A Constraint Programming Approach to the Hospitals / Residents Problem

David F. Manlove[⋆,†], Gregg O'Malley[†], Patrick Prosser, and Chris Unsworth[‡]

Department of Computing Science, University of Glasgow, Glasgow G12 8QQ, UK.
`davidm/gregg/pat/chrisu@dcs.gla.ac.uk`.

Abstract. An instance I of the Hospitals / Residents problem (HR) involves a set of residents (graduating medical students) and a set of hospitals, where each hospital has a given capacity. The residents have preferences for the hospitals, as do hospitals for residents. A solution of I is a *stable matching*, which is an assignment of residents to hospitals that respects the capacity conditions and preference lists in a precise way. In this paper we present constraint encodings for HR that give rise to important structural properties. We also present a computational study using both randomly-generated and real-world instances. We provide additional motivation for our models by indicating how side constraints can be added easily in order to solve hard variants of HR.

1 Introduction

Gale and Shapley described in their seminal paper [7] the classical Hospitals / Residents problem (HR), referred to by the authors as the College Admissions problem. An instance of HR involves a set of *residents* (i.e. graduating medical students) and a set of *hospitals*. Each resident ranks in order of preference a subset of the hospitals. Each hospital has an integral *capacity*, and ranks in order of preference those residents who ranked it. We seek to match each resident to an acceptable hospital, in such a way that a hospital's capacity is never exceeded. Moreover the matching must be *stable* – a formal definition of stability follows, but informally stability ensures that no resident and hospital, not already matched together, would rather be assigned to one another than remain with their assignees. Such a resident and hospital could form a private arrangement outside the matching, undermining its integrity. Gale and Shapley [7] described a linear-time algorithm for finding a stable matching, given an instance of HR.

Many centralised matching schemes that automate the process of assigning residents to hospitals employ algorithms that solve HR and its variants [25]. For example, the National Resident Matching Program (NRMP) in the US [23] is perhaps the largest such scheme. The NRMP has been in operation since 1952 and handles the annual allocation of some 31,000 residents to hospitals.

[⋆] Supported by RSE / Scottish Executive Personal Research Fellowship and EPSRC grant EP/E011993/1.
[†] Supported by EPSRC grant GR/R84597/01.
[‡] Supported by an EPSRC studentship.

Counterparts of the NRMP elsewhere are the Canadian Resident Matching Service (CaRMS) [5] and the Scottish Foundation Allocation Scheme (SFAS) [13]. Similar matching schemes are also used in educational and vocational contexts.

A special case of HR occurs when each hospital has capacity 1 – this is the Stable Marriage problem with Incomplete lists (SMI). In this context, residents are referred to as *men*, whilst hospitals are referred to as *women*. A special case of SMI occurs when the numbers of men and women are equal, and each man finds all women acceptable and vice versa – this is the classical Stable Marriage problem (SM), also introduced by Gale and Shapley [7]. A specialised linear-time algorithm for SM, known as the Gale / Shapley (GS) algorithm [7], can be generalised to the SMI case [12, Section 1.4.2]. Using a process known as "cloning hospitals" (described in more detail in Section 3), a given instance I of HR may be transformed to an instance J of SMI, and the GS algorithm can be applied to J in order to give a stable matching in I. However in general this method expands the instance size, so that in practice specialised algorithms (such as those described in [12, Section 1.6]; see also Figure 2) are used to solve HR directly and achieve a better worst-case time complexity.

Over the last few decades, stable matching problems, and SM in particular, have been the focus of much attention in the literature [7, 15, 12, 26]. Several encodings of SM and its variants as a Constraint Satisfaction Problem (CSP) have been formulated [1, 8, 16, 9–11, 19, 29, 30]. Moreover, recent papers have focussed on distributed variants of SM (including the Stable Roommmates problem, a non-bipartite extension of SM) where preference lists are to be kept private [27, 28, 3, 4]. However, no encoding for HR has been considered before now.

This paper is concerned with a Constraint Programming (CP) approach to solving HR. We firstly present in Section 3 a cloned model for HR, indicating how existing formulations of SMI as a CSP [8] can be used in order to model HR. We then present in Section 4 a constraint-based model of HR that deals directly with an HR instance without cloning, achieving improved time and space complexities. We show that the effect of Arc Consistency (AC) propagation [2] applied to this model yields the same structure as the action of established algorithms for HR [7, 12]. As a consequence, a stable matching for the given HR instance can be obtained without search (in fact we can in general obtain two complementary stable matchings following AC propagation, with optimality properties for the residents and hospitals respectively). We also demonstrate how a failure-free enumeration can be used to find all solutions for a given HR instance without search. These results therefore extend analogous results presented in [8] for SMI. In Section 5, we present a specialised n-ary constraint for HR, comparing and constrasting the time and space requirements for establishing AC with the models presented in Sections 3 and 4. Then, in Section 6, we describe the results of an empirical study which compares the various models presented in this paper in practice, on both randomly-generated and real-world data.

The models in Sections 4 and 5 are non-trivial extensions of earlier constraint models presented for SMI [8, 19, 29, 30]. In the SMI case, clearly each woman can be assigned at most one man, but to model an HR instance without cloning, the

Residents' preferences	M_0	M_z	Hospitals' preferences
$r_1 : h_1\ h_3$	–	–	$h_1 : (2) : \underline{r_3}\ r_7\ \underline{r_5}\ \underline{r_2}\ r_4\ r_6\ r_1$
$r_2 : \underline{h_1}\ h_5\ \underline{h_4}\ \underline{h_3}$	h_1	h_3	$h_2 : (3) : r_5\ \underline{r_6}\ r_3\ \underline{r_4}$
$r_3 : \underline{h_1}\ h_2\ h_5$	h_1	h_1	$h_3 : (1) : \underline{r_2}\ \underline{r_5}\ r_6\ r_1\ r_7$
$r_4 : h_1\ \underline{h_2}\ h_4$	h_2	h_2	$h_4 : (1) : \underline{r_8}\ \underline{r_2}\ r_4\ \underline{r_7}$
$r_5 : \underline{h_3}\ \underline{h_1}\ h_2$	h_3	h_1	$h_5 : (1) : r_3\ \underline{r_7}\ r_6\ \underline{r_8}\ r_2$
$r_6 : h_3\ \underline{h_2}\ h_1\ h_5$	h_2	h_2	
$r_7 : h_3\ \underline{h_4}\ \underline{h_5}\ h_1$	h_4	h_5	
$r_8 : \underline{h_5}\ \underline{h_4}$	h_5	h_4	

Fig. 1. An HR instance. The GS-list entries are underlined, and the middle two columns indicate the residents' assigned hospitals in M_0 and M_z (r_1 is unassigned in both).

main challenges are to maintain a representation of the *set* of assignees of a given hospital h_j, and of the identity of the worst resident assigned to h_j.

The benefits of our approach are two-fold: firstly, the CSP models presented here for HR indicate that AC propagation using a CP toolkit yields the same structure as given by established linear-time algorithms for HR, from which all solutions for a given instance can be generated in a failure-free manner without search. Secondly, and more importantly, our models can be used as a basis on which additional constraints can be imposed, covering variants of HR that arise naturally in practical applications, but which cannot be accommodated easily by existing algorithms. These include variants of HR that are NP-hard, and for which no polynomial-time algorithm is currently known. Examples of such variants, where appropriate side-constraints are suggested in three cases, are given in Section 7 to provide additional motivation for our approach.

In the next section we present notation and terminology relating to HR, which will be assumed in the remainder of this paper, and we also present some important structural and algorithmic results.

2 Definitions and fundamental results

We now give a formal definition of HR. An instance I of HR comprises a set $R = \{r_1, \ldots, r_n\}$ of *residents* and a set $H = \{h_1, \ldots, h_m\}$ of *hospitals*. Each resident $r_i \in R$ has an *acceptable* set of hospitals $A_i \subseteq H$; moreover r_i ranks A_i in strict order of preference. For each $h_j \in H$, denote by $B_j \subseteq R$ those residents who find h_j acceptable; h_j ranks B_j in strict order of preference. Finally, each hospital $h_j \in H$ has an associated *capacity*, denoted by $c_j \in \mathbb{Z}^+$, indicating the number of *posts* that h_j has. For each $r_i \in R$, let l_i^r denote the length of r_i's preference list, and for each $h_j \in H$, let l_j^h denote the length of h_j's preference list; we assume that $c_j \leq l_j^h$. Let L denote the total length of the residents' preference lists in I. Given $r_i \in R$ and $h_j \in A_i$, define $rank(r_i, h_j)$ to be the position of h_j in r_i's preference list; $rank(h_j, r_i)$ is defined similarly. An example HR instance is shown in Figure 1 (the hospital capacities are indicated in brackets).

An *assignment* M is a subset of $R \times H$ such that $(r_i, h_j) \in M$ implies that $h_j \in A_i$ (i.e. r_i finds h_j acceptable). If $(r_i, h_j) \in M$, we say that r_i is *assigned*

```
$M = \emptyset$;
while (some $r_i \in R$ is unassigned
  and $r_i$ has a non-empty list)
  $h_j$ = first hospital on $r_i$'s list;
  /* $r_i$ applies to $h_j$ */
  $M = M \cup \{(r_i, h_j)\}$ ;
  if ($h_j$ is over-subscribed)
    $r_k$ = worst resident assigned to $h_j$;
    $M = M \backslash \{(r_k, h_j)\}$ ;
  if ($h_j$ is full)
    $r_k$ = worst resident assigned to $h_j$;
    for (each successor $r_z$ of $r_k$ on $h_j$'s list)
      delete the pair $(r_z, h_j)$;
```

```
$M = \emptyset$;
while (some $h_j \in H$ is under-subscribed
  and some $r_i \in B_j$ is not assigned to $h_j$)
  $r_i$ = first such resident on $h_j$'s list;
  /* $h_j$ offers a post to $r_i$ */
  if ($r_i$ is assigned)
    $h_k = M(r_i)$;
    $M = M \backslash \{(r_i, h_k)\}$;
  $M = M \cup \{(r_i, h_j)\}$;
  for (each successor $h_z$ of $h_j$ on $r_i$'s list)
    delete the pair $(r_i, h_z)$;
```

Fig. 2. RGS algorithm for HR; HGS algorithm for HR.

to h_j, and h_j is *assigned* r_i. For any $q \in R \cup H$, we denote by $M(q)$ the set of assignees of q in M. If $r_i \in R$ and $M(r_i) = \emptyset$, we say that r_i is *unassigned*, otherwise r_i is *assigned*. Similarly, any hospital $h_j \in H$ is *under-subscribed*, *full* or *over-subscribed* according as $|M(h_j)|$ is less than, equal to, or greater than c_j, respectively.

A *matching* M is an assignment such that $|M(r_i)| \leq 1$ for each $r_i \in R$ and $|M(h_j)| \leq c_j$ for each $h_j \in H$ (i.e. each resident is assigned to at most one hospital, and no hospital is over-subscribed). For convenience, given a resident $r_i \in R$ such that $M(r_i) \neq \emptyset$, where there is no ambiguity the notation $M(r_i)$ is also used to refer to the single member of $M(r_i)$.

A *blocking pair* relative to a matching M is a (resident,hospital) pair $(r_i, h_j) \in (R \times H) \backslash M$ such that (i) $h_j \in A_i$, (ii) either r_i is unassigned in M or prefers h_j to $M(r_i)$, and (iii) either h_j is under-subscribed or prefers r_i to at least one member of $M(h_j)$. A matching is *stable* if it admits no blocking pair.

Gale and Shapley [7] described an algorithm for finding a stable matching in a given HR instance I, which is known as the *resident-oriented* Gale/Shapley (RGS) algorithm [12, Section 1.6.3]. This algorithm finds the *resident-optimal* stable matching M_0 in I, in which each assigned resident is assigned to the best hospital that he could obtain in any stable matching. On the other hand, the *hospital-oriented* (HGS) algorithm [12, Section 1.6.2] is a second algorithm for HR that finds the *hospital-optimal* stable matching M_z in I, in which each hospital is assigned the best set of residents that it could obtain in any stable matching. Figure 1 includes columns that give M_0 and M_z for the example HR instance shown. In general, the optimality property of each of M_0 and M_z is achieved at the expense of the hospitals and residents respectively (the "pessimality" of each of these matchings for the relevant parties is discussed in Sections 1.6.2 and 1.6.5 of [12]). The RGS and HGS algorithms for HR are shown in Figure 2 (the term "delete the pair (r_i, h_j)" refers to the operations of deleting r_i from h_j's preference list and vice versa). Using a suitable choice of data structures (extending those described in [12, Section 1.2.3]), both the RGS and the HGS algorithms can be implemented to run in $O(L)$ time and $O(nm)$ space.

The deletions made by each of the RGS and HGS algorithms have the effect of reducing the original set of preference lists in I. The reduced lists returned by the RGS (respectively HGS) algorithm are known as the *RGS-lists* (respectively *HGS-lists*). The intersection of the RGS-lists and the HGS-lists yields the *GS-lists.* (E.g. the GS-lists for the HR instance shown in Figure 1 are represented as underlined preference list entries.) The GS-lists in I have several useful properties, which are summarised below (these properties follow as a consequence of Lemmas 1.6.2 and 1.6.4, and Theorems 1.6.1 and 1.6.2 of [12]):

Theorem 1. *For a given instance of HR,*
(i) all stable matchings are contained in the GS-lists;
(ii) in M_0, each resident with a non-empty GS-list is assigned to the first hospital on his GS-list, whilst each resident with an empty GS-list is unassigned;
(iii) in M_z, each hospital h_j is assigned the first m_j members of its GS-list, where $m_j = \min\{c_j, g_j^h\}$ and g_j^h is the length of h_j's GS-list.

Given any $q \in R \cup H$, we denote by $GS(q)$ the set of hospitals or residents (as appropriate) that belong to q's GS-list in I.

Additional important results concern residents who are unassigned, and hospitals that are under-subscribed, in stable matchings in I. These results are collectively known as the *Rural Hospitals Theorem* [12, Section 1.6.4], and may be stated as follows:

Theorem 2. *For a given instance of HR,*
(i) each hospital is assigned the same number of residents in all stable matchings;
(ii) exactly the same residents are unassigned in all stable matchings;
(iii) any hospital that is under-subscribed in one stable matching is assigned precisely the same set of residents in all stable matchings.

3 A cloned model

In this section we indicate how an instance of HR may be reduced to an instance of SMI by "cloning" hospitals. This technique is described in [12, p.38]; see also [26, pp.131-132]. For completeness, we briefly restate the construction here. Let I be an instance of HR. We form an instance J of SMI by replacing each hospital $h_j \in H$ by c_j women in J, denoted by h_j^k $(1 \leq k \leq c_j)$. The preference list of h_j^k in J is identical to that of h_j in I. Each resident r_i in I corresponds to a man r_i in J, and each hospital h_j in r_i's list in I is replaced by $h_j^1\ h_j^2\ \ldots\ h_j^{c_j}$, in that order, in J. It may then be shown that the stable matchings in I are in one-one correspondence with the stable matchings in J.

In order to obtain the GS-lists of I, we can model J using the "conflict matrices" encoding of SMI as presented in [8]. In general AC may be established in $O(ed^r)$ time, where e is the number of constraints, d is the domain size, and r is the arity of each constraint [2]. Due to the cloning technique, the number of women in J is $\sum_{j=1}^{m} c_j = O(cm)$, where $c = \max\{c_j : h_j \in H\}$. Given the construction of the encoding in J [8], it follows that $e = O(nmc)$, $d = O(n+m)$ and $r = 2$, so that the time and space complexities for finding the GS-lists in I using the cloned model are $O((n+m)^4c)$ and $O((nmc)^2)$ respectively.

1.	$y_{j,k} < y_{j,k+1}$	$(1 \le j \le m, 1 \le k \le c_j - 1)$
2.	$y_{j,k} \ge q \Rightarrow x_i \le p$	$(1 \le j \le m, 1 \le k \le c_j, 1 \le q \le l_j^h)$
3.	$x_i \ne p \Rightarrow y_{j,k} \ne q$	$(1 \le i \le n, 1 \le p \le l_i^r, 1 \le k \le c_j)$
4.	$(x_i \ge p \wedge y_{j,k-1} < q) \Rightarrow y_{j,k} \le q$	$(1 \le i \le n, 1 \le p \le l_i^r, 1 \le k \le c_j)$
5.	$y_{j,c_j} < q \Rightarrow x_i \ne p$	$(1 \le j \le m, c_j \le q \le l_j^h)$

Fig. 3. Constraints for the CSP model of an HR instance.

4 A direct CSP-based model

We now present a direct CSP encoding of an HR instance that avoids cloning. Let I be an instance of HR. For $r_i \in R$ and $h_j \in H$, we use the terminology r_i *applies (or is assigned) to* h_j*'s* k^{th} *post* $(1 \le k \le c_j)$ in the case that h_j prefers exactly $k-1$ members of $M(h_j)$ to r_i. Also given a matching M, we denote the resident who is assigned to h_j's k^{th} post in M by $M_k(h_j)$ $(1 \le k \le |M(h_j)|)$.

We construct a CSP instance J with variables $X = \{x_1, \ldots, x_n\}$ and $Y = \{y_{j,k} : 1 \le j \le m \wedge 0 \le k \le c_j\}$, whose domains are initially defined as follows:

$$\begin{array}{ll} dom(x_i) = \{1, 2, \ldots, l_i^r\} \cup \{m+1\} & (1 \le i \le n) \\ dom(y_{j,0}) = \{0\} & (1 \le j \le m) \\ dom(y_{j,k}) = \{k, k+1, \ldots, l_j^h\} \cup \{n+k\} & (1 \le j \le m \wedge 1 \le k \le c_j). \end{array}$$

For the x_i variables $(1 \le i \le n)$, the value $m+1$ corresponds to the case that r_i's GS-list is empty, whilst the remaining values correspond to the ranks of preference list entries that belong to the GS-lists. A similar meaning applies to the $y_{j,k}$ variables $(1 \le j \le m,\ 1 \le k \le c_j)$, except that the value $n+k$ corresponds to the case that h_j's GS-list contains fewer than k entries.

More specificially, if $\min(dom(x_i)) \ge p$ $(1 \le p \le l_i^r)$, then during the RGS algorithm, r_i applies to his p^{th}-choice hospital or worse, so that in M_0, either r_i is assigned to such a hospital or is unassigned. Similarly if $\max(dom(x_i)) \le p$, then during the HGS algorithm, r_i was offered a post by his p^{th}-choice hospital or better, so that r_i is assigned to such a hospital in M_z.

From the hospitals' point of view, if $\min(dom(y_{j,k})) \ge q$ $(1 \le q \le l_j^h)$, then during the HGS algorithm, h_j offers its k^{th} post to its q^{th}-choice resident or worse, so that in M_z, either h_j's k^{th} post is filled by such a resident, or is unfilled. Similarly if $\max(dom(y_{j,k})) \le q$, then during the RGS algorithm, some resident r_i applied to h_j's k^{th} post, where $rank(h_j, r_i) \le q$, so that h_j's k^{th} post is filled by r_i or better in M_0.

The constraints in J are given in Figure 3 (in the context of Constraints 2-5, p denotes the rank of h_j in r_i's list and q denotes the rank of r_i in h_j's list). An interpretation of the constraints is now given. Constraint 1 ensures that h_j's filled posts are occupied by residents in preference order, and that if post $k-1$ is unfilled then so is post k. Constraint 2 states that if h_j's k^{th} post is filled by a resident no better than r_i or is unfilled, then r_i must be assigned to a hospital no worse than h_j. Constraints 3 and 5 reflect the consistency of deletions carried out by the HGS and RGS algorithms respectively (i.e. if h_j is deleted from r_i's list, then r_i is deleted from h_j's list, and vice versa). Finally Constraint 4 states that if r_i is assigned to a hospital no better than h_j or is unassigned, and h_j's first $k-1$ posts are filled by residents better than r_i, then h_j's k^{th} post must be filled by a resident at least as good as r_i.

It turns out that establishing AC in J yields a set of domains that correspond to the GS-lists in I. To demonstrate this, we define some additional notation. For each j $(1 \leq j \leq m)$, define $S_j = \{rank(h_j, r_i) : r_i \in GS(h_j)\}$. Let d_j denote the number of residents assigned to hospital h_j in any stable matching in I. For each k $(1 \leq k \leq d_j)$, let $q_{j,k} = rank(h_j, M_{z_k}(h_j))$ and $t_{j,k} = rank(h_j, M_{0_k}(h_j))$. The *GS-domains* for the variables in J are defined as follows:

$$dom(x_i) = \begin{cases} \{rank(r_i, h_j) : h_j \in GS(r_i)\}, & \text{if } GS(r_i) \neq \emptyset \\ \{m+1\}, & \text{otherwise} \end{cases}$$

$$dom(y_{j,k}) = \begin{cases} \{s \in S_j : q_{j,k} \leq s \leq t_{j,k}\}, & \text{if } 1 \leq k \leq d_j \\ \{n+k\}, & \text{if } d_j + 1 \leq k \leq c_j. \end{cases}$$

We prove in [20] (we omit the proof here for space reasons) that, following AC propagation in J, the domain of each variable is a subset of its GS-domain, and conversely, the GS-domains are arc consistent in J. Given that AC algorithms find the unique maximal set of arc consistent domains [2], we therefore have:

Theorem 3. *Let I be an instance of HR, and let J be a CSP instance obtained by the encoding of this section. Then the domains remaining after AC propagation in J correspond exactly to the GS-lists in I.*

For example, in the context of the HR instance given in Figure 1, the GS-domains for x_2, $y_{1,1}$ and $y_{1,2}$ are $\{1,3,4\}$, $\{1\}$ and $\{3,4\}$ respectively. In general, following AC propagation in J, matchings M_0 and M_z may be obtained as follows. Let $x_i \in X$. If $x_i = m+1$, resident r_i is unassigned in both M_0 and M_z. Otherwise, in M_0 (respectively M_z), r_i is assigned to the hospital h_j such that $rank(r_i, h_j) = p$, where $p = \min(dom(x_i))$ (respectively $p = \max(dom(x_i))$).

In the context of the time complexity function for establishing AC as mentioned in Section 3, for this encoding we have $e = O(Lc)$ and $d = O(n+m)$ (recall that L is the total length of the residents' preference lists in I). The constraints shown in Figure 3 may be revised in $O(1)$ time, assuming that upper and lower bounds for the variables' domains are maintained throughout propagation. It follows by [31] that the time complexity for establishing AC in this model is $O(Lc(n+m))$. Since the space complexity is $O(Lc)$, the model presented in this section is more efficient than the cloned model in terms of both time and space.

The next result, proved in [20] (we also omit the proof here), states that the encoding presented above can be used to enumerate all the solutions of I in a failure-free manner using AC propagation with a value-ordering heuristic.

Theorem 4. *Let I be an instance of HR and let J be a CSP instance obtained by the encoding of this section. Then the following search process enumerates all solutions in I without repetition and without ever failing due to an inconsistency:*

- *AC is established as a preprocessing step, and after each branching decision including the decision to remove a value from a domain;*
- *if all domains are arc consistent and some variable x_i has two or more values in its domain then search proceeds by setting x_i to the minimum value p in its domain. On backtracking, the value p is removed from the domain of x_i;*
- *when a solution is found, it is reported and backtracking is forced.*

5 A specialised n-ary constraint

We now present a specialised n-ary constraint HRN for the Hospitals / Residents problem. A model based on HRN requires only one constraint for the whole problem. We assume that this constraint will be processed by an AC5 [31] type arc consistency algorithm. That is, the algorithm has a stack of calls to revise constraints, and if a variable v loses a value then a call to all constraints involving v will be added to the stack along with the removed value.

5.1 Preliminaries

Our model involves a constrained integer variable x_i corresponding to each resident $r_i \in R$, where the domain values represent ranks, as in Section 4. In addition, we associate a single constrained integer variable y_j corresponding to each hospital $h_j \in H$ with similar meanings for the domain values. In this model only the x variables are search variables, meaning that a solution consists of a single value being assigned to each x variable, but the y variables may have multiple values remaining in their associated domains.

We assume that we have the following functions, each being of $O(1)$ complexity, that operate over constrained integer variables:

- $getMin(v)$ delivers the smallest value in $dom(v)$.
- $getMax(v)$ delivers the largest value in $dom(v)$.
- $getValue(v, a)$ returns the a^{th} smallest value in $dom(v)$, if $|dom(v)| < a$ then $getMax(v)$ is returned.
- $setMax(v, a)$ removes all values greater than a from $dom(v)$.
- $remVal(v, a)$ removes the value a from $dom(v)$.
- $PL(r_i, k)$ returns the k^{th} entry in r_i's preference list.
- $swap(a, b)$ swaps the values of the variables a and b.

The HRN constraint also requires the following data structures:

- $\check{x}$ is an array of n reversible integer variables containing the previous lower bounds of all x variables. All are initially set to $min(x) - 1$. On backtracking the values in $\check{x}$ are restored by the solver.
- $\check{y}$ is an array of m reversible integer variables containing the value that represents y's least favourite resident to be offered a post at y. For hospital h_j, $\check{y}_j$ will equal the c_j^{th} lowest value in $dom(y_j)$. All are initially set to $min(y) - 1$. On backtracking the values in $\check{y}$ are restored by the solver.

```
1. init()
2.     for $i := 1$ to $n$ loop
3.         apply($i$);
4.     end loop;
5.     for $j := 1$ to $m$ loop
6.         offer($j$);
7.     end loop;
```

Fig. 4. Method *init*.

- *post* : an $m \times c$ matrix of reversible integer variables which stores applications for hospital posts. Each array element is initialised to ∞ (i.e. the largest integer). Row $post_j$ stores the applications for hospital h_j and entry $post_{j,k}$ stores the k^{th} best application received by hospital h_j.

To implement a constraint we require two methods: one that is called at the head of search to initialise the constraint and one that is called when a value is removed from a constrained variable. We now give the first of these methods:

The *init* method (Figure 4) is called at the head of search. Each resident applies to their favourite hospital (lines 2-4) via the *apply*(i) function (details given later), then each hospital makes an offer to their c favourite residents (lines 5-7) via the *offer*(j) function (details given later).

As HRN constrains two sets of variables we require two different method to call when a value is removed from one of the variable's domains. These methods are given below:

The *deltaX* method, shown in Figure 5(a), is called when some value a, where $a < m + 1$, is removed from $dom(x_i)$. The index j of the hospital a represents is found (line 2), and r_i is then removed from the domain of h_j (line 3). If a represents the last hospital r_i applied to (line 4), then r_i will make a new application to its new favourite via the *apply*(i) function (line 5). Note that either the deletion on line 3 or an indirect deletion via a call to the *apply*(i) function (details given later) could cause a reduction in the domain of some y variable and thus a call to *deltaY* will be placed on the call stack.

The *deltaY* method, shown in Figure 5(b), is called when some value a, where $a < n + 1$, is removed from $dom(y_j)$. The index i of the resident a represents is found (line 2) and h_j is then removed from the domain of r_i (line 3). If a represents a resident h_j that made an offer to (line 4), then h_j will make a new set of offers via the *offer*(j) function (line 5). Note that either the deletion on line 3 or an indirect deletion via a call to the *offer*(j) function (details given later), could cause a reduction in the domain of some x variable and thus a call

```
1. deltaX($i$,$a$)
2.     $j := PL(r_i, a)$;
3.     remValue($y_j$,rank($h_j, r_i$));
4.     if $a = \check{x}_i$ then
5.         apply($i$);
```

```
1. deltaY($j$,$a$)
2.     $i := PL(h_j, a)$;
3.     remValue($x_i$,rank($r_i, h_j$));
4.     if $a \leq \check{y}_j$ then
5.         offer($j$);
```

Fig. 5. (a) Method *deltaX*. (b) Method *deltaY*.

```
1. apply($i$)
2. for $k := \check{x}_i + 1$ to min($x_i$) loop
3.    $j := PL(r_i, k)$;
4.    apply($j$,rank($h_j, r_i$));
5.    if $post_{j,c_j} < \infty$ then
6.       setMax($y_j$,$post_{j,c_j}$);
7. end loop;
8. $\check{x}_i :=$ min($x_i$);
```

```
1. apply($j$,$a$)
2. for $k := 1$ to $c_j$ loop
3.    if $post_{j,k} = a$ then
4.       $a := n + 1$;
5.    if $post_{j,k} > a$ then
6.       swap($post_{j,k}, a$);
7. end loop;
```

Fig. 6. (a) Function *apply*(i). (b) Function *apply*(j, a).

to *deltaX*. Therefore the propagation of this constraint results from the mutual recursion between methods *deltaX* and *deltaY*.

The *apply*(i) function of Figure 6(a) is called either at the head of search (via the *init* method) or when the lower bound of x_i changes (via the *deltaX* method). Resident r_i will apply to each hospital that it prefers to any other in its domain, and to which it has not previously applied to (line 2). First the hospital h_j to be applied to is found (line 3), then resident r_i makes an application to hospital h_j via a call to the *apply*(j, a) function(line 4). If c_y applications have been made to hospital h_j (line 5) then h_j must not consider any resident worse then its c_j^{th} favourite applicant (line 6). $\check{x}_i$ is then updated with the current lower bound of x_i (line 8). As the runtime of this function is dependent on the number of domain reductions made since the previous call to this function, it therefore has $O(1)$ complexity per deletion.

The *apply*(j, a) function of Figure 6(b) is called only by the *apply*(i) function when hospital h_j receives an application from its a^{th} choice resident. The hospital's preference for this applicant is placed in the list of applicants in ascending order. If more than c_j applications have been received then the worst applicant will drop off the end of the array and will effectively be removed from the list. This function runs in $O(c)$ time.

Figure 7 gives the *offer*(j) function which can be called either at the head of search (via the *init* method) or when a resident that was previously offered a place has been removed from *dom*(y_j) (via the *deltaX* method). Hospital h_j will offer a post to r_i, the c_j^{th} favourite resident still in its domain, and to all other residents that it prefers to r_i to which it has not yet offered a place to. $\check{y}_j$ is then updated to its preference for r_i. This function contains one loop which cycles at most c_j times, therefore it runs in $O(c)$ time.

```
1. offer($j$)
2. for $k := \check{y}_i + 1$ to getValue($h_j$,$c_j$) loop
3.    $i :=$ PL($h_j, k$);
4.    setMax($x_i$,rank($r_i, h_j$))
5. end loop;
6. $\check{y}_j :=$ getValue($h_j$,$c_j$);
```

Fig. 7. Function *offer*(j).

Model:	Cloned	CBM	HRN
Time:	$O((n+m)^4\,c)$	$O(Lc(n+m))$	$O(Lc)$
Space:	$O((nmc)^2\,)$	$O(Lc)$	$O(nm)$

Table 1. Summary of time and space complexities for the HR models of this paper.

5.2 Complexity

The *deltaX* and *deltaY* methods contains no loops, but each calls a function which runs in $O(c)$ time. Thus *deltaX* and *deltaY* both have a complexity of $O(c)$. The *deltaX* method can be called at most once for each value in the domain of an x_i variable, and similarly *deltaY* can be called at most once for each value in the domain of the y_j variable. Therefore we have a time complexity of $O(Lc)$. Hence the time complexity for the HRN constraint improves those of the models presented in earlier sections. The space complexity of this encoding is dominated by the ranking arrays, and is $O(nm)$. However, if preference lists are short we may economically trade time for space, or use some sparse data structure, or a hash table to map preferences to indices.

Table 1 summarises the time and space complexities for the HR models in this paper (the columns refer respectively to the models in Sections 3, 4 and 5).

5.3 Searching for all solutions

Arc consistency processing on the HRN constraint yields the *GS-domains* as defined in Section 4. A search process need only consider the resident variables (and need not instantiate the hospital variables), following a similar process to that outlined in Theorem 4.

6 Computational experience

The three encodings presented in this paper were implemented using the JSolver toolkit, i.e. the Java version of ILOG Solver, in order to carry out an empirical analysis. The objective was to compare the runtimes for these models as applied to randomly-generated and real-world data. Our studies were carried out using a 2.8Ghz Pentium 4 processor with 512 Mb of RAM, running Microsoft Windows XP Professional and Java2 SDK 1.4.2.6 with an increased heap size of 512 Mb.

Random problem instances were generated with varying number of residents n, number of hospitals m, capacity c (uniform for each hospital), and a fixed residents' preference list size of 10. Hence we classify problems via the triple $n/m/c$. Instances were generated as follows. First, a uniformly random preference list of length 10 was produced for each resident, then a preference list was produced for each hospital by randomly permuting their acceptable residents. A sample size of 100 was used for each value of $n/m/c$.

Table 2 shows the mean time in seconds to construct the model and find all solutions, for the each of the four models applied to random instances with varying $n/m/c$ attributes. A table entry of $-$ signifies that there was insufficient

	50/13/4	100/20/5	500/63/8	1k/100/10	5k/250/20	20k/550/37	50k/1.2k/42
Cloned	5.84	–	–	–	–	–	–
CBM	0.24	0.36	1.69	4.75	–	–	–
HRN	0.12	0.15	0.19	0.22	0.53	1.42	4.2

Table 2. Average computation times in seconds to find all solutions to 100 randomly-generated HR instances with attributes $n/m/c$.

space to create the model of that size using the specified encoding. Table 3 shows the time to establish AC (shown as "AC") and find all solutions (shown as "ALL") to three anonymised HR instances arising from SFAS [13]. The first column indicates $n/m/c$, where c is the average hospital capacity; also $l_i^r \leq 5$ in each case. (For each instance, the Cloned model ran out of memory.)

The results indicate that the HRN model was typically able to handle larger problem instances than the other models, and the average runtime was faster than for the other models in all cases. The HRN model was also applied to instances as large as $500k/11.8k/85$, finding all solutions on average in 35 seconds. As mentioned in the Introduction, instances of the NRMP typically involve around 31,000 residents and 2,300 hospitals, with residents' preference lists of size between 4 and 7 [23]. The HRN model finds all solutions to problems of size $200k/3k/67$ in 22 seconds on average. This leads us to believe that Constraint Programming is indeed a suitable technology for the HR problem.

7 Motivation: side-constraints

It is natural to build additional constraints on top of the constraint models of HR presented in this paper, in order to cope with generalisations of HR for which the RGS and HGS algorithms are inapplicable. In this section we present several variants of HR that are either NP-hard or for which no polynomial-time algorithm is currently known. In the first three cases we suggest additional side-constraints that can be added to any of our base models in order to cope with the more general problem, providing additional motivation for our approach.

Resident-exchange-stable HR. During a previous run of the SFAS matching scheme, two residents complained that, had they swapped their given hospitals, they could each have been better off. Such a swap would not have been permitted by the hospitals, of course, as it would have violated the stability criterion. However it would be desirable to avoid such a situation arising if possible, and this leads to the problem of finding a *resident-exchange stable matching* given

	# Solutions	CBM		HRN	
		AC	ALL	AC	ALL
502/41/13.2	1	1.61	1.64	0.17	0.17
510/43/11.5	1	1.64	1.7	0.17	0.17
245/34/3.9	1	0.26	0.26	0.12	0.12

Table 3. Time taken to establish AC and find all solutions to three SFAS instances.

an instance I of HR. This is a stable matching M in I such that there are no two assigned residents r_i, r_j such that r_i prefers $M(r_j)$ to $M(r_i)$, and r_j prefers $M(r_i)$ to $M(r_j)$. It is known that a such a matching need not exist in I, and indeed the problem of deciding whether such a matching exists in I is NP-complete [14, 21], even if each hospital has capacity 1. For any two residents r_i, r_j and for any two hospitals h_k, h_l such that r_i prefers h_l to h_k and r_j prefers h_l to h_k, the additional constraint $x_i = p_1 \Rightarrow x_j \neq p_2$ should be added, where $rank(r_i, h_k) = p_1$ and $rank(r_j, h_l) = p_2$.

HR with forbidden pairs. Let F be a set of (resident,hospital) pairs in an instance I of HR. An adminstrator of a matching scheme may wish to exclude the pairs in F from any matching. Hence a matching M in I must not include any member of F, however a pair in F could still form a blocking pair (hence we cannot simply delete pairs in F from the preference lists). The task is to find a matching in I that is stable in the usual sense. Clearly a stable matching need not exist, given an instance of HR with forbidden pairs. However given an instance of SMI with forbidden pairs, there exists a linear-time algorithm to find a stable matching or report that none exists [6], and it is straightforward to extend this algorithm to HR. However no polynomial-time algorithm is currently known for the problem of finding a matching M in I (in the usual sense) with the fewest number of forbidden pairs. One possibility for modelling this problem is to add new variables $T = \{t_{i,p} : 1 \leq i \leq n \wedge 1 \leq p \leq l_i^r\}$, each with domain $\{0, 1\}$, and a constraint $x_i = p \Rightarrow t_{i,p} = 1$, for each $(r_i, h_j) \in F$, where $rank(r_i, h_j) = p$, and then minimise the sum of the values of the variables in T.

HR with groups. An extension of HR that has practical relevance arises when residents may form groups, and may decide that they are only prepared to be matched to a given hospital if the whole group is matched to it. More formally, each hospital $h_j \in H$ may have one or more associated groups $G_j \subseteq R$. A matching M must satisfy the additional property that if $(r_i, h_j) \in M$ for some $r_i \in G_j$, then $(r_k, h_j) \in M$ for all $r_k \in G_j$. No polynomial-time algorithm for this problem is currently known. However this variant can be modelled as follows. For any group $G_j = \{r_{i_1}, \ldots, r_{i_k}\}$, add the constraint $x_{i_a} = p_{i_a} \Rightarrow x_{i_b} = p_{i_b}$ $(1 \leq a \neq b \leq k)$ where $rank(r_{i_a}, h_j) = p_{i_a}$ and $rank(r_{i_b}, h_j) = p_{i_b}$. A particular case of this problem is the Hospitals / Residents problem with Couples (HRC), described below.

Other generalisations of HR. The Hospitals / Residents problem with Ties (HRT) arises when ties are permitted in the preference lists of hospitals and/or residents. For example, a popular hospital may be indifferent among several applicants. The SFAS scheme [13] already permits ties in the hospitals' lists. However it is known [18] that, in the presence of ties, stable matchings can be of different sizes, and the problem of finding a maximum stable matching is NP-hard, even for very restricted instances of SMI with ties. It has already been demonstrated [9, 10] that the earlier encodings of [8] can be extended to the case where preference lists in a given SMI instance may involve ties. We have begun to consider the corresponding extension of the models presented in Sections 4 and 5 to the HRT case, and further details will appear elsewhere.

HRC (in which couples submit joint preference lists over pairs of hospitals) is another generalisation of HR. Again it is possible that an instance need not admit a stable matching (where the stability definition is extended to the couples case), and the problem of deciding whether such a matching exists is NP-complete [24]. A constraint-based solution to this problem is motivated by the NRMP, which permits couples to submit joint preference lists.

8 Conclusions and future work

In this paper we have presented three CP models of an HR instance. The empirical results for the models as presented in Section 6 are broadly in line with what may be expected, given the summary of time and space complexities presented in Table 1. Our results indicate that, as is the case for SMI [8], CSP encodings of HR are "tractable", a notion that has been explored in detail by Green and Cohen [11]. However it remains open as to whether there exists a CSP encoding of HR that gives rise to the GS-lists, for which AC may be established in $O(L)$ time and using $O(nm)$ space. The time complexity of $O(L)$ is optimal, since SM is a special case of HR, and a lower bound of $\Omega(L)$ holds for the problem of finding a stable matching, given an instance of SM [22].

Acknowledgement

The authors are grateful to ILOG SA for providing access to the JSolver toolkit via an Academic Grant Licence.

References

1. B. Aldershof, O.M. Carducci, and D.C. Lorenc. Refined inequalities for stable marriage. *Constraints*, 4:281–292, 1999.
2. C. Bessière and J-C. Régin. Arc consistency for general constraint networks: Preliminary results. In *Proceedings of IJCAI '97*, vol. 1, pp. 398–404, 1997.
3. I. Brito and P. Meseguer. Distributed stable matching problems. In *Proceedings of CP '05*, *LNCS* vol. 3705, pp. 152–166. Springer, 2005.
4. I. Brito and P. Meseguer. Distributed stable matching problems with ties and incomplete lists. In *Proceedings of CP '06*, *LNCS* vol. 4204, pp. 675–679. Springer, 2006.
5. Canadian Resident Matching Service. How the matching algorithm works. Web document available at `http://www.carms.ca/matching/algorith.htm`.
6. V.M.F. Dias, G.D. da Fonseca, C.M.H. de Figueiredo and J.L. Szwarcfiter. The stable marriage problem with restricted pairs. *Theoretical Computer Science*, 306(1-3):391–405, 2003.
7. D. Gale and L.S. Shapley. College admissions and the stability of marriage. *American Mathematical Monthly*, 69:9–15, 1962.
8. I.P. Gent, R.W. Irving, D.F. Manlove, P. Prosser, and B.M. Smith. A constraint programming approach to the stable marriage problem. In *Proceedings of CP '01*, *LNCS* vol. 2239, pp. 225–239. Springer, 2001.
9. I.P. Gent and P. Prosser. An empirical study of the stable marriage problem with ties and incomplete lists. In *Proceedings of ECAI '02*, pp. 141–145. IOS Press, 2002.

10. I.P. Gent and P. Prosser. SAT encodings of the stable marriage problem with ties and incomplete lists. In *Proceedings of SAT '02*, pp. 133–140, 2002.
11. M.J. Green and D.A. Cohen. Tractability by approximating constraint languages. In *Proceedings of CP '03*, *LNCS* vol. 2833, pp. 392–406. Springer, 2003.
12. D. Gusfield and R.W. Irving. *The Stable Marriage Problem: Structure and Algorithms.* MIT Press, 1989.
13. R.W. Irving. Matching medical students to pairs of hospitals: a new variation on a well-known theme. In *Proceedings of ESA '98*, *LNCS* vol. 1461, pp. 381–392. Springer, 1998.
14. R.W. Irving The Man-Exchange Stable Marriage Problem. Technical Report TR-2004-177, University of Glasgow, Department of Computing Science, 2004.
15. D.E. Knuth. Mariages Stables *Les Presses de L'Université de Montréal*, 1976.
16. I.J. Lustig and J. Puget. Program does not equal program: constraint programming and its relationship to mathematical programming. *Interfaces*, 31:29–53, 2001.
17. A.K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8:99–118, 1977.
18. D.F. Manlove, R.W. Irving, K. Iwama, S. Miyazaki, and Y. Morita. Hard variants of stable marriage. *Theoretical Computer Science*, 276 (1-2) : 261–279, 2002.
19. D.F. Manlove and G. O'Malley. Modelling and solving the stable marriage problem using constraint programming. In *Proceedings of the Fifth Workshop on Modelling and Solving Problems with Constraints*, held at IJCAI '05, pp. 10–17, 2005.
20. D.F. Manlove, G. O'Malley, P. Prosser and C. Unsworth. A Constraint Programming Approach to the Hospitals / Residents Problem. Technical Report TR-2007-236, University of Glasgow, Department of Computing Science, 2007.
21. E. McDermid, C. Cheng and I. Suzuki. Hardness results on the man-exchange stable marriage problem with short preference lists. *Information Processing Letters*, 101:13–19, 2007.
22. C. Ng and D.S. Hirschberg. Lower bounds for the stable marriage problem and its variants. *SIAM Journal on Computing*, 19:71–77, 1990.
23. National Resident Matching Program. About the NRMP. Web document available at `http://www.nrmp.org/about_nrmp/how.html`.
24. E. Ronn. NP-complete stable matching problems. *Journal of Algorithms*, 11:285–304, 1990.
25. A.E. Roth. The evolution of the labor market for medical interns and residents: a case study in game theory. *Journal of Political Economy*, 92(6):991–1016, 1984.
26. A.E. Roth and M.A.O. Sotomayor. *Two-sided matching: a study in game-theoretic modeling and analysis.* Cambridge University Press, 1990.
27. M.-C. Silaghi, M. Zanker and R. Barták. Desk-mates (stable matching) with privacy of preferences, and a new distributed CSP framework. In *Proceedings of the CP 2004 workshop on CSP Techniques with Immediate Application (CSPIA)*, pp. 83–96, 2004.
28. M.-C. Silaghi, A. Abhyankar, M. Zanker and R. Barták. Desk-mates (stable matching) with privacy of preferences, and a new distributed CSP framework. In *Proceedings of FLAIRS 2005*, pp. 671-677. AIII Press, 2005.
29. C. Unsworth and P. Prosser. An n-ary constraint for the stable marriage problem. In *Proceedings of the Fifth Workshop on Modelling and Solving Problems with Constraints*, held at IJCAI '05, pp. 32–38, 2005.
30. C. Unsworth and P. Prosser. A specialised binary constraint for the stable marriage problem. In *Proceedings of SARA '05*, *LNAI* vol. 3607, pp. 218-233. Springer, 2005.
31. P. van Hentenryck, Y. Deville, and C-M. Teng. A generic arc-consistency algorithm and its specializations. *Artificial Intelligence*, 57:291–321, 1992.