Specialised Constraints for Stable Matching Problems*

Student name: Chris Unsworth
Supervisor name: Patrick Prosser

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

Abstract. The stable marriage problem (SM) and the Hospital / Residents problem (HR) are both stable matching problems. They consist of two sets of objects that need to be matched to each other; in SM men to women, and in HR residents to hospitals. Each set of objects expresses a ranked preference for the objects in the other set, in the form of a preference list. The problem is then to find a matching of one set to the other such that the matching is stable. We discuss issues concerning the creation of specialised constraints to solve these problems. We then present results that suggest that constraint programming is indeed a practical technology for solving these problems.

1 Introduction

In the Stable Marriage problem (SM) [7] we have n men and n women. Each man ranks the n women into a preference list, as 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. In this case m_i and w_l would form a blocking pair.

This problem has also attracted the interest of the constraint programming community [2, 4, 6, 5, 9]. In [4] two models are introduced. The first encoding uses $2n$ constrained integer variables, one for each man and woman, with an initial domain of $\{1$ to $n\}$ and uses n^2 table constraints, one for each potential couple. The constraints explicitly list the disallowed tuples that correspond to unstable or inconsistent assignments. This model has a space complexity of $O(n^4)$ and runs in $O(n^4)$ time. The second uses $2n^2$ constrained integer variables one for each potential partner of each man and woman, with an initial domain of $\{0, 1\}$ and uses $O(n^2)$ implication constraints. This model has a space complexity of $O(n^2)$ and runs in $O(n^2)$ time which is optimal in the size of the preference lists.

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

In the Hospital / Residents problem (HR) [3] we have n residents and m hospitals. Again each resident ranks the m hospitals into a preference list, while each hospital ranks the n residents. Each hospital has a capacity c, meaning a hospital can accommodate at most c residents. The problem is to find a matching of residents to hospitals that respects the capacities of the hospitals and ensures the matching is stable. In this case a matching I is unstable if it contains a resident r_i and a hospital h_j such that r_i would prefer to be matched to h_j than to the hospital it is assigned to in I and h_j has either been assigned a resident it prefers less than r_i or has spare capacity. To date there have been no published constraint-based solutions to this problem.

Algorithms have been published for both of these problems and optimal constraint models have been published for the stable marriage problem. So the main question would be why design a specialised constraint for these problems?

The SM algorithm and the optimal constraint encodings all have a time complexity of $O(n^2)$, but in practice it can take over a minute for the constraint models to find a solution to a problem of size 45 while the SM algorithm can find a solution in less than two hundredths of a second. There have not been any optimal HR constraint models published, but I assume the same performance gap would exist. The advantage of the constraint solutions are their versatility. Many harder variants of the stable matching problems can be solved by adding simple side constraints to the existing constraint models, this is not possible with the matching algorithms. So the motivation behind a specialised constraint is to try and combine the efficiency of the algorithm with the versatility of the constraint models.

2 Specialised constraint solutions

By specialised constraint we mean a constraint that has been designed to solve a specific problem or sub problem. The first design decision when writing a specialised constraint is its arity. One option would be to write a binary constraint that acts between each pair that could potentially be unstable or inconsistent. The advantage of this approach is that it can lead to very clear simple code, because only the two constrained variables need be considered. The main disadvantage is the large number of constraints that are needed to create the model can lead to redundant propagation calls. For example if we used a binary constraint for SM, for a man m_i we would need n constraints, one for each potential partner. If during propagation a value was removed from the domain of m_i then each of the n constraints connected to m_i would need to be revised. In most cases only one of these n constraint revisions will have an effect. Thus we would make $n-1$ redundant calls. Another option would be to make the constraint n-ary. For example an n-ary constraint for SM would constrain all n men and all n women. This would mean that if a value was removed from the domain of m_i only one constraint would need to be revised, thus reducing the time complexity by n. The downside to this approach is the code can become complicated and hard to follow.

The next aspect of constraint design to consider is what type of domain reduction event should trigger propagation. The three main types of domain

reduction event are when a variable is instantiated, when the bounds of a variables domain changes and when any domain value is removed. To solve a pure SM problem propagation need only occur when the bounds of a variable changes. However if we add side constraints it may become necessary to propagate whenever any domain value is removed. To solve a HR problem the constraints need to be propagated whenever a domain variable is removed.

The final element of a specialised constraint is the propagation methods. Propagation methods are called when the constraint is initialised and when a domain reduction event occurs. The method that is to be called on initialisation has to handle any possible initial state of the variables. This method would where necessary remove one or more values from one or more of the constrained variables, which would in turn cause a domain reduction event, effectively kick-starting propagation. The methods that will be called when a domain reduction event occurs will have to handle all types of domain reduction. It is important to note that if the constraint is to be used in search and/or in conjunction with other constraints, the domain reduction events will not necessarily be triggered by the constraint it self. Therefore no assumptions can be made as to the current search state. Because of this, these methods either need to be generic or to check the state before acting.

It may be the case that information needs to be stored within a constraint object, such as a preference list. If this data is static and used for reference only then a standard integer variable will suffice. However if the data is dynamic meaning that its value will be updated during search, then a reversible variable will need to be used. Reversible variables should be supplied by the constraint toolkit. The solver will store the values of each of the reversible variables at each choice point and restore them on backtracking.

3 Computational Experience

We now present empirical results from our studies of stable marriage problems and hospital resident problems.

3.1 Stable Marriage Problems

We implemented the specialised constraint encodings using the JSolver toolkit [1], i.e. the Java version of ILOG Solver. We implemented three constraint encodings for SM. The first is an encoding presented in [4], namely the optimal $O(n^2)$ boolean encoding (Bool). The other two are encodings which use specialised constraints, a binary constraint referred to as SM2 [12] and an n-ary constraint referred to as SMN [11]. Our experiments were run on a Pentium 4 2.8Ghz processor with 512 Mbytes of RAM, running Microsoft Windows XP Professional and Java2 SDK 1.4.2.6 with an increased heap size of 512 Mbytes.

Table 1 shows the average time taken to create the model and find all solutions for ten randomly generated instances of size 100 up to 1000. Time is measured in seconds, and an entry − means that an out of memory error occurred. As can be seen the SM2 encoding significantly out performed the optimal Bool encoding in both space and time, and the SMN encoding out performs both other models by orders of magnitude.

model	size n					
	100	200	400	600	800	1000
Bool	2.02	6.73	–	–	–	–
SM2	0.47	1.97	10.13	27.27	54.98	124.68
SM2N	0.03	0.07	0.24	0.73	1.56	3.35

Table 1. Average computation times in seconds to find all solutions to 10 randomly generated stable marriage problems each of size n

	Bool	SM2	SMN
time	$O(n^2)$	$O(n^3)$	$O(n^2)$
constraints space	$O(n^2)$	$O(n^2)$	$O(n^2)$
variables space	$O(n^2)$	$O(n)$	$O(n)$

Table 2. a summary of the complexities of the three SM constraint models

Tables 1 and 2 raise the following question, if the Bool encoding is optimal then why is it dominated by both the SM2 and SMN encodings? 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 and SMN models take less than 0.25 seconds to generate the required 2600 variables each with a domain of 1 to 1300. As can be seen in table 2 theoretically the space complexity of the constraints used by SM2, SMN and Bool are the same. In practise this is not the case as SM2 requires exactly n^2 constraints each of size $O(1)$ to solve a problem of size n, and SMN requires one constraint of size $O(n^2)$. To solve the same problem Bool would require $2n + 6n^2$ constraints of size $O(1)$. Therefore the Bool encoding requires more variables and more constraints, resulting in a prohibitively large model.

We now demonstrate the versatility of the specialised constraints by introducing a richer problem where stable matching is but one feature of the problem. A derivative of SM is the sex equal stable marriage problem (SESMP). This is an optimisation problem, where the search goal is to find a matching that is equally good for both the men and the women. This problem has been proven to be NP-Hard [8]. We can model this problem easily, using our SM2 and/or SMN constraint, by introducing summation constraints within each sex. Table 3 show the average times to find a sex equal solution to one hundred randomly generated stable marriage problems, using the n-ary stable marriage constraint presented in [11].

	size n					
problem	1000	1200	1400	1600	1800	2000
SESMP	3.65	5.02	8.73	14.44	17.59	22.44

Table 3. Average computation times to find a sex equal matching in seconds from 100 randomly generated stable marriage problems each of size n

3.2 Hospital Resident Problems

We implemented three constraint encodings for HR. The first being a traditional constraint model presented in [10] referred to as CBM. The other two are encodings which use specialised constraints, a binary constraint referred to as HR2 and an n-ary constraint referred to as HRN. These experiments where run on the same machine as above.

	50/13/4	100/20/5	500/63/8	1k/100/10	5k/250/20	20k/550/37	50k/1.2k/42
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 4. Average computation times in seconds to find all solutions to 100 randomly-generated HR instances with attributes $n/m/c$.

Table 4 shows average time taken to create the model and find all solutions for one hundred randomly generated instances. The sizes of the instances are shown in the format $n/m/c$, where n is the number of residents, m is the number of hospitals and c is the uniform capacities of the hospitals. Again we see the specialised constraints significantly outperform the more traditional model. The performance on instances $50k/1.2k/42$ are of particular interests because these problems are similar in size to those in the National Resident Matching Program (NRMP) in the US. This suggests that our specialised constraints are of practical worth.

	CBM	HR2	HRN
time	$O((n+m)^3c)$	$O((n+m)^3c)$	$O((n+m)^2c)$
constraints space	$O(nmc)$	$O(nm)$	$O(nm)$
variables space	$O(n+mc)$	$O(n+m)$	$O(n+m)$

Table 5. a summary of the complexities of the three HR constraint models

Table 5 shows a summary of both the time and space complexities of the three HR constraint models.

4 Conclusion

We have discussed some issues that need to be taken into account when writing specialised constraints. We have shown that specialised constraints can be

used to solve matching problems quicker and more efficiently than traditional constraint models. We have also shown that the gain in efficiency is not at the cost of versatility, by solving SESMP an impure SM derivative. Furthermore, our empirical study of random HR instances strongly suggests that specialised constraints are indeed a practical solution technique for this important problem.

5 Future work

The matching problems as outlined in this paper both assume that all preference lists are strictly ordered. An extension of these matching problems is to allow the preference lists to contain ties. By ties we mean that there is indifference between two or more possible matches. I have created a prototype binary constraint for SM with ties and will be documented in the near future. I will then extend this to create an n-ary constraint for the same problem, then look into HR with ties. I will then go on to look at other problems with a view to solving them with specialised constraints.

Acknowledgments

I would like to thank Patrick Prosser and Douglas Graham for their editorial input and ILOG SA for providing access to the JSolver toolkit via an Academic Grant Licence.

References

1. ILOG JSolver. http://www.ilog.com/products/jsolver/.
2. B. Aldershof and O. Carducci. Refined inequalities for stable marriage. *Constraints*, 4:281–292, 1999.
3. D. Gale and L. Shapley. College admissions and the stability of marriage. *American Mathematical Monthly*, 69:9–15, 1962.
4. 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.
5. I. Gent and P. Prosser. An empirical study of the stable marriage problem with ties and incomplete lists. In *ECAI'02*, 2002.
6. 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.
7. D. Gusfield and R. W. Irving. *The Stable Marriage Problem: Structure and Algorithms*. The MIT Press, 1989.
8. A. Kato. Complexity of the sex-equal stable marriage problem. *Japan Journal of Industrial and Applied Mathematics (JJIAM)*, 10:1–19, 1993.
9. I. Lustig and J. Puget. Program does not equal program: constraint programming and its relationship to mathematical programming. *Interfaces*, 31:29–53, 2001.
10. D. Manlove, G. O'Malley, P. Prosser, and C. Unsworth. A constraint programming approach to the hospitals / residents problem. In *Under Review*, 2005.
11. C. Unsworth and P. Prosser. An n-ary constraint for the stable marriage problem. In *Under Review*, 2005.
12. C. Unsworth and P. Prosser. A specialised binary constraint for the stable marriage problem. In *SARA 2005*, 2005.