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, Scotland.
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. Our study suggests that Constraint Programming is indeed an applicable technology for solving this problem, in terms of both theory and practice.

1 Introduction

Gale and Shapley described in their seminal paper [4] 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 [4] 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 [22]. For example, the National Resident Matching Program (NRMP) in the US [20] 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. Counterparts of the NRMP elsewhere are the Canadian Resident Matching Service

⋆ Supported by RSE / Scottish Executive Personal Research Fellowship.
† Supported by EPSRC grant GR/R84597/01.
‡ Supported by an EPSRC studentship.

(CaRMS) [3] and the Scottish PRHO Allocation scheme (SPA) [11]. 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 [4]. A specialised linear-time algorithm for SM, known as the Gale / Shapley (GS) algorithm [4], can be generalised to the SMI case [10, 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 [10, 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 [4, 13, 10, 24]. Several encodings of SM and its variants as a Constraint Satisfaction Problem (CSP) have been formulated [1, 6, 14, 7–9, 17, 25, 26]. 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 [6] 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 [4, 10]. 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 [6] for SMI. In Sections 5 and 6, we present specialised binary and n-ary constraints 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 7, 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. Finally, Section 8 presents some concluding remarks, and discusses future work.

The models in Sections 4-6 are non-trivial extensions of earlier constraint models presented for SMI [6, 17, 25, 26]. In the SMI case, clearly each woman can be assigned at most one man, but to model an HR instance without cloning, the 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

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).

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. Examples of such variants include the Hospitals / Residents problem with Ties (in which preference lists may include ties; see Section 8 for more details), the Hospitals / Residents problem with Couples (in which couples submit joint preference lists), and the generalisation of the Sex-Equal Stable Marriage problem (in which one seeks a stable matching such that the sums of the ranks of the men's and women's partners are as close as possible) to the HR case. All of these variants are known to be NP-hard [16, 21, 12].

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 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 [4] 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 [10, 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 [10, 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 [10]). 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 [10, 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 [10]):

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;*

```
M = ∅;
while (some r_i ∈ 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 ∪ {(r_i, h_j)} ;
  if (h_j is over-subscribed)
    r_k = worst resident assigned to h_j;
    M = M\{(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 = ∅;
while (some h_j ∈ H is under-subscribed
  and some r_i ∈ 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\{(r_i, h_k)};
  M = M ∪ {(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.

(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, attributed to Gale and Sotomayor [5] and Roth [23], 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* [10, 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 [10, p.38]; see also [24, 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 $(1 \le k \le c_j)$. 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 [6]. 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

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.

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 [6], 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 $O((n+m)^4 c)$ and $O((nmc)^2)$ respectively.

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 [18] (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 [27] 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 [18] (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 binary constraint

We now present a specialised binary constraint HR2 that acts between an integer variable, representing a resident, and an object of type *Hospital*, enforcing stability and consistency. The model of this section involves an HR2 constraint between each acceptable (resident, hospital) pair.

5.1 Preliminaries

Our model involves a constrained integer variable x_i corresponding to each resident $r_i \in R$, as in Section 4, whose domain is initially defined as before, with similar meanings for the domain values. In addition, we associate a *Hospital* object y_j with each hospital $h_j \in H$, with the following attributes:

- *cap* : an integer constant equal to c_j (the capacity of hospital h_j).
- *post* : an array of integers of length *cap*, which stores assignments to hospital posts. Each array element is initialised to ∞ (i.e. the largest integer).
- $pref$: a constrained integer variable whose initial domain is $\{1, 2, \ldots, l_j^h\}$ (corresponding to the ranks of residents in h_j's list), plus the value $n + 1$ (corresponding to h_j being under-subscribed).

We also 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 $dom(v)$.
- $getNext(v, a)$ returns the smallest value greater than a in $dom(v)$, assuming that $a < getMax(v)$, otherwise the function returns a.
- $setMax(v, a)$ sets the maximum value in $dom(v)$ to be $min(getMax(v), a)$.
- $remVal(v, a)$ removes the value a from $dom(v)$.

We assume that constraints are processed by an arc consistency algorithm such as AC5 [27] or AC3 [15]. That is, the algorithm has a stack of constraints that are awaiting revision, and if a variable v loses a value then all constraints involving v are added to the stack along with the method that must be applied to those

```
1. xAppliesTo(y,yRx) {
2. r = yRx;
3. for (i = 1 to y.cap)
4.   if (y.post[i] = r)
5.     return;
6.   if (y.post[i] > r)
7.     swap(y.post[i],r);
8. if (y.post[y.cap] < ∞)
9.   setMax(y.pref,y.post[y.cap]); }
```

```
1. getLastChoice(y) {
2. choice = getMin(y.pref);
3. for (i = 2 to y.cap)
4.   choice = getNext(y.pref,choice);
5. return choice; }
```

Fig. 4. (a) Method *xAppliesTo*. (b) Method *getLastChoice*.

constraints (so that the stack contains methods and their arguments). Furthermore, we also assume that a call to a method, together with its argument, is only added to the stack if it is not already on the stack. In our pseudocode below we use the . (dot) operator as an attribute selector, such that $a.b$ delivers the b attribute of a.

The *xAppliesTo* method of Figure 4(a) is called when a resident r_i (represented by variable x) applies to a hospital h_j (represented by object y). In the pseudocode we assume that yRx represents $rank(h_j, r_i)$. The method stores all assignments involving hospital h_j in strict preference order, with the most-preferred resident in $y.post[1]$. The method loops through each element of the $y.post$ array (lines 3 to 7). If r_i is already in the list of h_j's assignees then no action is taken (lines 4 and 5). If the current value of r (which is initially $rank(h_j, r_i)$) is less then the value in $y.post[i]$ (line 6), then the value in r is swapped with the value in $y.post[i]$ (line 7) and the loop continues, so that the value of r is inserted in order into the $y.post$ array. On termination of the loop, if the last element of $y.post$ has been assigned a value (line 8), then h_j is assigned c_j residents, consequently we can set the maximum value of $y.pref$ (line 9). This method contains only one loop which iterates c_j times, and all methods used are of $O(1)$ complexity. Hence the complexity of *xAppliesTo* is $O(c)$.

A hospital h_j (represented by object y) offers a post to a resident r_i (represented by variable x) if r_i occupies one of the first c_j undeleted entries in h_j's preference list. Correspondingly, y offers a post to x if $rank(h_j, r_i)$ is one of the first $y.cap$ values in $dom(y.pref)$. To test for this condition we use the *getLastChoice* method of Figure 4(b), which returns h_j's rank of the worst resident that it can currently offer a post to. Firstly the lowest value in $dom(y.pref)$ is found (line 2). The loop then iterates to find the r^{th}-largest rank in $dom(y.pref)$, where $r = y.cap$ (lines 3 and 4). This value is then returned via variable choice (line 5). The time complexity of this method is again $O(c)$.

5.2 The HR2 constraint

A binary Hospitals / Residents constraint (HR2) is an object that acts between a variable x (representing a resident $r_i \in R$) and an object y (representing a hospital $h_j \in H$), and has attributes x, y, xRy and yRx. Here, yRx is as above (representing $rank(h_j, r_i)$), whilst xRy represents $rank(r_i, h_j)$.

```
1. deltaX(C) {
2. if getMin(C.x) = C.xRy
3.    xAppliesTo(C.y,C.yRx);
4. if getMax(C.x) < C.xRy
5.    remVal(C.y.pref,C.yRx); }
```

```
1. deltaY(C) {
2. if C.yRx ≤ getLastChoice(C.y)
3.    setMax(C.x,C.xRy);
4. if getMax(C.y.pref) < C.yRx
5.    remVal(C.x,C.xRy); }
```

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

Therefore a constraint C between x_i and y_i is constructed via a call to the function $C = HR2(x_i, rank(r_i, h_j), y_j, rank(h_j, r_i))$. This will construct a constraint C such that $C.x = x_i$, $C.y = y_j$, $C.xRy = rank(r_i, h_j)$ and $C.yRx = rank(h_j, r_i)$. To construct our encoding we would then make calls to HR2 for all i and j where r_i and h_j find each other acceptable, thus creating $O(nm)$ constraints.

Three methods, $deltaX$, $deltaY$, and $init$, act on a constraint C and achieve arc consistency between a resident x and hospital y across C. The $deltaX$ method, shown in Figure 5(a), is called when a value is removed from $dom(x)$. If r_i's most-preferred undeleted hospital is h_j (line 2) then r_i applies to h_j (line 3). In the call to $xAppliesTo$, r_i becomes assigned to h_j if the assignment has not already been made (line 7 of $xAppliesTo$), and if h_j is now full, then the tail of h_j's preference list is cropped (line 9 of $xAppliesTo$), and this will in turn generate a call to $deltaY$ (described below). If r_i prefers his worst undeleted hospital to h_j (line 4), then h_j has been deleted from r_i's preference list, and consequently r_i is deleted from h_j's list (line 5) – this in turn will generate a call to $deltaY$, which is now described.

The $deltaY$ method, shown in Figure 5(b), is called when a value is removed from $dom(y.pref)$. If resident r_i is among the first c_j undeleted residents on h_j's preference list (line 2), then r_i need consider no hospital that it finds inferior to h_j (line 3). This action may delete values from the domain of x and subsequently generate calls to $deltaX$. If h_j prefers its worst undeleted resident to r_i (line 4), then r_i has been deleted from h_j's preference list, and consequently h_j is deleted from r_i's list (line 5). This may then generate calls to $deltaX$. Note also that lines 4 and 5 of $deltaY$ are symmetrical to lines 4 and 5 in $deltaX$.

Finally, the $init(C)$ method is called to start the process of making constraint C arc consistent, and makes calls to the $deltaX(C)$ and $deltaY(C)$ methods.

5.3 Complexity

The $deltaX$ method has no loops and thus its time complexity is that of the most complex method it calls, which is the $xAppliesTo$ method with a complexity of $O(c)$, consequently $deltaX$ has a complexity of $O(c)$. Similarly the complexity of the $deltaY$ method is that of the most complex method it calls, which is $getLastChoice$, with a complexity of $O(c)$. Both of the methods called by $init$ thus have a time complexity of $O(c)$, and hence $init$'s complexity is also $O(c)$.

Each HR2 constraint C has three methods. The $init(C)$ method will be called only once and is of complexity $O(c)$. The $deltaX(C)$ method can at worst be called once for each value in the domain of $C.x$. As the maximum length of a resident's preference list is m, and $deltaX(C)$ has a complexity of $O(c)$, the combined worst case complexity of all possible calls to $deltaX(C)$ is $O(mc)$. Similarly $deltaY(C)$ can at worst be called once for each of the n possible values in the domain of $C.y.pref$. As $deltaY(C)$ has a complexity of $O(c)$, the combined worst case complexity of all possible calls to $deltaY(C)$ is $O(nc)$. Therefore the overall worst case time complexity for a single constraint is $O(c(m+n))$, and as there are L of the HR2 constraints, the overall time complexity of enforcing arc consistency on this model is $O(Lc(n+m))$, which is the same as the time complexity for the model of Section 4. Furthermore, as there are $O(nm)$ HR2 constraints, each of size $O(1)$, the space complexity of a model using the HR2 constraint is $O(nm)$.

6 A specialised n-ary constraint

We now present a specialised n-ary constraint HRN for the Hospitals / Residents problem. This constraint acts between an array of integer variables, $x[1], \ldots, x[n]$, representing the residents (as before), and an array of objects of type *Hospital*, $y[1], \ldots, y[m]$, representing the hospitals (again, as before). (Strictly speaking the arity of the HRN constraint is $n+m$, but for simplicity we refer to it as an n-ary constraint.) A model based on HRN requires only one constraint for the whole problem. Henceforth we assume that we have access to the hospital class and all the same functions as with the binary constraint defined in Section 5.

6.1 The Constraint

An n-ary Hospitals / Residents constraint (HRN) is an object that acts between an array of residents and an array of hospitals, and has the following attributes:

- x is an array of constrained integer variables representing the residents, such that resident $r_i \in R$ is represented by $x[i]$.
- y is an array of objects of type *Hospital* representing the hospitals, such that hospital $h_j \in H$ is represented by $y[j]$.
- xRy is an $n \times m$ integer array such that $xRy[i][j] = rank(r_i, h_j)$ if r_i finds h_j acceptable, and is 0 otherwise.
- yRx is an $m \times n$ integer array such that $yRx[j][i] = rank(h_j, r_i)$ if h_j finds r_i acceptable, and is 0 otherwise.
- xpl is an $n \times m$ integer array such that, for each i $(1 \leq i \leq n)$ and k $(1 \leq k \leq l_i^r)$, $xpl[i][k] = j$ if and only if $rank(r_i, h_j) = k$.
- ypl is an $m \times n$ integer array such that, for each j $(1 \leq j \leq m)$ and k $(1 \leq k \leq l_j^h)$, $ypl[j][k] = i$ if and only if $rank(h_j, r_i) = k$.

Again, we have three methods that act on an n-ary constraint C, namely $deltaX$, $deltaY$ and $init$. The $deltaX$ method, shown in Figure 6(a), is called when a value a, where $a < m+1$, is removed from $dom(x[i])$. If a is the rank of a hospital h_k that r_i prefers to his most-preferred undeleted hospital (line 2) (i.e. r_i has

```
1.  deltaX(C,i,a) {
2.  if (a < getMin(C.x[i]))
3.     k = getMin(C.x[i]);
4.     j = C.xpl[i][k];
5.     xAppliesTo(C.y[j],C.yRx[j][i]);
6.  else
7.     j = C.xpl[i][a];
8.     remVal(C.y[j].pref,C.yRx[j][i]) }
```

```
1.  deltaY(C,j,a) {
2.  if (a > getMax(C.y[j].pref))
3.     i = C.ypl[j][a];
4.     remVal(C.x[i],C.xRy[i][j]);
5.  else
6.     k = getMin(C.y[j].pref);
7.     for (z=1 to C.y[j].cap)
8.        i = C.ypl[j][k];
9.        setMax(C.x[i],C.xRy[i][j])
10.       k = getNext(C.y[j].pref,k) }
```

Fig. 6. (a) Method *deltaX*. (b) Method *deltaY*.

been rejected by h_k), the index j of r_i's new favourite hospital is found (lines 3 and 4) and r_i applies to h_j (line 5). This may result in a subsequent call to *deltaY* via the *xAppliesTo* method. If the rank of r_i's most-preferred undeleted hospital is not larger than a, the hospital h_j at position a of r_i's list is found (line 7), and r_i is deleted from h_j's preference list (line 8). This will generate a call to $deltaY(C, j, C.yRx[j][i])$, which is now described.

The *deltaY* method, shown in Figure 6(b), is called when a value a, where $a < n + 1$, is removed from $dom(y[j].pref)$. If the removed value a is larger than the rank of h_j's worst undeleted resident (line 2), then the resident r_i at position a of h_j's list is found (line 3), and h_j is deleted from r_i's preference list (line 4). This will in turn generate a call to $deltaX(C, i, C.xRy[i][j])$. If a is not larger than the rank of h_j's worst undeleted resident (line 5), then h_j will offer a post to the first c_j undeleted residents on its list (lines 6 to 10). Lines 6 and 8 identify the most-preferred undeleted resident r_i and his corresponding rank k in h_j's list. All hospitals inferior to h_j are then deleted from r_i's list (line 9). We then identify the next undeleted resident in h_j's list (line 10) whilst respecting h_j's capacity (controlled by the loop condition in line 7). Essentially, lines 6 to 10 reconstruct the offers from hospital h_j following the removal of a from $dom(y[j].pref)$. Note that the call to *setMax* in line 9 may in turn generate calls to *deltaX*. Therefore the propagation of this constraint results from the mutual recursion between methods *deltaX* and *deltaY*.

Finally the *init* method makes calls to $deltaX(C, i, 0)$ for all i ($1 \leq i \leq n$), and $deltaY(C, j, 0)$ for all j ($1 \leq j \leq m$).

6.2 Complexity

The *deltaX* method of this section contains no loops, but calls the *xAppliesTo*() method which has a complexity of $O(c)$, and thus *deltaX* also has a complexity of $O(c)$. The *deltaY* method contains only one loop, which iterates c_j times, and all methods used run in $O(1)$ time. Therefore the time complexity of *deltaY* is also $O(c)$. 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 *pref* attribute of a $y[j]$ variable. Therefore

Model:	Cloned	CBM	HR2	HRN
Time:	$O((n+m)^4 c)$	$O(Lc(n+m))$	$O(Lc(n+m))$	$O(Lc)$
Space:	$O((nmc)^2)$	$O(Lc)$	$O(nm)$	$O(nm)$

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

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 xRy and yRx, and is $O(nm)$, though comparable to that of the model presented in Section 5. 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, 5 and 6).

6.3 Searching for all solutions, using HR2 or HRN

Arc consistency processing on the HR2 and HRN constraints 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. Because the search process will backtrack, the variable $y[j].post$ would need to be reversible, in order that values corresponding to assignment information can be restored on backtracking.

Until now we have assumed that values are removed only as a result of arc consistency processing. This is not true with the backtracking search. Consequently we require minor modifications to our methods. For the HR2 constraint the *deltaX* method needs to consider the case when $C.xRy < getMin(C.x)$, i.e. r_i prefers h_j to each undeleted hospital on his preference list. Therefore to prevent (r_i, h_j) being a blocking pair, h_j must be full and must prefer its worst resident to r_i, i.e. we then make a call to $setMax(C.y, C.yRx - 1)$.

For the n-ary constraint HRN, *deltaX* must consider the case where the deleted value a is less than the smallest remaining value in the domain of $C.x[i]$, i.e. $a < getMin(C.x[i])$. Therefore again, to prevent (r_i, h_j) being a blocking pair (where $j = C.xpl[i][a]$), we make the call $setMax(C.y[j].pref, C.yRx[j][i] - 1)$.

7 Computational experience

The four 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.

	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	–	–	–
HR2	0.15	0.18	0.42	0.88	9.91	112	–
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 $n/m/c$.

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 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 SPA [11]. 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 [20]. 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.

	# Solutions	CBM		HR2		HRN	
		AC	ALL	AC	ALL	AC	ALL
502/41/13.2	1	1.61	1.64	0.26	0.28	0.17	0.17
510/43/11.5	1	1.64	1.7	0.27	0.31	0.17	0.17
245/34/3.9	1	0.26	0.26	0.14	0.16	0.12	0.12

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

8 Conclusions and future work

In this paper we have presented four CP models of an HR instance. The empirical results for the models as presented in Section 7 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 [6], CSP encodings of HR are "tractable", a notion that has been explored in detail by Green and Cohen [9]. 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 [19].

The natural extension of this work is to build additional constraints on top of one of the models presented here, in order to cope with generalisations of HR for which the RGS and HGS algorithms are inapplicable. Section 1 described three possible variants of HR that are relevant in this context. One of these was the Hospitals / Residents problem with Ties (HRT), which 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 SPA scheme [11] already permits ties in the hospitals' lists. However it is known [16] 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 [7, 8] that the earlier encodings of [6] 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, 5 and 6 to the HRT case, and further details will appear elsewhere.

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. Morgan Kaufmann, 1997.
3. Canadian Resident Matching Service. How the matching algorithm works. Web document available at `http://www.carms.ca/matching/algorith.htm`.
4. D. Gale and L.S. Shapley. College admissions and the stability of marriage. *American Mathematical Monthly*, 69:9–15, 1962.
5. D. Gale and M. Sotomayor. Some remarks on the stable matching problem. *Discrete Applied Mathematics*, 11:223–232, 1985.

6. 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.
7. 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.
8. 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.
9. 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.
10. D. Gusfield and R.W. Irving. *The Stable Marriage Problem: Structure and Algorithms*. MIT Press, 1989.
11. 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.
12. A. Kato. Complexity of the sex-equal stable marriage problem. *Japan Journal of Industrial and Applied Mathematics*, 10:1–19, 1993.
13. D.E. Knuth. Mariages Stables *Les Presses de L'Université de Montréal*, 1976.
14. I.J. Lustig and J. Puget. Program does not equal program: constraint programming and its relationship to mathematical programming. *Interfaces*, 31:29–53, 2001.
15. A.K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8:99–118, 1977.
16. 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.
17. 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.
18. D.F. Manlove, G. O'Malley, P. Prosser and C. Unsworth. A Constraint Programming Approach to the Hospitals / Residents Problem. Technical Report TR-2005-196 of the Computing Science Department of Glasgow University, 2005.
19. C. Ng and D.S. Hirschberg. Lower bounds for the stable marriage problem and its variants. *SIAM Journal on Computing*, 19:71–77, 1990.
20. National Resident Matching Program. About the NRMP. Web document available at `http://www.nrmp.org/about_nrmp/how.html`.
21. E. Ronn. NP-complete stable matching problems. *J. Algorithms*, 11:285–304, 1990.
22. 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.
23. A.E. Roth. On the allocation of residents to rural hospitals: a general property of two-sided matching markets. *Econometrica*, 54:425–427, 1986.
24. A.E. Roth and M.A.O. Sotomayor. *Two-sided matching: a study in game-theoretic modeling and analysis*. Cambridge University Press, 1990.
25. 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.
26. 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.
27. P. van Hentenryck, Y. Deville, and C-M. Teng. A generic arc-consistency algorithm and its specializations. *Artificial Intelligence*, 57:291–321, 1992.