A Specialised Binary Constraint for the Stable Marriage Problem*

Chris Unsworth[1] and Patrick Prosser[1]

Department of Computing Science, University of Glasgow, Scotland.
chrisu/pat@dcs.gla.ac.uk

Abstract. We present a specialised binary constraint for the stable marriage problem. This constraint acts between a pair of integer variables where the domains of those variables represent preferences. Our constraint enforces stability and disallows bigamy. For a stable marriage instance with n men and women we require n^2 of these constraints, and the complexity of enforcing arc-consistency is $O(n^3)$. Although this is non-optimal, empirical evidence suggests that in practical terms our encoding significantly outperforms the optimal encoding given in [7] in both space and time.

1 Introduction

In the Stable Marriage problem (SM) [6, 10] we have n men and n women. Each man ranks the n women into a preference list. So also do the women. The problem is then to produce a matching of men to women such that it is stable. By a matching we mean that there is a bijection from men to women, and by stable we mean that there is no incentive for partners to divorce and elope. A matching is unstable if there are two couples (m_i, w_j) and (m_k, w_l) such that m_i prefers w_l to his current partner w_j, and w_l prefers m_i to her current partner m_k. Stable matching problems occur naturally while matching people to posts [18], such as the allocation of residents to hospitals in the US [17], Canada [5], and Scotland [11]. Variants of the problem occur such as the allocation of groups of students to university accommodation [12], and hard variants have been identified and studied by Irving and Manlove [16]. The problem has also attracted the interest of the constraint programming community [4, 7, 9, 8, 14].

Figure 1 is an instance of the stable marriage problem, and has 6 men and 6 women. Figure 1(a) shows the problem initially, with each man and woman's preference list. Figure 1(b) shows the intersection of the male and female-oriented Gale-Shapley lists (GS-lists) [10], where the GS-lists are reduced preference lists. A man-optimal (woman-pessimal) stable matching can now be found by marrying men (women) to their most (least) preferred choices in there GS-lists. Conversely, we can produce a woman-optimal (man-pessimal) matching by marrying women (men) to their most (least) preferred choice in their GS-lists. An

* The first author is supported by EPSRC. Software support was given by an ILOG SA's academic grant.

instance of SM admits at least one stable matching and this can be found via the Extended Gale-Shapley algorithm in time $O(n^2)$, where there are n men and n women.

Men's lists	Women's lists
1: 1 3 6 2 4 5	1: 1 5 6 3 2 4
2: 4 6 1 2 5 3	2: 2 4 6 1 3 5
3: 1 4 5 3 6 2	3: 4 3 6 2 5 1
4: 6 5 3 4 2 1	4: 1 3 5 4 2 6
5: 2 3 1 4 5 6	5: 3 2 6 1 4 5
6: 3 1 2 6 5 4	6: 5 1 3 6 4 2

(a)

Men's lists	Women's lists
1: 1	1: 1
2: 2	2: 2
3: 4	3: 4 6
4: 6 5 3	4: 3
5: 5 6	5: 6 4 5
6: 3 6 5	6: 5 6 4

(b)

Fig. 1. (a) An SM instance with 6 men and 6 women; (b) the corresponding GS-lists.

We present a remarkably simple constraint encoding for the stable marriage problem. We introduce a specialised binary constraint with only three methods, where each method is no more than two lines of code. We prove that enforcing arc-consistency in this encoding results in the male-oriented Gale-Shapley lists. We then go on to show how we can extend this encoding by introducing a modest amount of additional code, such that the encoding can be embedded in richer impure problems where the stability of marriages is only part of a larger problem, and the male and female oriented GS-lists are produced. Our empirical results suggest, that although our encodings has $O(n^3)$ time complexity, it significantly outperforms the optimal encoding proposed in [7] in both space and time. In the presentation that follows we will take a one sided, male-oriented, view of the problem. Everything that is presented also has an equivalent and symmetric female-orientation.

2 The Extended Gale-Shapley Algorithm (EGS)

We now describe the male-oriented Extended Gale-Shapley (EGS) algorithm (shown in Figure 2). In particular, we explain what is meant by a *proposal*, an *engagement*, and for a man to become *free*. We will use this later to show that this algorithm and our constraint encoding are equivalent.

The EGS algorithm [10] produces a stable matching between men m_1 to m_n and women w_1 to w_n, where each man (woman) ranks each of the women (men) into preference order. Via a process of proposals from men to women the algorithm delivers reduced preference lists, called GS-lists (Gale-Shapley lists), such that if each man (woman) is paired with his (her) best (worst) partner in their GS-list the marriages will be stable.[1]

[1] Strictly speaking, the given algorithm produces MGS-lists, the male GS-lists. But for the sake of brevity we will refer to them as GS-lists.

```
 1  assign each person to be free
 2  WHILE (some man m is free)
 3  DO BEGIN
 4     w := first woman on m's list
 5     IF (some man p is engaged to w)
 6     THEN assign p to be free
 7     assign m and w to be engaged to each other
 8     FOR (each successor p of m on w's list)
 9     DO BEGIN
10        delete p from w's list
11        delete w from p's list
12        END
13     END
```

Fig. 2. The male-oriented Extended Gale/Shapley algorithm.

We will assume that we have an instance I of the stable marriage problem, and that for any person q in I, $PL(q)$ is the ordered list of persons in the original preference list of q and $GS(q)$ is the ordered list of people in the GS-list for q, and initially $GS(q)$ equals $PL(q)$. In a *proposal* from man m to woman w, w will be at the head of the man's GS-list $GS(m)$. This leads to an *engagement* where m is no longer free and all men that w prefers less than m are removed from her GS-list, i.e. the last entry in $GS(w)$ becomes m. Further, when a man p is removed from $GS(w)$ that woman is also removed from his GS-list, i.e. w is removed from $GS(p)$, consequently bigamy is disallowed. Therefore m and w are engaged when m is no longer free, w is head of $GS(m)$, and m is at the tail of $GS(w)$. A man p becomes *free* when p was engaged to w (i.e. the head of $GS(p)$ is w) and w receives a proposal from man m that she prefers to p. On becoming free, p is added to the list of free men and w is removed from $GS(p)$.

The algorithm starts with all men free and placed on a list (line 1). The algorithm then performs a sequence of proposals (lines 2 to 13). A man m is selected from the free list (line 2), and his most preferred woman w is selected (line 4). If w is engaged, then her partner p becomes free. The pair m and w then become engaged (lines 7 to 12).

3 Preliminaries

We assume that each man and woman's preference list has been read into two dimensional integer arrays mpl and wpl respectively. $mpl[i]$ is the preference list for the i^{th} man where $mpl[i][j]$ is the i^{th} man's j^{th} preference, and similarly $wpl[j]$ is the preference list for the j^{th} woman. Using our problem in Figure 1(a), if we consider our third man he will have a preference list $mpl[3] = (1, 4, 5, 3, 6, 2)$.

We also assume we have the inverse of the preference lists, i.e. mPw and wPm, where $mPw[i][j]$ is the i^{th} man's preference for the j^{th} woman and $wPm[j][i]$ is the j^{th} woman's preference for the i^{th} man. Again, considering the

third man in Figure 1, his inverse preference list will be $mPw[3] = (1, 6, 4, 2, 3, 5)$, $mPw[3][2]$ is his preference for the second woman, and that is 6, i.e. woman 2 is in the sixth position of man 3's preference list.[2]

We associate a constrained integer variable with each man and each woman, such that $x[i]$ is a constrained integer variable representing the i^{th} man m_i in stable marriage instance I and has a domain $dom(x[i])$ initially of 1 to n. Similarly, we have an array of constrained integer variables for women, such that $y[j]$ represents the j^{th} woman w_j in I. The values in the domain of a variable correspond to preferences, such that if variable $x[i]$ is assigned the value j this corresponds to m_i being married to his j^{th} choice of woman, and this will be woman $mpl[i][j]$. For example, if $x[2]$ (in Figure 1) is set to 3 then this corresponds to m_2 marrying his third choice, w_1 (and conversely $y[1]$ would then have to be assigned the value 5). Again referring to Figure 1(a) our sixth man's domain is $dom(x[6]) = (1, 2, 3, 4, 5, 6)$, as is everyone else's, and in 1(b) $dom(x[6]) = (1, 4, 5)$. 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 $dom(v)$.
- $setMax(v, a)$ sets the maximum value in $dom(v)$ to be $\min(getMax(v), a)$.
- $removeValue(v, a)$ removes the value a from $dom(v)$.

We assume that constraints are processed by an arc-consistency algorithm such as AC5 [19] or AC3 [15]. That is, the algorithm has a stack of constraints that are awaiting revision and if a variable loses values then all the constraints that the variable is involved in are added to the stack along with the method that must be applied to those constraints, i.e. the stack contains methods and their arguments. Furthermore, we also assume that a call to a method, with its arguments, is only added to the stack if it is not already on the stack. We'll refer to this stack as the *call stack*.

4 A Binary Stable Marriage Constraint (SM2)

We now give a description of our binary stable marriage constraint, where arc-consistency on such an encoding is equivalent to an application of the male-oriented EGS algorithm. Note that the constraint as described (minimally) cannot be used within a search process, however we will later show how this can be done. Our constraint is binary in that it constrains a man and a woman, such that stability is maintained and bigamy is disallowed. In a stable marriage problem with n men and n women we will then require n^2 of these constraints. We now start by describing the attributes of the constraint, how to construct the constraint, and then the three methods that act upon it. We will use a java-like pseudo-code such that the . (dot) operator is an attribute selector, such that $a.b$ delivers the b attribute of a.

[2] The inverse of the preference lists can be created when reading in the preference lists such that $mPw[i][mpl[i][j]] = j$, and this does not affect the overall complexity of constructing our model.

4.1 The attributes

A binary stable marriage constraint (SM2) is an object that acts between a man and a woman, and has the following attributes:

- x and y are constrained integer variables representing the man and the woman that are constrained.
- xPy is x's preference for y. That is, if x corresponds to the i^{th} man and y corresponds to the j^{th} woman then $xPy = mPw[i][j]$.
- yPx is y's preference for x. If y corresponds to the j^{th} woman and x to the i^{th} man then $yPx = wPm[j][i]$.

Therefore a constraint between the i^{th} man and j^{th} woman is constructed via a call to the function $SM2(x[i], mPw[i][j], y[j], wPm[j][i])$. This will construct a constraint object c such that $c.x = x[i]$, $c.y = y[j]$, $c.xPy = mPw[i][j]$, and $c.yPx = wPm[j][i]$. To construct our constraint encoding we would then make a call to $SM2$, as shown, with i and j varying from 1 to n creating the n^2 constraints.

4.2 The propagation methods

We now describe three methods that achieve male-oriented arc-consistency between a man x and woman y across a constraint c.

deltaMin(c) This method is called across the constraint c between x and y when the lower bound of the domain of x increases. If the lower bound increases such that the lowest value in the domain of x corresponds to that man's preference for woman y (line 2) then that woman need not consider any man she prefers less than man x. Consequently we can remove from her domain all preferences greater than her preference for man x (line 3).

```
1.  deltaMin(c)
2.    IF getMin(c.x) = c.xPy
3.    THEN setMax(c.y,c.yPx)
```

deltaMax(c) We now describe the method that deals with the situation when the upper bound of a woman $c.y$ is reduced. If woman $y's$ least preferred choice is better than her preference for man x (line 2) then man x is no longer in $y's$ preference list. Therefore we remove woman y from man $x's$ preference list (line 3).

```
1.  deltaMax(c)
2.    IF getMax(c.y) < c.yPx
3.    THEN removeValue(c.x,c.xPy)
```

init(c) The *init* method is called when the constraint is created, and is simply a call to $deltaMin$.

```
1.  init(c)
2.    deltaMin(c)
```

5 Comparison to EGS

We now compare the behaviour of our binary constraint model (SM2) to the male-oriented EGS algorithm. In our comparison we will describe steps in the EGS algorithm in *italics* and the SM2 constraint encoding in normal font. For ease of exposition we may refer to a constraint acting between man m and woman w as $C_{x,y}$, where m and w are people in the SM instance, and x and y are the corresponding variables in our constraint encoding. Sometimes we will use m and w as a particular person (rather than m_i and w_j), and x and y as particular variables (rather than $x[i]$ and $y[j]$) for sake of brevity. Additionally, we assume we have the function $fiance(y[j])$ and that it delivers the constrained integer variable $z = x[i]$ where $i = wpl[j][max(dom(y[j])]$, i.e. the least preferred partner of $y[j]$.

- *Initially the EGS algorithm sets all men to be free by adding them to the free list (line 1).* Equivalently, before propagation starts the set of calls $\{init(C_{x[i],y[j]})|1 \leq i, j \leq n\}$ is added to the empty call stack.
- *EGS picks a man m from the free list and he then proposes to his first choice woman w (lines 4 to 7).* Initially, the constraints on the stack will be revised using the $deltaMin$ method, called directly via $init$. When executing the call $deltaMin(C_{x,y})$, if y is not $x's$ current favourite (i.e. $min(dom(x)) \neq xPy$) then no action is taken. However, if y is $x's$ favourite the equivalent of a proposal will be made (as described next).
- *When m makes a proposal to w all values that appear in $GS(w)$ after the proposing man are removed (lines 8 to 10), i.e. they become engaged.* With the call $deltaMin(C_{x,y})$, when y is $x's$ favourite, the maximum of $dom(y)$ is set to $y's$ preference for x, therefore removing all less preferred men. Effectively, x and y become engaged.
- *To maintain monogamy EGS removes the newly engaged woman from the GS-lists of all men that have just been removed from her preference list (line 11).* From the action above, the maximum of $dom(y)$ has been lowered, consequently the set of calls $\{deltaMax(C_{x[i],y})|1 \leq i \leq n\}$ are added to the call stack. For a call to $deltaMax(C_{x,y})$, if $y's$ preference for x is greater than the maximum value in $dom(y)$ then $y's$ preference for x has already been removed from $dom(y)$, consequently $x's$ preference for y is removed from $dom(x)$. Therefore, x and y can never be married.
- *In EGS, if m makes a proposal to w, who is already engaged to p, then $w's$ previous fiance p is assigned to be free and added to the free list (lines 5 and 6.)* On initiating the call $deltaMin(C_{x,y})$, $y's$ fiance corresponds to the maximum value in $dom(y)$, because all less preferred men will have been removed (as above). Therefore if y receives a proposal from x via the call $deltaMin(C_{x,y})$, and y prefers x to her current fiance $z = fiance(y)$, the maximum of $dom(y)$ will be set lower than her preference for z and therefore the preference for z will be removed from $dom(y)$. Consequently, the set of calls $\{deltaMax(C_{x[i],y})|1 \leq i \leq n\}$ will be stacked, one of which will be the call $deltaMax(C_{z,y})$, and the preference for y will then be removed from

$dom(z)$. And because y was $z's$ previous favourite the preference for y would have been $min(dom(z))$. Therefore removing that value will increase $z's$ domain minimum, and the set of calls $\{deltaMin(z, y[j])|1 \leq j \leq n\}$ are then added to the stack. And this effectively assigns man z to be free.

6 Proof that GS-lists are produced by the SM2 encoding

In order to prove that EGS and our SM2 constraint encoding are equivalent we will first show that if EGS removes a value then so does SM2. Conversely, we then prove that if SM2 removes a value then so does EGS. To help make these proofs more readable we will first give some definitions of terms and phrases that will be used.

- $x[i]$ proposes to $y[j]$ when the method $deltaMin(C_{x[i],y[j]})$ is called, where $y[j]$ is $x[i]$'s favourite potential partner, i.e. $j = mpl[i][k]$ where $k = min(dom(x[i]))$.
- $x[i]$ is said to be removed from the domain of $y[j]$ when k is removed from $dom(y[j])$ where $k = wPm[j][i]$. Conversely $y[j]$ is said to be removed from the domain of $x[i]$ when l is removed from $dom(x[i])$ where $l = mPw[i][j]$.
- $x[i]$ is free when there is a call $deltaMin(C_{x[i],y[j]})$ on the call stack, where $y[j]$ is $x[i]$'s favourite potential partner. Therefore, to make $x[i]$ free the set of calls $\{deltaMin(C_{x[i],y[j]})|1 \leq j \leq n\}$ is added to the call stack, although only one of these calls will have an effect, and that is when $y[j]$ is $x[i]$'s most preferred partner.
- $x[i]$ is engaged to $y[j]$ if and only if $x[i]$ is not free and $j = mpl[i][k]$ and $i = wpl[j][l]$, where $k = min(dom(x[i]))$ and $l = max(dom(y[j]))$.
- $x[i]$ is rejected by $y[j]$ if k has been removed from $dom(y[j])$, where $k = wPm[j][i]$.

Proof that if EGS removes a person from a GS-list then SM2 will remove the corresponding value from the corresponding variable We use a proof by cases, considering the situations where EGS can remove people from GS-lists, and the situation where men can become free. There are 3 cases to consider. First we prove that if a proposal in EGS causes people to be removed from GS-lists, then the corresponding proposal in SM2 will result in the same corresponding values being removed from the relevant constrained integer variables. We then prove that if an event in EGS causes a man to be placed on the free list then a similar event in SM2 will add calls to $deltaMin$ to the call stack. Finally we prove that because the order in which the proposals are made does not effect the resulting domains, if EGS removes a person from a GS-list then SM2 will have a corresponding effect.

Lemma 1 In EGS, when a man proposes to a woman and removes values from her GS-list, the constraint encoding will remove the corresponding set of values from the domain of the corresponding variable.

Proof In EGS a proposal is made from m_i to w_j when m_i is free and w_j is the first entry in $GS(m_i)$, i.e. w_j is m_i's favourite potential partner. All men in $GS(w_j)$ that w_j prefers less than m_i are then removed from $GS(w_j)$, and for each removed man p, w_j is removed from $GS(p)$ thus preventing bigamy.
The proposal is made in SM2 via a call to $deltaMin(C_{x[i],y[j]})$, where $x[i]$ and $y[j]$ correspond to m_i and w_j, and the smallest value in $dom(x[i])$ is $x[i]$'s preference for $y[j]$ (see line 2 of $deltaMin$), i.e. $y[j]$ is $x[i]$'s favourite potential partner. All values greater than $wPm[j][i]$ will be removed from $dom(y[j])$ (line 3 of $deltaMin$). Consequently all values in $dom(y[j])$ corresponding to men she likes less than $x[i]$ will be removed. Since the maximum value in $dom(y[j])$ has decreased the set of calls $\{deltaMax(x[k], w[j])|1 \leq k \leq n\}$ are added to the call stack. When a call to $deltaMax(x[k], y[j])$ is executed and $wPm[j][k]$ is greater than the maximum value in $dom(w[j])$, $mPw[k][j]$ will then be deleted from $dom(x[k])$. □

Lemma 2 If circumstances occur that cause EGS to assign m_i to be free then the same circumstances will cause SM2 to assign $x[i]$ to be free

Proof The EGS algorithm adds men to the free list under two conditions. The first is when the algorithm is initiated and all men are set to be free. The second is when m_i is rejected by a woman he was previously engaged to, w_j.
When SM2 is initialised the set of calls $\{init(C_{x[i],y[j]})|1 \leq i, j \leq n\}$ will be added to the empty call stack. This will in turn make a call to each method in the set $\{deltaMin(C_{x[i],y[j]})|1 \leq i, j \leq n\}$. This effectively assigns all men to be free.
If man $x[i]$ is rejected by woman $y[j]$, then the value $mPw[i][j]$ will be removed from $dom(x[i])$. Because $x[i]$ previously proposed to $y[j]$, the minimum value in $dom(x[i])$ must have been $mPw[i][j]$. Therefore when $mPw[i][j]$ was removed from $dom(x[i])$, the minimum value in $dom(x[i])$ must have increased and thus caused the set of calls $\{deltaMin(C_{x[i],y[k]})|1 \leq k \leq n\}$ to be put on the call stack, thus effectively assigning $x[i]$ to be free. □

Lemma 3 If EGS removes a person from a GS-list then SM2 will remove the corresponding value from the relevant variable's domain.

Proof EGS only removes people from a GS-list as a direct result of a proposal (the WHILE loop in Figure 2). From lemma 1 when EGS removes a person from a GS-list SM2 removes the corresponding values from the domain of the relevant variable. From lemma 2, when EGS causes a man to be free so too does SM2. Consequently when EGS removes a person from a GS-list then SM2 will remove the corresponding value from the relevant variable's domain. □

Proof that if SM2 removes a value from a variable then EGS will remove the corresponding person from the relevant GS-List As above this proof will be split into three parts. First, if in SM2 a value is removed from

a variable's domain as a result of a proposal then the same proposal in EGS will cause the corresponding person to be removed from the relevant GS-List (i.e. the converse of lemma 1). Second, if SM2 assigns a man to be free then EGS in the same circumstances will also assign the same man to be free (the converse of lemma 2). The third combines the previous two to say if SM2 removes a value from the domain of a variable then EGS will remove the corresponding person from the relevant GS-List (similar to lemma 3).

Lemma 4 If in SM2 a value is removed from a variable's domain as a result of a proposal then the same proposal in EGS will cause the corresponding person to be removed from the relevant GS-List.

Proof In SM2 only two types of domain reductions can occur. The maximum value of $dom(y[j])$ can be set to $wPm[j][i]$ in a call to $deltaMin(C_{x[i],y[j]})$ or $mPw[i][j]$ could be removed from $dom(x[i])$ in a call to $deltaMax(C_{x[i],y[j]})$. Therefore all values removed by SM2 as the result of a proposal must be one of these two types. Because $deltaMin$ only alters the domains of y variables it can only cause calls to $deltaMax$, and likewise $deltaMax$ only removes values from the domains of x variables so can only cause calls to $deltaMin$. And because a call to $deltaMin$ is classed as a proposal the only propagation effect from $deltaMax$ is further proposals.

When a proposal is made in SM2 by a call to $deltaMin(C_{x[i],y[j]})$ The maximum value of $dom(y[j])$ will be set to $wPm[j][i]$, thus removing all men $y[j]$ likes less than $x[i]$. The resulting set of calls $\{deltaMax(C_{x[k],y[j]})|1 \leq k \leq n\}$ will then remove $mPw[k][j]$ from $dom(x[k])$ for all $x[k]$, where $wPm[j][k]$ is greater than the maximum value in $dom(y[j])$, thus removing $y[j]$ from the domains of all men she likes less than $x[i]$.
In EGS when a proposal is made from man m_i to woman w_j all men in $GS(w_j)$ corresponding to men she likes less than m_i are removed. Then w_j is removed from $GS(m_k)$ for all m_k where m_k was removed from $GS(w_j)$. □

Lemma 5 If circumstances occur that cause SM2 to assign man $x[i]$ to be free then the same circumstances will cause EGS to assign the same man m_i to be free.

Proof In SM2 there are only two events that will cause man $x[i]$ to be placed on the free list. The first is when the set of calls $\{init(C_{x[i],y[j]})|1 \leq i,j \leq n\}$ is placed on the stack. This will in turn call $\{deltaMin(C_{x[i],y[j]})|1 \leq i,j \leq n\}$. This effectively assigns all men to be free.
The other is when the minimum value of $dom(x[i])$ is increased and thus causes the set of calls $\{deltaMin(C_{x[i],y[j]})|1 \leq j \leq n\}$ to be placed on the stack. This can only be due to a call to $deltaMax(C_{x[i],y[j]})$ where $mPw[i][j]$ is the minimum value in $dom(x[i])$. If $mPw[i][j]$ was not the minimum value in $dom(x[i])$ when the call to $deltaMax(C_{x[i],y[j]})$ was made then removing $mPw[i][j]$ from $dom(x[i])$ won't cause any further propagation. If $mPw[i][j]$ is the minimum value in $dom(x[i])$ then $x[i]$ will either be engaged to $y[j]$ or will have not yet

proposed and thus be already assigned free. Because $y[j]$ can only be engaged to one man at a time, however many values are removed from $dom(y[j])$ as a result of a call to $deltaMin(C_{x[k],y[j]})$ (a proposal from some other man x[k]) only one man $x[i]$ can be placed on the free list that was not already there, where $x[i]$ was previously engaged to $y[j]$.
The EGS algorithm adds men to the free list under two conditions. The first is when the algorithm is first started and all men are set to be free. The second is man m_i is placed on the free list if he is rejected by a woman w_j, where m_i and w_j were previously engaged. □

Lemma 6 If SM2 removes a value from a domain then EGS will also remove the corresponding person from the relevant GS-List.

Proof SM2 only removes values as a direct result of a proposal. From lemma 4 if in SM2 a value is removed from a variable's domain as a result of a proposal then the same proposal in EGS will cause the corresponding person to be removed from the relevant GS-List. From lemma 5 if circumstances occur that cause SM2 to assign $x[i]$ to be free then the same circumstances will cause EGS to assign m_i to be free. Consequently, if SM2 removes a value from a domain then EGS will remove the corresponding person from the relevant GS-List. □

Proof that GS-Lists are produced This section simply pulls together the previously presented lemmas to prove that the GS-Lists are produced.

Theorem When a SM instance is made arc consistent using SM2, the resulting domain values will be the equivalent of the GS-Lists produced by EGS from the same SMP.

Proof From lemma 6 if SM2 removes a value from a domain then EGS will also remove the corresponding person from the relevant GS-List, therefore the values removed by SM2 must be a subset of the values removed by EGS. From lemma 3 if EGS removes a person from a GS-list then SM2 will remove the corresponding value from the relevant variable's domain, therefore the values removed by EGS must be a subset of the values removed by SM2. Therefore the set of values removed by SM2 must be the equivalent of the set of people removed by EGS, and thus the remaining values will be equivalent. Therefore SM2 produces the equivalent of the GS-Lists. □

7 Complexity of the model

In [10] section 1.2.3 it is shown in the worst case there is at most $n(n-1)+1$ proposals that can be made by the EGS algorithm, and that the complexity is then $O(n^2)$. We argue that the complexity of our SM2 encoding is $O(n^3)$. First we claim that the call to our methods $deltaMin()$ and $deltaMax()$ is of complexity $O(1)$.

When the lower bound of a man $m's$ domain is increased (or is initially given) this will result in the n calls $\{deltaMin(m, woman[i])|1 \leq i \leq n\}$. Only one of these will result in the reduction of a woman w's domain, and this single event will result in the n additional calls $\{deltaMax(man[i], w|1 \leq i \leq n\}$. This then amounts to $O(n)$ steps. We assume that a lower bound can change at most $n-1$ times, and that this can happen to all n men. Therefore in total we have at most $O(n^3)$ steps performed.

8 Enhancing the model

The full GS-Lists are the union of the male and female Gale-Shapley lists remaining after executing male and female oriented versions of EGS. It has been proven that the same lists can be produced by running the female orientated version of EGS on the male-oriented GS-lists [10]. Because SM2 produces the same results as EGS the full GS-Lists can be produced in the same way. But because of the structure of this specialised constraint it is also possible to combine the male and female orientated versions of SM2 into one constraint. This combined gender free version of SM2 will then produce the full GS-List with only one run of the arc-consistency algorithm.

The SM2 constraint as presented so far has only considered domain values being removed by the constraint's own methods. If we were to use the constraint to find all possible stable matchings, unless arc consistency reduces all variable domains to a singleton, it will be necessary to assign and remove values from variable domains as part of a search process. Therefore, we need to add code to SM2 to maintain consistency and stability in the event that domain values are removed by methods other than those within SM2. It is important to note that these external domain reductions could also be caused by side constraints as well as a search process.

There are four types of domain reduction that external events could cause: a variable is instantiated; a variable's minimum domain value is increased; a variable's maximum domain value is reduced; one or more values are removed from the interior of a variable's domain. We now describe two new methods, *inst* and $removeValue$, and the enhancements required for $deltaMin$. We note that $deltaMax$ does not need to change, and describe the required enhancements for incomplete preference lists.

inst(c) The method $inst(c)$ is called when a x variable is instantiated.

```
1.   inst(c)
2.    IF getValue(c.x) = c.xPy
3.    THEN setValue(c.y,c.yPx)
4.    ELSE IF getValue(c.x) > xPy
5.         THEN setMax(c.y,c.yPx - 1)
6.         ELSE removeValue(c.y,c.yPx)
```

If x prefers to be matched to y (line 2) then y must be instantiated to her preference for x to maintain consistency (line 3). However, if x is matched to

someone that he prefers less than y (line 4) then in order to maintain stability y can only marry people that she prefers to x (otherwise x and y will elope). Consequently we delete all less preferred men, including x, from her domain (line 5). Finally, if x would not prefer to be matched to y than his current partner then we simply deny y the opportunity of marrying x (line 6). Note that $getValue(v)$ delivers the integer value that has been assigned to variable v, $setValue(v, a)$ instantiates v to the integer value a, and these methods are of $O(1)$ complexity.

removeValue(c,a) This method is called when the integer value a is removed from $dom(x)$, and this value is neither the largest nor smallest in $dom(x)$.

```
1.   removeValue(c,a)
2.    IF a = c.xPy
3.    THEN removeValue(c.y,c.yPx)
```

If the value a corresponds to x's preference for y (line 2) then the corresponding value must be removed from $dom(y)$, and that is yPx (line 3), and this must be done to prevent bigamy.

Enhancements to deltaMin(c) Up till now we have assumed that all values removed from the head of $dom(x)$ are as a result of x being rejected by some y variables. We now drop this assumption. We add a new conditional (line 4 below) to address the situation where x would prefer to be matched to y than to his current best preference, consequently y must only consider partners she prefers to x (line 5), and this is done to avoid instability.

```
1.   deltaMin(c)
2.    IF getMin(c.x) = c.xPy
3.    THEN setMax(c.y,c.yPx)
4.    ELSE IF getMin(c.x) > c.xPy
5.         THEN setMax(c.y,c.yPx - 1)
```

No enhancements to deltaMax(c) We now consider the situation where some process, other than a proposal, removes values from the tail of a woman's preference list, i.e. when the maximum value of $dom(y)$ changes. The $deltaMax$ method will be called, and the instance continues to be stable as y can still marry partners. However, we need to prevent bigamy, by removing y from the domains of the corresponding x variables removed from the tail of $dom(y)$, and this is just what $deltaMax$ does. Therefore, no enhancement is required.

Incomplete Lists (SMI) The encoding can also deal with incomplete preference lists, i.e. instances of the stable marriage problems with incomplete lists (SMI). For a SM instance of size n we introduce the value $n+1$. The value $n+1$ must appear in the preference lists $mpl[i]$ and $wpl[j]$ as a *punctuation* mark, such that any people after $n+1$ are considered unacceptable. For example, if we had an instance of size 3 and a preference list $PL(m_i)$ = (3,2) we would construct

$mpl[i] = (3, 2, 4, 1)$ and this would result in the inverse $mPw[i] = (4, 2, 1, 3)$. Consequently $x[i]$ would always prefer to be unmatched (assigned the value 4) than to be married to $y[1]$. We now need to modify the *init* method such that it sets the maximum value in $dom(x[i])$ to be $mPw[i][n + 1]$. These modifications will only work in the full implementation (i.e. it requires the above enhancements).

9 Computational Experience

We implemented our encodings using the JSolver toolkit [1], i.e. the Java version of ILOG Solver, and also the Koalog [3] and JChoco [2] toolkits. All three implementations performed similarly, therefore we present only our JSolver results. We implemented four constraint encodings for SM. The first two are those presented in [7], namely the $O(n^4)$ forbidden tuples model (FT) and the optimal $O(n^2)$ boolean encoding (Bool). In the FT model there are n^2 binary table constraints and $2.n$ variables with domains 1 to n. The constraints explicitly list the disallowed tuples that correspond to unstable or bigamous assignments. In the Bool encoding there are $2.n^2$ 0/1 variables, where a variable corresponds to a specific man or woman being matched with a given preference. Implication constraints act between 0/1 variables to simulate the topping and tailing of preference lists. The minimal encoding from section 4 (and referred to as SM-lite) produces the intersection of the male and female GS-lists, but cannot be used in search. The full encoding, referred to as SM2, is the full implementation that can be used in search and allows us to enumerate all solutions in a failure-free manner as in the [7] encodings. Our experiments were run on a Pentium 4 2.8Ghz processor with 512 Mbytes of random access memory, running Microsoft Windows XP Professional and Java2 SDK 1.4.2.6 with an increased heap size of 512 Mbytes.

	size n										
model	45	100	200	300	400	500	600	700	800	900	1000
FT	8.94	-	-	-	-	-	-	-	-	-	-
Bool	0.25	1.2	4.4	-	-	-	-	-	-	-	-
SM2lite	0.16	0.22	0.45	0.89	1.72	2.79	3.96	5.62	7.48	9.46	11.94
SM2	0.16	0.23	0.5	0.94	1.82	2.95	4.21	5.95	8.02	9.82	12.47

Fig. 3. Average computation times in seconds to produce the GS-lists, from 10 randomly generated stable marriage problems each of size n

Our first experiment measures the time taken to generate a model of a given SM instance and make that model arc-consistent, i.e. to produce the GS-lists. Figure 3 shows the average time taken to produce the GS-lists for ten randomly generated instances of size 45 up to 1000. Time is measured in seconds, and an entry $-$ means that an out of memory error occurred. We can see that the SM2-

lite and SM2 versions perform much the same, and dominate the other models.

	size n										
model	45	100	200	300	400	500	600	700	800	900	1000
FT	9.32	-	-	-	-	-	-	-	-	-	-
Bool	0.36	2.02	6.73	-	-	-	-	-	-	-	-
SM2	0.21	0.47	1.97	5.43	10.13	19.22	27.27	43.04	54.98	77.89	124.68

Fig. 4. Average computation times in seconds to find all solutions to 10 randomly generated stable marriage problems each of size n

This second experiment measures the time taken to generate a model and find all possible stable matchings. Figure 4 shows the average time taken to find all solutions on the same randomly generated instances used in the first experiment. Again it can be seen that the SM2 model dominates the other models.

	FT	bool	SM2
time	$O(n^4)$	$O(n^2)$	$O(n^3)$
constraints	$O(n^4)$	$O(n^2)$	$O(n^2)$
variables	$O(n)$	$O(n^2)$	$O(n)$

Fig. 5. a summary of the complexities of the three constraint models

Figures 3 and 4 raise the following question, if the Bool encoding is optimal then why is it dominated by the SM2 encoding? The main reason for this is that there is no significant difference in the space required to represent variables with significant differences in domain size, because domains are represented as intervals when values are consecutive. Considering only the variables, the Bool encoding uses $O(n^2)$ space whereas the SM2 model uses $O(n)$ space. For example, with $n = 1300$ the Bool encoding runs out of memory just by creating the 2.1300^2 variables whereas the SM2 model takes less than 0.25 seconds to generate the required 2600 variables each with a domain of 1 to 1300. As can be seen in Figure 5 theoretically the space complexity of the constraints used by SM2 and Bool are the same. In practise this is not the case as SM2 requires exactly n^2 constraints to solve a problem of size n whereas Bool requires $2n+6n^2$ constraints. Therefore the Bool encoding requires more variables and more constraints, resulting in a prohibitively large model.

We now investigate the unweighted sex equal optimisation problem. In the (NP-Hard) sex equal stable marriage problem (SESMP) [10, 13] both men and women are to be equally well matched. In an unweighted SESMP scores are the same as preferences, therefore we find the matching that minimises the abso-

lute difference between the sum of the men's preferences and the sum of the women's preferences. In the SM2 model the values in the domains of variables are preferences, consequently it is straight forward to model the SESMP. All that is required is to add a search goal to minimise the absolute difference between the sum of all x variables and the sum of all y variables. However it is difficult to model the same problem using the Bool constraints. This is because we now have to introduce $2.n$ additional variables with domains 1 to n and an additional $O(n^2)$ channelling constraints to set those variables. Figure 6 gives the average run time to find the unweighted sex equal matchings to 100 random problems using the SM2 and Bool models.

	size n									
model	100	200	300	400	500	600	700	800	900	1000
Bool	1.9	6.95	-	-	-	-	-	-	-	-
SM2	0.4	1.2	2.87	5	9.15	14.49	20.86	28.65	38.25	52.64

Fig. 6. Average computation times in seconds to find a sex equal solution to 100 randomly generated stable marriage problems each of size n

As can be seen, again the SM2 encoding solves problems faster and can extend to larger instances. The Bool encoding fails to model problems of size 300 and above, whereas the SM2 encoding can solve problems of size 1000 in less than a minute.

10 Conclusion

We have presented a specialised binary constraint for the stable marriage problem. We have demonstrated that the constraint can be used when stable marriage is just a part of a larger, richer problem. Our experience has shown that the constraint is simple to implement in a constraint programming toolkit, such as JSolver, JChoco, and Koalog. The complexity of the constraint is $O(n^3)$, and does not compare well to the theoretically optimal $O(n^2)$ complexity of the boolean encoding in [7]. However, our constraint is more practical than those in [7], typically being able to solve larger problems faster. For example, we have been able to enumerate all solutions to instance of size 1000 in minutes, whereas in [8] the largest problems investigated were of size 60. It is obvious that our model wastes considerable effort. The arc-consistency algorithm typically adds $n-1$ redundant calls to the revision stack whenever a change takes place, and it is trivial to detect those redundancies. This suggests that it would be easy to design a space efficient n-ary SM constraint that is of complexity $O(n^2)$.

Acknowledgements

We are grateful to ILOG SA for providing us with the JSolver toolkit via an Academic Grant licence. We would also like to thank our reviewers.

References

1. ILOG JSolver. http://www.ilog.com/products/jsolver/.
2. JChoco constraint programming system. http://choco.sourceforge.net/.
3. Koalog Constraint Solver. http://www.koalog.com/.
4. B. Aldershof and O. Carducci. Refined inequalities for stable marriage. *Constraints*, 4:281–292, 1999.
5. Canadian Resident Matching Service. How the matching algorithm works. Web document available at `http://www.carms.ca/matching/algorith.htm`.
6. D. Gale and L. Shapley. College admissions and the stability of marriage. *American Mathematical Monthly*, 69:9–15, 1962.
7. I. Gent, R. Irving, D. Manlove, P. Prosser, and B. Smith. A constraint programming approach to the stable marriage problem. In *CP'01*, pages 225–239, 2001.
8. I. Gent and P. Prosser. An empirical study of the stable marriage problem with ties and incomplete lists. In *ECAI'02*, 2002.
9. M. Green and D. Cohen. Tractability by approximating constraint languages. In *Proceedings of CP '03*, volume 2833 of *Lecture Notes in Computer Science*, pages 392–406. Springer-Verlag, 2003.
10. D. Gusfield and R. W. Irving. *The Stable Marriage Problem: Structure and Algorithms*. The MIT Press, 1989.
11. R. Irving. Matching medical students to pairs of hospitals: a new variation on a well-known theme. In *Proceedings of ESA '98*, volume 1461 of *Lecture Notes in Computer Science*, pages 381–392. Springer-Verlag, 1998.
12. R. Irving and D. Manlove. The stable roommates problem with ties. *Journal of Algorithms*, 43:85–105, 2002.
13. A. Kato. Complexity of the sex-equal stable marriage problem. *Japan Journal of Industrial and Applied Mathematics (JJIAM)*, 10:1–19, 1993.
14. I. 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. Manlove, R. Irving, K. Iwama, S. Miyazaki, and Y. Morita. Hard variants of stable marriage. *Theoretical Computer Science*, 276:261–279, 2002.
17. N. R. M. Program. About the NRMP. Web document available at `http://www.nrmp.org/about_nrmp/how.html`.
18. A. 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.
19. P. van Hentenryck, Y. Deville, and C.-M. Teng. A generic arc-consistency algorithm and its specializations. *Artificial Intelligence*, 57:291–321, 1992.