignees in M' (otherwise (x_i, y_i) would not appear in $L(c)$) and at least $c(y_i) - 1$ b-superior assignees in M (otherwise (a, b) and (x_i, y_i) would block M).

case (c) there are two entries (x_i, y_i) and (x_j, y_j) $(i < j)$ in $L(c)$ with $x_i = y_i = y_j$. Then y_i has at most $c(y_i) - 2$ b-superior assignees in M' (otherwise (x_i, y_i) would not appear in $L(c)$) and exactly $c(y_i)$ b-superior applicants in M (otherwise (a, b) and (x_j, y_j) would block M).

Hence, for each of the programmes y_i covered by cases (a) and (b), where y_i appears only once as the second member of a pair in X, there is a b-superior applicant who is assigned to y_i in M but not in M'. For each one covered by case (c), where y_i appears twice as the second member of a pair in X, there are two b-superior applicants who are assigned to y_i in M but not in M'. Therefore, for every pair (x_i, y_i) we can identify a b-superior applicant who is assigned to y_i in M but not in M'. But, by Lemma 6.2, no programme is assigned fewer b-superior applicants in M than in M'. Hence the number of b-superior applicants who are assigned in M exceeds the number who are assigned in M' by at least $2k$. But these additional assigned applicants must all be linked applicants, and there can be at most $2k - 1$ linked b-superior applicants — a contradiction. □

Note that a stronger version of the pruning algorithm would stop accepting entries from $L(c)$ as soon as the number of accepted entries exceeds the number of b-superior linked applicants.

Figure 10 summarises Algorithm F, our FPT algorithm for SHRC in the case of a strict preference list of applicants. It is assumed that Phase 1 of Algorithm C has already been executed to provide the initial preference lists for input to this algorithm. If no matching is output by Algorithm F then no stable matching exists.

Theorem 6.1. *For an instance of SHRC in which the master list of applicants is strict and there are k couples, Algorithm F correctly determines whether a stable matching exists in*

```
procedure solve(int k)    // k is the number of unprocessed couples
    if (k > 0)
         c = (a, b), where a is the highest ranked unprocessed linked applicant;
         prune c's preference list to contain at most 2k entries;
         for each possible assignment of c (including being unassigned)
              update the programmes' capacities;
              // couple c has now been processed
              solve(k-1);
    else
         for each single applicant a, in order of score
              assign a to the first programme on his list, if any,
                        with remaining capacity > 0;
              if the resulting matching M is stable
                   output M and halt;
```

Figure 10: Algorithm F – An FPT algorithm for SHRC with strictly ranked applicants

$O(2^{k+2}(k+1)!f(n))$ time, where f is a polynomial function of constant degree (independent of k) of the input size n.

Proof If the algorithm returns a matching then it is immediate that the matching is stable. Lemma 6.3 establishes that pruning the current couple's preference list to length at most $2k$ cannot exclude a stable matching. So it suffices to show that, for a fixed assignment of the couples, the proposed way in which the single applicants are processed is the only way in which a stable matching can be produced. But this follows at once from the fact that, at any stage, if the next single applicant a is not assigned to the best available programme p then a and p will inevitably form a blocking pair for the resulting matching.

Let $g(n, k)$ be the complexity of the algorithm. Then it follows from the fact that the loop is executed at most $2k + 1$ times that

$$g(n, k) = (2k + 1)g(n, k - 1) + f_1(n)$$

for $k \geq 1$, where f_1 is a polynomial function of n. Because of the obvious polynomial-time complexity of assigning single applicants and checking stability, we also have

$$g(n, 0) = f_2(n)$$

where f_2 is a polynomial function of n. Iteration of the recurrence relation leads to

$$g(n, k) = \prod_{i=0}^{k}(2i + 1)f_2(n) + \sum_{j=1}^{k}\prod_{i=o}^{j-1}(2i + 1)f_1(n),$$

and the claimed complexity follows by straightforward manipulation of this formula. □

7 An empirical study

There is a huge number of combinations of options that could be implemented for the various algorithms, so some selectivity has been essential. The following variants of the three algorithms, already referred to in previous sections, were included in the study, together with a basic version of the FPT algorithm of Section 6.

Algorithm C

- C-RAN: random waiting list;
- C-STA: stack waiting list;
- C-SGL: random waiting list subject to prioritising single applicants;
- C-CPL: random waiting list subject to prioritising couples;
- C-RLP: random waiting list but with review list prioritised.

Algorithm BB

- BB-RAN: random best blocker chosen at each step;
- BB-SCO: best blocker with highest scoring applicant chosen at each step (with score of a couple equal to the lower of the scores of its members);
- BB-USE: best blocker chosen according to usage (see definition in Section 5);
- BB-USS: best blocker chosen according to usage, but with single-blockers prioritised;
- BB-SGL: random best single-blocker chosen if there is one, otherwise random best couple-blocker;
- BB-CPL: random best couple-blocker chosen if there is one, otherwise random best single-blocker.

Algorithm RP

- RP-RAN: random agent introduced at each step;
- RP-SGL: random agent introduced at each step but singles first;
- RP-CPL: random agent introduced at each step but couples first.

Algorithm F

- F: basic implementation.

In this empirical study, we focused entirely on the case of a strictly ordered master list. We generated example problem instances with four different numbers of applicants – 100, 500, 1000 and 2000. For each of these sizes n we varied the number of linked applicants from around 5% of the total up to 100%; this was done by using our random instance generator to create a set of 1000 base instances of size n with no ties, and then randomly pairing together more and more of the applicants until all applicants were in couples. In all cases, we set the number of programmes to be one tenth of the number of applicants, the number of places, distributed randomly among the programmes, to be equal to the number of applicants, and the length of each applicant's preference list to be (somewhat arbitrarily) six. Applicants were given random distinct scores.

All programs were written in Java and were run on a PC with a 2.84 GHz processor, and with 3.5 GB of RAM, running Windows XP. All variants of Algorithms C, BB and RP were allowed a fixed maximum running time on each instance, 1 second, 5 seconds, 10 seconds and 20 seconds for the instances of size 100, 500, 1000 and 2000 respectively. A failure message was output if no stable matching was found within that time. The main output from each program run was the number of instances, out of 1000, for which a stable matching was found within the allocated time. Of course, even if all program variants fail on a particular instance, this need not necessarily be because a stable matching does not

exist. Our conclusions can only be in terms of how feasible it is to find a stable matching, not how likely it is that one exists. In addition for each data set, we aggregated the number of instances that were solved by at least one of the algorithm variants. It was feasible to run Algorithm F only on instances with 100 applicants and 2 or 5 couples. Exhaustive runs in these two cases took around three minutes and eight hours respectively.

Tables 1, 2, 3 and 4 report the numbers of instances of sizes 100, 500, 1000 and 2000, respectively, that were solved, out of a total of 1000 instances in each case, using the selected variants of Algorithms C, BB and RP. Only the variants of the algorithms that were competitive on the smaller instances were run on instances of size 2000, to reduce to manageable proportions the total running time of the programs.

In each column of these tables, the most successful algorithm variant(s), in terms of the number of instances solved, is/are highlighted in bold. The row labeled 'total' in each table reports how many of the 1000 instances were solved by at least one of the algorithm variants, giving a lower bound on the number of these instances that do admit a stable matching. The final row in Table 1 gives, in the few cases where this was feasible, the number of instances solved by Algorithm F, in other words, the actual number of solvable instances.

	Number of couples										
Algorithm	2	5	10	15	20	25	30	35	40	45	50
C-RAN	981	965	909	870	827	801	740	648	604	529	453
C-STA	978	937	831	753	676	640	605	531	508	470	407
C-SGL	981	962	907	862	822	801	753	685	627	545	446
C-CPL	974	927	821	758	712	681	646	586	554	506	451
C-RLP	968	920	825	708	555	424	288	193	136	84	49
BB-RAN	**983**	**966**	**916**	**882**	**851**	**829**	772	701	621	530	430
BB-SCO	968	922	819	722	604	527	444	306	248	172	107
BB-USE	982	962	911	872	839	816	**773**	**705**	**662**	**591**	**507**
BB-USS	968	929	863	805	751	714	686	659	647	582	**507**
BB-SGL	968	931	864	819	779	749	716	699	654	553	429
BB-CPL	981	952	843	687	563	496	410	344	325	329	425
RP-RAN	929	841	704	601	501	411	384	353	274	256	228
RP-SGL	975	925	796	705	613	536	477	394	336	280	211
RP-CPL	917	843	693	586	489	405	358	304	266	222	220
Total	984	967	921	888	861	848	825	793	769	728	672
F	984	969	-	-	-	-	-	-	-	-	-

Table 1: Instances of size 100 (1 second per instance)

In cases where a stable matching is not found, the variants of Algorithm BB can provide potentially useful information not readily available from Algorithms C and RP, namely how near they came to finding a stable solution. It is not unreasonable to regard a matching with the smallest number of best blockers as being one that is closest to stability – since the minimum number of best blockers is the same as the minimum number of agents involved in blocking pairs.

Table 5 shows the average and maximum, taken over all instances that were not solved by any of the algorithms, of the minimum number of best blockers found by any of variants of Algorithm BB. This gives an indication of how close, in general, Algorithm BB came to finding a stable matching in cases where none of the algorithms was able to find one. The clear messages is that, except when a high proportion of the applicants are in couples, we can either find a stable matching, or come very close to one, in the vast majority of the instances that we studied.

One factor that could significantly affect the chances of a stable solution and the performance of the various algorithms is the density of the compatibility matrix used

	Number of couples										
Algorithm	12	25	50	75	100	125	150	175	200	225	250
C-RAN	**976**	**958**	908	862	**811**	729	586	352	163	40	5
C-STA	965	925	807	745	660	588	481	331	191	41	**10**
C-SGL	**976**	957	904	861	801	**752**	677	504	244	61	4
C-CPL	964	908	804	767	709	580	426	253	122	30	5
C-RLP	962	922	805	546	271	92	19	3	1	0	0
BB-RAN	**976**	**958**	**911**	**870**	800	655	412	169	51	14	0
BB-SCO	958	914	793	663	498	342	230	122	65	29	8
BB-USE	**976**	957	909	867	799	696	501	254	81	27	4
BB-USS	963	934	880	825	764	716	**682**	**546**	**281**	**71**	4
BB-SGL	963	934	879	828	773	720	680	529	232	44	0
BB-CPL	974	943	783	482	215	95	25	8	0	1	2
RP-RAN	888	771	579	453	320	188	119	67	35	16	4
RP-SGL	952	897	701	547	395	277	170	83	41	9	3
RP-CPL	872	778	585	424	306	183	115	63	28	11	1
Total	976	958	911	871	820	775	739	642	401	143	29

Table 2: Instances of size 500 (5 seconds per instance)

	Number of couples										
Algorithm	25	50	100	150	200	250	300	350	400	450	500
C-RAN	975	**956**	910	865	**820**	761	503	153	35	1	0
C-STA	961	906	766	672	594	440	238	80	23	1	0
C-SGL	975	956	908	866	818	**786**	**675**	**362**	**101**	6	0
C-CPL	955	913	802	752	707	525	242	69	11	0	0
C-RLP	968	923	789	376	89	13	0	0	0	0	0
BB-RAN	**976**	**956**	**914**	**871**	772	424	100	11	3	0	0
BB-SCO	963	916	785	655	492	324	173	97	32	**7**	**2**
BB-USE	**976**	955	912	868	782	475	140	19	5	0	0
BB-USS	968	934	878	824	775	745	638	296	47	0	0
BB-SGL	968	933	879	822	786	754	654	294	44	0	0
BB-CPL	969	934	745	395	80	7	0	0	0	0	0
RP-RAN	879	770	562	376	251	167	66	27	3	1	0
RP-SGL	948	863	695	503	326	239	103	33	6	1	0
RP-CPL	879	780	559	425	256	140	64	12	2	0	0
Total	976	956	914	871	830	799	714	469	151	12	2

Table 3: Instances of size 1000 (10 seconds per instance)

	Number of couples										
Algorithm	50	100	200	300	400	500	600	700	800	900	1000
C-RAN	**970**	**957**	926	**855**	**810**	744	518	82	1	0	0
C-SGL	**970**	**957**	924	854	799	**745**	**670**	**360**	**25**	0	0
BB-RAN	**970**	**957**	**927**	**855**	592	74	3	0	0	0	0
BB-USS	955	940	894	817	763	707	475	58	1	0	0
BB-SGL	956	939	894	820	760	718	508	67	0	0	0
Total	970	957	927	856	812	751	680	369	26	0	0

Table 4: Instances of size 2000 (20 seconds per instance)

		Percentage of linked applicants										
Instance size		5	10	20	30	40	50	60	70	80	90	100
100	Average	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.03	1.04	1.19
	Maximum	1	1	1	1	1	1	1	2	2	2	4
500	Average	1.00	1.00	1.00	1.04	1.03	1.04	1.07	1.56	3.37	6.83	12.4
	Maximum	1	1	1	2	2	2	4	10	14	23	29
1000	Average	1.00	1.02	1.02	1.04	1.03	1.11	1.26	4.13	11.6	23.1	37.6
	Maximum	1	2	2	2	2	6	6	17	30	43	61
2000	Average	1.00	1.00	1.01	1.04	1,12	1.15	2.64	15.8	37.9	68.2	102
	Maximum	1	1	1	2	2	4	22	45	69	102	144

Table 5: Average and maximum of overall minimum number of best blockers in unsolved instances of sizes 100, 500, 1000 and 2000

to form couples' preference lists. The experiments reported above used a compatibility probablity of 0.75, so to investigate this effect we repeated a subset of the simulations using a lower compatibility probability of 0.3. Table 6 gives the results of running the algorithms on the instances of size 100, but using that lower compatibility probability. Again, the final row of the table summarises the results of running Algorithm F on these instances.

	Number of couples										
Algorithm	2	5	10	15	20	25	30	35	40	45	50
C-RAN	983	946	884	820	796	740	694	645	574	521	431
C-STA	979	917	812	693	654	560	522	518	462	448	376
C-SGL	983	946	877	815	794	741	695	660	589	**532**	431
C-CPL	976	914	800	706	685	638	607	576	519	506	**435**
C-RLP	974	927	838	730	621	502	367	275	180	127	98
BB-RAN	**985**	**950**	**891**	**847**	**804**	**758**	**713**	676	560	478	385
BB-SCO	973	930	841	742	623	544	413	333	227	158	115
BB-USE	983	932	884	838	792	751	709	680	590	522	410
BB-USS	973	932	865	806	753	706	655	646	572	522	410
BB-SGL	973	934	868	812	767	733	690	**686**	**593**	528	379
BB-CPL	983	936	781	606	461	368	280	257	225	262	384
RP-RAN	926	820	635	493	408	292	260	205	143	109	116
RP-SGL	976	908	792	656	552	420	339	277	214	165	118
RP-CPL	917	821	619	502	373	276	189	126	110	89	108
Total	985	950	894	848	813	776	754	743	679	643	559
F	985	952	-	-	-	-	-	-	-	-	-

Table 6: Instances of size 100 with 0.3 compatibility (1 second per instance)

It is obvious that allowing more time for any particular algorithm variant might increase the number of instances solved. To obtain a feel for this, we re-ran all of the experiments on instances of size 100 but allowing 10 seconds per instance rather than 1 second, and a subset of the experiments, for the most successful algorithm variants, on instances of size 500 but allowing 50 seconds per instance rather than 5 seconds. The results are shown in Tables 7 and 8.

Finally, we conducted a much smaller-scale study of larger instances, of sizes 10000, 20000 and 30000 (the latter of the order of magnitude of NRMP instances), to give an indication as to whether the conclusions drawn from studying smaller instances would continue to hold in such cases. Table 9 summarises the numbers of instances solved for the cases of size 30000, where just 10 instances in each set were involved. Note that here, each algorithm was allowed 30 seconds execution time on each instance, but, with only a handful of exceptions, the solved instances were, in practice, solved in under 1 second. The pattern of solved instances was similar for problem sizes 10000 and 20000.

This smaller scale study of instances of larger size confirms the general trends dis-

	Number of couples										
Algorithm	2	5	10	15	20	25	30	35	40	45	50
C-RAN	981	963	907	876	833	816	778	715	664	623	572
C-STA	978	938	830	760	701	681	641	585	572	542	502
C-SGL	981	963	906	861	829	816	782	737	686	628	**580**
C-CPL	974	927	826	778	739	724	710	654	639	611	573
C-RLP	968	921	824	711	571	438	308	207	149	89	56
BB-RAN	**983**	**966**	**916**	**883**	**853**	**838**	**802**	**751**	701	629	530
BB-SCO	968	922	819	722	604	527	444	306	248	172	107
BB-USE	982	962	911	872	841	820	782	735	700	**653**	579
BB-USS	968	929	863	805	751	714	687	670	668	637	579
BB-SGL	968	931	865	818	779	749	729	722	**706**	644	548
BB-CPL	981	954	842	697	576	523	451	387	362	401	534
RP-RAN	928	840	694	601	492	441	394	358	321	305	250
RP-SGL	975	925	796	714	617	557	493	418	358	312	258
RP-CPL	917	843	693	590	500	416	375	325	293	253	262
Total	984	967	921	886	861	848	831	803	786	761	727

Table 7: Instances of size 100 (10 seconds per instance)

	Number of couples										
Algorithm	12	25	50	75	100	125	150	175	200	225	250
C-RAN	**976**	957	908	864	813	752	654	453	233	75	9
C-SGL	**976**	957	902	861	801	**759**	**716**	582	319	100	8
BB-RAN	**976**	**958**	**911**	**870**	**815**	732	538	299	100	27	2
BB-USS	963	934	880	825	764	717	697	**628**	**414**	**154**	**15**
BB-SGL	963	934	882	830	771	727	704	612	357	93	3
Total	976	958	911	870	821	778	756	683	493	199	24

Table 8: Instances of size 500 (50 seconds per instance)

Algorithm	750	1500	3000	4500	6000	7500	9000	10500	12000	13500	15000
C-RAN	10	9	10	7	6	5	0	0	0	0	0
C-STA	10	9	8	6	6	5	4	0	0	0	0
C-SGL	10	9	10	7	6	5	5	0	0	0	0
C-CPL	10	9	5	5	6	0	0	0	0	0	0
C-RLP	10	9	0	0	0	0	0	0	0	0	0
BB-RAN	10	9	10	1	0	0	0	0	0	0	0
BB-SCO	10	8	6	4	4	2	1	0	0	0	0
BB-USE	10	9	10	0	0	0	0	0	0	0	0
BB-USS	10	9	10	6	5	1	0	0	0	0	0
BB-SGL	10	9	10	6	5	2	0	0	0	0	0
BB-CPL	10	9	6	0	0	0	0	0	0	0	0
RP-RAN	10	8	2	2	0	0	0	0	0	0	0
RP-SGL	10	9	6	6	1	2	0	0	0	0	0
RP-CPL	9	8	4	5	0	0	0	0	0	0	0
Total	10	9	10	7	6	5	5	0	0	0	0

Table 9: Instances of size 30000 solved by various algorithms (30 seconds per instance)

cernable for smaller instances. When the proportion of couples is small, the likelihood of finding a stable solution remains high. However, as the ratio of couples to single applicants increases the decrease in the proportion of solved instances becomes steeper, reinforcing the belief that the absolute number of couples, and not just the proportion, is significant in this respect. For instances of size 10000 or greater, none of the algorithms was able to solve a single instance in which more than 60% of the applicants were in couples. The other notable feature for larger instances is the prominence of Algorithm C-SGL: this variant dominated all of the others in the sense that every single instance of size 10000 or more that was solved by at least one of the algorithm variants was solved by this particular one.

8 Discussion and conclusions

In this section we make a number of observations, on the basis of the empirical study, regarding the effectiveness of the various algorithms and the question of the existence of stable matchings.

Algorithm C
Variants C-RAN (random waiting list) and C-SGL (random waiting list subject to prioritising single applicants) are the most successful, the former generally being superior when the number of linked applicants is up to about 40% of the total, and the latter when this ratio is exceeded. However C-SGL appears to become pre-eminent in the case of larger instances. Use of a stack waiting list is generally less successful (except for a curious unexplained outlier in the 500 data set when all applicants are in couples) – this is a significant observation in view of the tendency for implementations of Gale-Shapley like algorithms to organise applicants in a stack (as in the algorithm described by Klaus et al. [7]). Prioritising the review list appears to be a bad strategy.

Algorithm BB
The relative success rate of the variants of Algorithm BB is very much dependent on the instance size and number of couples, but the effectiveness of this conceptually simple approach, at least when the proportion of couples is relatively small, came as something of a surprise. It is curious how BB-USE is better than BB-USS on data sets of size 100, but BB-USS is markedly better on the larger data sets. Also strange is the fact that BB-CPL consistently outperforms BB-SGL when the number of couples is very small, but otherwise BB-SGL is greatly superior. BB-RAN is never beaten when the proportion of linked applicants is small (up to 30% of applicants). On larger data sets with many couples BB-USS and BB-SGL perform much better than the other variants, so it seems to be important to prioritise single applicants. Prioritising by score seems to be a poor strategy (perhaps because it encourages cyclic behaviour) – but again there are curious outliers in the 500 and 1000 data sets when all applicants are in couples.

Algorithm RP
It appears that our version of the Roth-Peranson algorithm is generally not competitive with the other algorithms. This may not be entirely surprising, since in this approach stability has to be achieved over and over again as each successive agent is admitted. RP-SGL (single applicants admitted first) is consistently the best of the three variants, confirming the findings reported in the original paper of Roth and Peranson [15].

Algorithm F
Although Algorithm F could be run only on instances with 100 applicants and 2 or 5 couples, the results are of some interest since they give a definitive answer to the question of how many such instances do admit a stable matching. In the case of 2 couples (first column of Tables 1 and 6), all such instances were identified by at least one of the heuristics,

but with 5 couples (second column of Tables 1 and 6) there were 2 instances with a stable matching that none of the heuristics was able to solve.

Overall comparison
Variants of Algorithm BB – BB-RAN and BB-USE in particular – are the clear winners on data sets of size 100. On larger data sets, variants of Algorithm C – C-RAN and C-SGL in particular – are competitive, and the latter variant clearly outperforms any variant of Algorithm BB as the proportion of couples grows, and indeed is unambiguously the most effective variant for larger instances. Ideally, one might have wished that a single algorithm would emerge as being unambiguously the best, but unfortunately the empirical results do not lead to such a conclusion. Instead, it appears that it is a good strategy, in practice, to have a number of algorithm variants available to maximise the chances of finding a stable solution for any particular instance.

General
When the proportion of couples is low, the best algorithms solve all, or almost all, of the instances that can be solved by any of the algorithms. Based on the limited evidence from Algorithm F, it appears that all, or almost all, of such instances that do have a stable matching can be identified and solved by the best heuristics. By contrast, for higher proportions of couples, the total number of instances solved, aggregated over all of the algorithm variants, can be substantially greater than the number solved by any one algorithm. An increase in both the number of couples and the proportion of couples makes it harder to find a stable matching, but we do not have the evidence to judge to what extent this is because a stable matching is less likely to exist. For all of the algorithms, prioritising single applicants rather than couples appears to be a good strategy, in other words that assigning the "easier" agents first seems to be advisable. We have no strong intuition as to why this should be the case, as in some other contexts, such as the use of variable-ordering heuristics in constraint programming, handling the more awkward cases first can often be the preferred strategy [4]. This is an issue that may be worthy of further consideration and investigation.

Additional observations
Running the various algorithms on numerous data sets revealed a number of additional interesting facts. For example, just as the number of solved instances decreased with an increasing couples to singles ratio, so the average number of proposal steps for the solved instances increased. For small number of couples, and even for the less effective algorithms, it appeared that most instances that were solved at all were solved very quickly. However, for larger numbers of couples many of the solved instances required a large number of proposals.

When the compatibility probability is reduced, so that couples' preference lists are typically shorter, the numbers of instances for which a stable solution was found were generally somewhat reduced. The overall pattern of results was somewhat similar to the earlier case. However, the successful variants of Algorithm C seemed to be less affected than the successful variants of Algorithm BB, and the variants of Algorithm RP seemed to be most severely affected. Somewhat bizarrely, and inexplicably, the least effective of all of the variants, Algorithms C-RLP and BB-SCO, improve when the compatibility probability is lower. The use of a relatively high compatibility probability in our main experiments can be justified by the fact that, in practice, linked applicants tend to submit highly correlated preference lists, typically focusing on one geographical region in which a high proportion of pairs of programmes are compatible.

When the algorithms are allowed more time to find stable matchings, there is very little change in the results pattern for low or moderate numbers of couples. Indeed, in

the trials that we conducted, few if any additional instances of this kind were solved when the time available was increased by a factor of 10. In one or two cases, the shorter runs actually solved more such instances (presumably because in just one or two instances a 'lucky' sequence of random choices was made). For larger numbers of couples, there were some significant increases in the numbers of instances solved. With one notable exception, this trend of improvement was similar for all variants, though slightly more accentuated for variants of Algorithm C when the number of couples was very high. The exception was Algorithm BB-SCO, where the results, in all cases, were completely unchanged when extra time was allowed, revealing the fact that failure to find a stable matching in this case is the result of cyclic behaviour that appears to be inevitable in any particular instance because of the fixed order in which best blockers are chosen.

More general contexts
Although the results and observations presented above are based on SHRC, the particular version of the HRC problem that is relevant in our application domain, we believe that they are indicative of the likely behaviour of the various algorithms were they to be tailored for more general settings, where both programmes and couples have the freedom to form their own preference lists. We note that our definition of stability is appropriate for this more general context provided that the concept of superiority is replaced by that of position within individual programme preference lists. Each of algorithms C, BB and RP can easily be amended to handle the more general problem, except that, in the case of Algorithms C and BB, Phase 1 is no longer applicable and Phase 2 therefore starts with an empty matching.

Our empirical study was based exclusively on instances with a strictly ordered master list. If, in a practical setting such as SFAS, the master list has ties, then these can, of course, be broken arbitrarily to produce a strictly ordered list. However, the prospects of finding a stable solution are even greater in this case, since different instances of SHRC can be created by breaking ties in different ways, and failure to find a solution for one such instance need not imply failure for another.

Acknowledgment

We acknowledge the contribution of one of the reviewers who suggested the investigation of the problem from the point of view of fixed-parameter tractability.

References

[1] B. Aldershof and O.M. Carducci. Stable matchings with couples. *Discrete Applied Mathematics*, 68:203–207, 1996.

[2] P. Biró, R.W. Irving, and I. Schlotter. Stable matching with couples – theory and practice. Technical report, no. TR-2011-324, University of Glasgow, School of Computing Science, 2011.

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

[4] R.M. Haralick and G.L. Elliott. Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence*, 14(3):263–313, 1980.

[5] R.W. Irving, D.F. Manlove, and S. Scott. The stable marriage problem with master preference lists. *Discrete Applied Mathematics*, 156:2959–2977, 2008.

[6] B. Klaus and F. Klijn. Stable matchings and preferences of couples. *Journal of Economic Theory*, 121:75–106, 2005.

[7] B. Klaus, F. Klijn, and J. Massó. Some things couples always wanted to know about stable matchings (but were afraid to ask). *Review of Economic Design*, 11:175–184, 2007.

[8] B. Klaus, F. Klijn, and T. Nakamura. Corrigendum: Stable matchings and preferences of couples. *Journal of Economic Theory*, 144:2227–2233, 2009.

[9] F. Kojima, P.A. Pathak, and A.E. Roth. Matching with couples: stability and incentives in large markets. Working paper, 2010.

[10] D. Marx and I. Schlotter. Stable assignment with couples: parameterized complexity and local search. In *Proceedings of IWPEC 2009: The Fourth International Workshop on Parameterized and Exact Computation*, volume 5917 of *Lecture Notes in Computer Science*, pages 300–311. Springer, 2009. To appear in *Discrete Optimization*.

[11] E. McDermid and D.F. Manlove. Keeping partners together: Algorithmic results for the hospitals / residents problem with couples. *Journal of Combinatorial Optimization*, 19:279–303, 2010.

[12] `http://www.nrmp.org/about_nrmp/how.html` (National Resident Matching Program website).

[13] E. Ronn. NP-complete stable matching problems. *Journal of Algorithms*, 11:285–304, 1990.

[14] A. E. Roth. The evolution of the labor market for medical interns and residents: a case study in game theory. *Journal of Political Economy*, 6(4):991–1016, 1984.

[15] 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.

[16] A. E. Roth and J. H. Vande Vate. Random paths to stability in two-sided matching. *Econometrica*, 58(6):1475–1480, 1990.

[17] A.E. Roth. Deferred acceptance algorithms: history, theory, practice, and open questions. *International Journal of Game Theory*, 36:537–569, 2008.

[18] J. Sethuraman, C-P. Teo, and L. Qian. Many-to-one stable matching: geometry and fairness. *Mathematics of Operations Research*, 31:581–596, 2006.

[19] `http://www.nes.scot.nhs.uk/sfas/` (Scottish Foundation Allocation Scheme website).