Stable Roommates and Constraint Programming

Patrick Prosser

School of Computing Science, University of Glasgow, pat@dcs.gla.ac.uk

What's the problem?

Something like stable marriage problem … but without sex.

Ant: Bea, Ann, Cat

Bob: Bea, Cat, Ann

Cal: Ann, Bea, Cat

Ann: Bob, Ant, Cal

Bea: Cal, Ant, Bob

Cat: Cal, Bob, Ant

- Men rank women,
- Women rank men
- Match men to women in a matching M such that there is no incentive for a (m,w) pair not in M to divorce and elope
- i.e. it is *stable*, there are *no blocking pairs*

Order n squared

Stable Marriage Problem (SM)

What's the problem?

Ant: ***Bea***, Ann, Cat	Ann: Bob, Ant, Cal
Bob: Bea, Cat, Ann	Bea: Cal, ***Ant***, Bob
Cal: Ann, Bea, Cat	Cat: Cal, Bob, Ant

- Men rank women,
- Women rank men
- Match men to women in a matching M such that there is no incentive for a (m,w) pair not in M to divorce and elope
- i.e. it is *stable*, there are *no blocking pairs*

Order n squared

Ant: ***Bea***, Ann, Cat

Bob: Bea, ***Cat***, Ann

Cal: Ann, Bea, Cat

Ann: Bob, Ant, Cal

Bea: Cal, ***Ant***, Bob

Cat: Cal, ***Bob***, Ant

- Men rank women,
- Women rank men
- Match men to women in a matching M such that there is no incentive for a (m,w) pair not in M to divorce and elope
- i.e. it is *stable*, there are *no blocking pairs*

Order n squared

Stable Marriage Problem (SM)

What's the problem?

Ant: ***Bea***, Ann, Cat

Bob: Bea, ***Cat***, Ann

Cal: ***Ann***, Bea, Cat

Ann: Bob, Ant, ***Cal***

Bea: Cal, ***Ant***, Bob

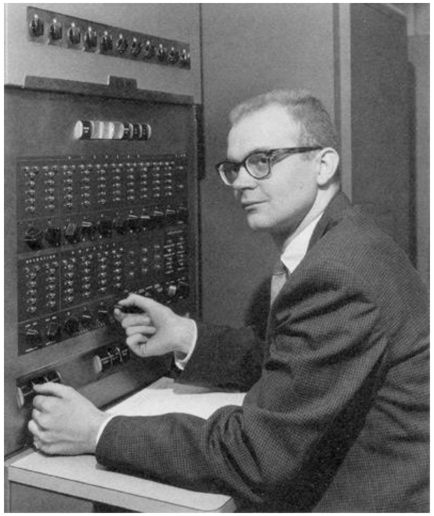
Cat: Cal, ***Bob***, Ant

- Men rank women,
- Women rank men
- Match men to women in a matching M such that there is no incentive for a (m,w) pair not in M to divorce and elope
- i.e. it is *stable*, there are *no blocking pairs*

Order n squared

In the **S**table **R**oommates problem (SR) [8, 7] we have an even number of agents to be matched together as couples, where each agent strictly ranks all other agents. The problem is then to match pairs of agents together such that the matching is stable, i.e. there doesn't exist a pair of agents in the matching such that $agent_i$ prefers $agent_j$ to his matched partner and $agent_j$ prefers $agent_i$ to his matched partner[1].

In the **S**table **R**oommates problem (SR) [8, 7] we have an even number of agents to be matched together as couples, where each agent strictly ranks all other agents. The problem is then to match pairs of agents together such that the matching is stable, i.e. there doesn't exist a pair of agents in the matching such that $agent_i$ prefers $agent_j$ to his matched partner and $agent_j$ prefers $agent_i$ to his matched partner[1].



Order n squared
(Knuth thought not)

Stable Roommates (SR)

What's the problem?

JOURNAL OF ALGORITHMS 6, 577–595 (1985)

An Efficient Algorithm for the "Stable Roommates" Problem

ROBERT W. IRVING*

Department of Mathematics, University of Salford, Salford M5 4WT, United Kingdom

Received January 31, 1984; accepted May 1, 1984

The stable marriage problem is that of matching n men and n women, each of whom has ranked the members of the opposite sex in order of preference, so that no unmatched couple both prefer each other to their partners under the matching. At least one stable matching exists for every stable marriage instance, and efficient algorithms for finding such a matching are well known. The stable roommates problem involves a single set of even cardinality n, each member of which ranks all the others in order of preference. A stable matching is now a partition of this single set into $n/2$ pairs so that no two unmatched members both prefer each other to their partners under the matching. In this case, there are problem instances for which no stable matching exists. However, the present paper describes an $O(n^2)$ algorithm that will determine, for any instance of the problem, whether a stable matching exists, and if so, will find such a matching. © 1985 Academic Press, Inc.

1. INTRODUCTION AND HISTORY

The Stable Marriage Problem

The stable marriage assignment problem was introduced by Gale and Shapley [1] in the context of assigning applicants to colleges, taking into account the preferences of both the applicants and the colleges.

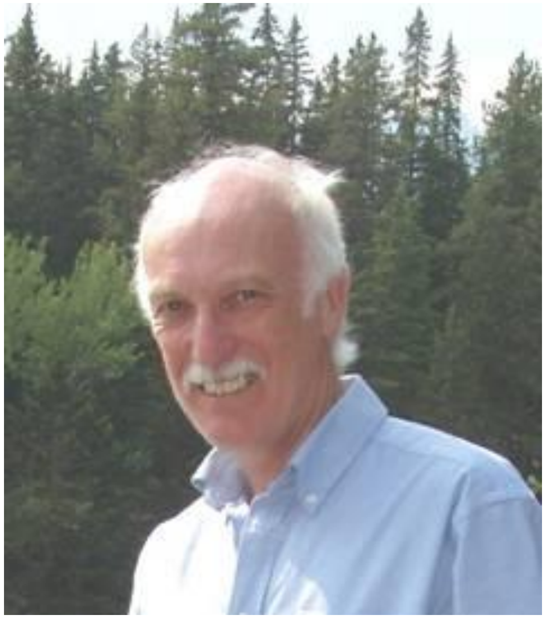
In its most familiar form, a problem instance involves two disjoint sets of cardinality n, the men and the women, with each individual having ranked the n members of the opposite sex in order of preference. A stable matching is defined as a one-to-one correspondence between the men and women with the property that there is no couple both of whom prefer each other to their actual partners.

Gale and Shapley demonstrated that at least one stable matching exists for every problem instance, and described an algorithm that would yield one such solution. McVitie and Wilson [4] proposed an alternative recursive

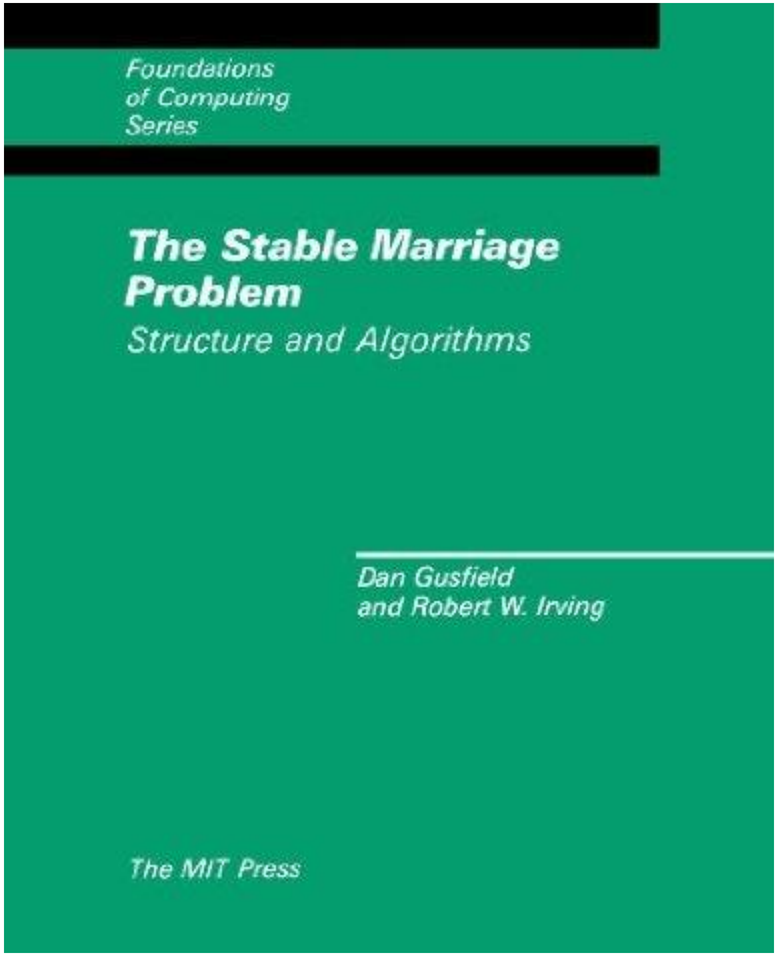
*Present address: Department of Computer Science, University of Glasgow, Glasgow G12 8QQ, United Kingdom.

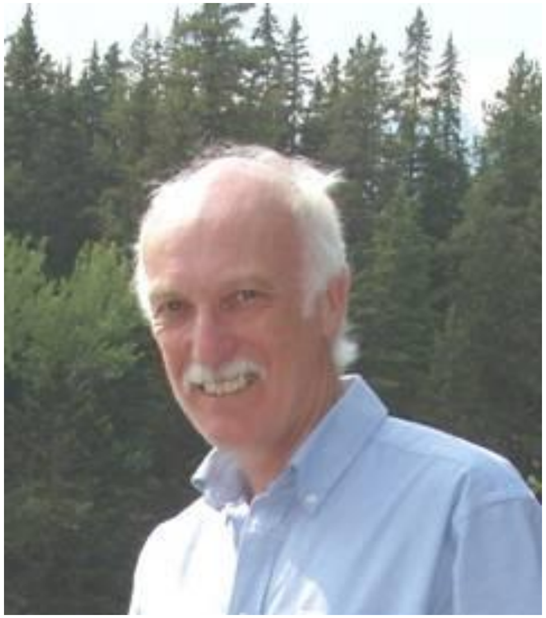
577

0196-6774/85 $3.00
Copyright © 1985 by Academic Press, Inc.
All rights of reproduction in any form reserved.



Order n squared
(Rob thought so)





The green book

1	:	8	2	9	3	6	4	5	7	10
2	:	4	3	8	9	5	1	10	6	7
3	:	5	6	8	2	1	7	10	4	9
4	:	10	7	9	3	1	6	2	5	8
5	:	7	4	10	8	2	6	3	1	9
6	:	2	8	7	3	4	10	1	5	9
7	:	2	1	8	3	5	10	4	6	9
8	:	10	4	2	5	6	7	1	3	9
9	:	6	7	2	5	10	3	4	8	1
10	:	3	1	6	5	2	9	8	4	7

Taken from "***The green book***"

10 agents, each ranks 9 others, gender-free
(n=10, n should be even)

Stable Roommates (SR)

What's the problem?

1 : 8 2 9 3 6 4 5 7 10
2 : 4 3 8 9 5 1 10 6 7
3 : 5 6 8 2 1 7 10 4 9
4 : 10 7 9 3 1 6 2 5 8
5 : 7 4 10 8 2 6 3 1 9
6 : 2 8 7 3 4 10 1 5 9
7 : 2 1 8 3 5 10 4 6 9
8 : 10 4 2 5 6 7 1 3 9
9 : 6 7 2 5 10 3 4 8 1
10 : 3 1 6 5 2 9 8 4 7

(1,7) (2,3) (4,9) (5,10) (6,8)
(1,7) (2,8) (3,5) (4,9) (6,10)
(1,7) (2,8) (3,6) (4,9) (5,10)
(1,4) (2,8) (3,6) (5,7) (9,10)
(1,4) (2,9) (3,6) (5,7) (8,10)
(1,4) (2,3) (5,7) (6,8) (9,10)
(1,3) (2,4) (5,7) (6,8) (9,10)

7 stable matchings

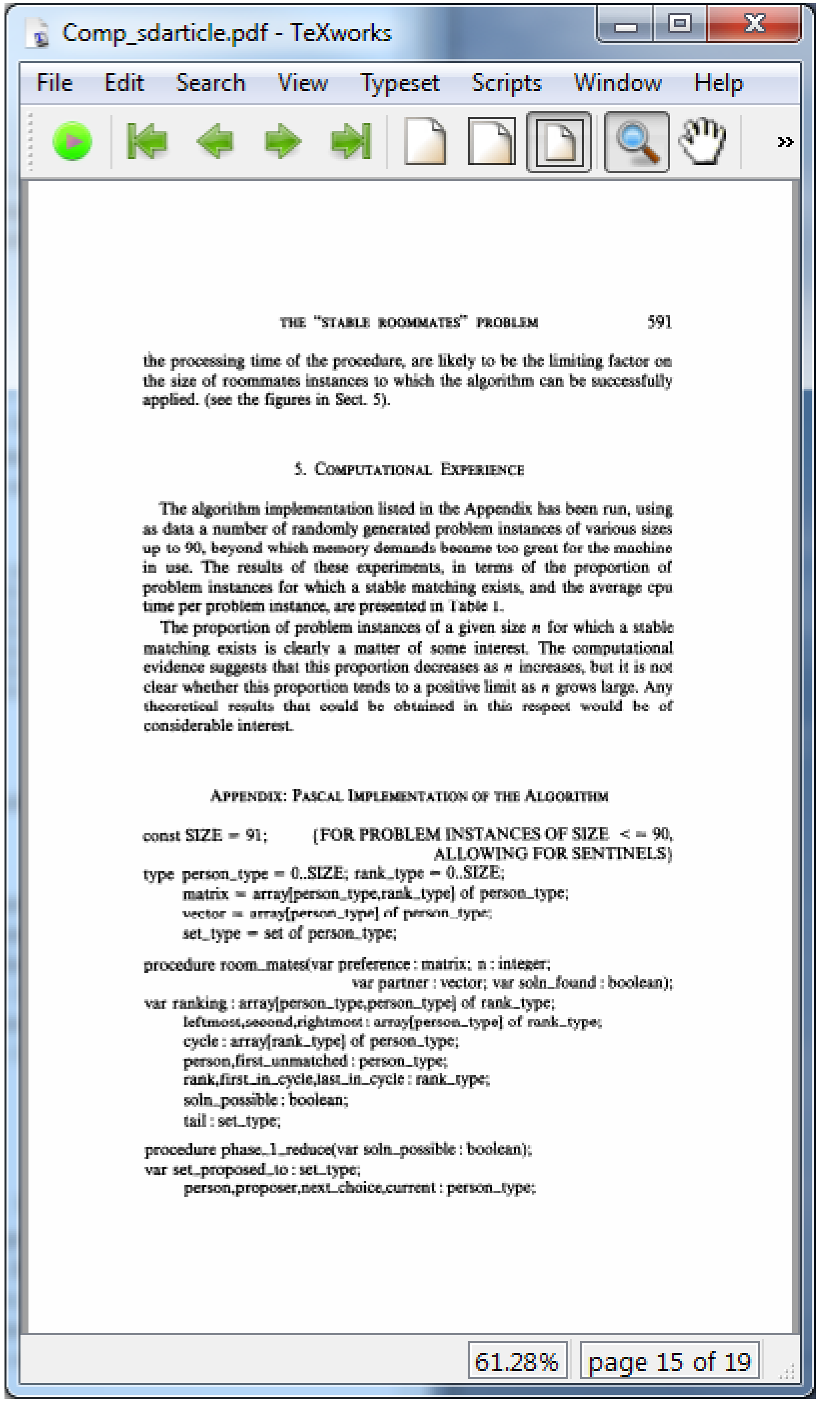
Taken from "***The green book***"

Stable Roommates (SR)

What's the problem?

1985

Code



the processing time of the procedure, are likely to be the limiting factor on the size of roommates instances to which the algorithm can be successfully applied. (see the figures in Sect. 5).

5. COMPUTATIONAL EXPERIENCE

The algorithm implementation listed in the Appendix has been run, using as data a number of randomly generated problem instances of various sizes up to 90, beyond which memory demands became too great for the machine in use. The results of these experiments, in terms of the proportion of problem instances for which a stable matching exists, and the average cpu time per problem instance, are presented in Table 1.

The proportion of problem instances of a given size n for which a stable matching exists is clearly a matter of some interest. The computational evidence suggests that this proportion decreases as n increases, but it is not clear whether this proportion tends to a positive limit as n grows large. Any theoretical results that could be obtained in this respect would be of considerable interest.

APPENDIX: PASCAL IMPLEMENTATION OF THE ALGORITHM

```
const SIZE = 91;        {FOR PROBLEM INSTANCES OF SIZE <= 90,
                                 ALLOWING FOR SENTINELS}
type person_type = 0..SIZE; rank_type = 0..SIZE;
     matrix = array[person_type,rank_type] of person_type;
     vector = array[person_type] of person_type;
     set_type = set of person_type;

procedure room_mates(var preference : matrix; n : integer;
                     var partner : vector; var soln_found : boolean);
var ranking : array[person_type,person_type] of rank_type;
     leftmost,second,rightmost : array[person_type] of rank_type;
     cycle : array[rank_type] of person_type;
     person,first_unmatched : person_type;
     rank,first_in_cycle,last_in_cycle : rank_type;
     soln_possible : boolean;
     tail : set_type;

procedure phase_1_reduce(var soln_possible : boolean);
var set_proposed_to : set_type;
     person,proposer,next_choice,current : person_type;
```

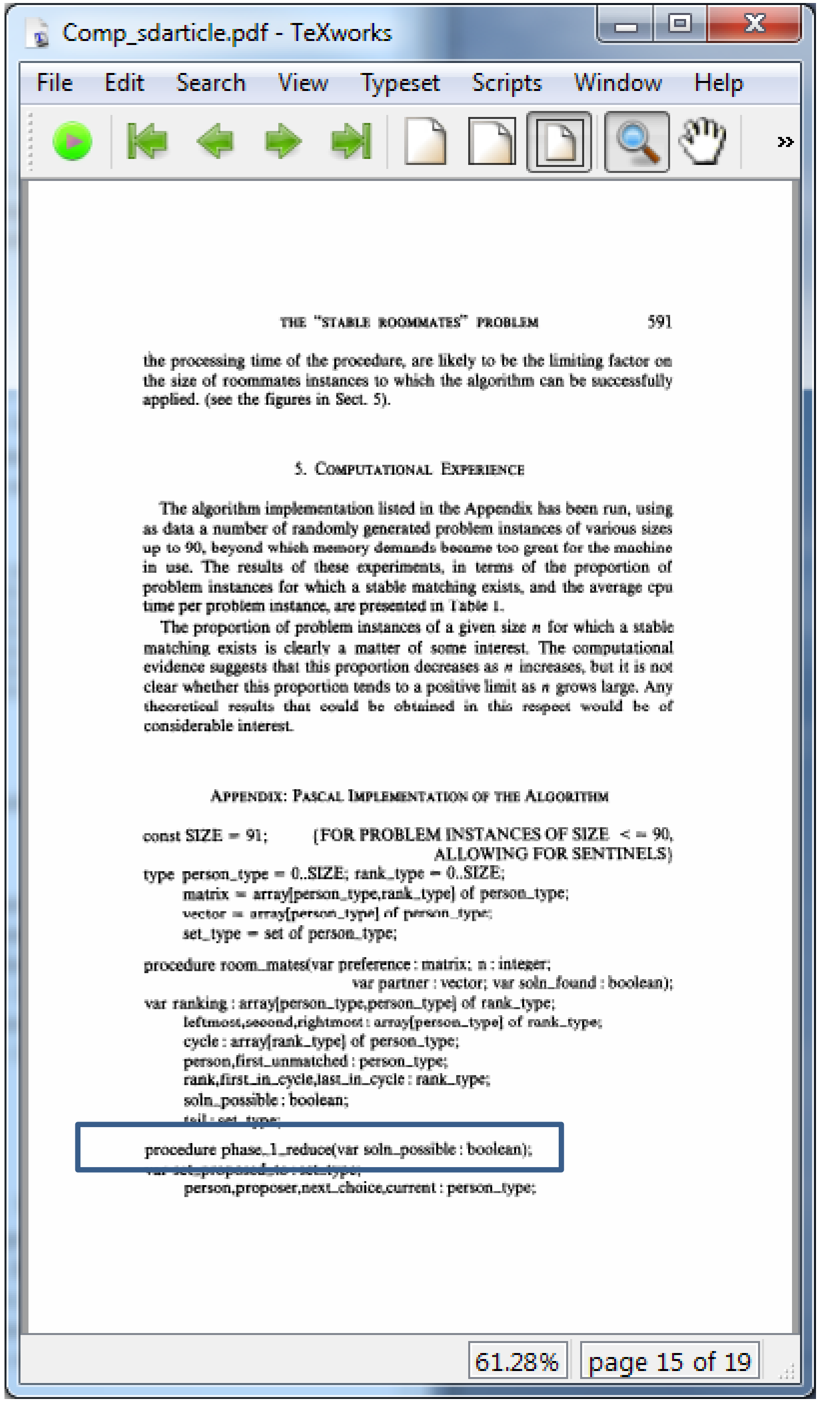
The Algorithm (Pascal)

Stable Roommates (SR)

What's the problem?

1985

Code



THE "STABLE ROOMMATES" PROBLEM 591

the processing time of the procedure, are likely to be the limiting factor on the size of roommates instances to which the algorithm can be successfully applied. (see the figures in Sect. 5).

5. COMPUTATIONAL EXPERIENCE

The algorithm implementation listed in the Appendix has been run, using as data a number of randomly generated problem instances of various sizes up to 90, beyond which memory demands became too great for the machine in use. The results of these experiments, in terms of the proportion of problem instances for which a stable matching exists, and the average cpu time per problem instance, are presented in Table 1.

The proportion of problem instances of a given size n for which a stable matching exists is clearly a matter of some interest. The computational evidence suggests that this proportion decreases as n increases, but it is not clear whether this proportion tends to a positive limit as n grows large. Any theoretical results that could be obtained in this respect would be of considerable interest.

APPENDIX: PASCAL IMPLEMENTATION OF THE ALGORITHM

```
const SIZE = 91;        {FOR PROBLEM INSTANCES OF SIZE  < = 90,
                                     ALLOWING FOR SENTINELS}
type person_type = 0..SIZE; rank_type = 0..SIZE;
      matrix = array[person_type,rank_type] of person_type;
      vector = array[person_type] of person_type;
      set_type = set of person_type;

procedure room_mates(var preference : matrix; n : integer;
                          var partner : vector; var soln_found : boolean);
var ranking : array[person_type,person_type] of rank_type;
      leftmost,second,rightmost : array[person_type] of rank_type;
      cycle : array[rank_type] of person_type;
      person,first_unmatched : person_type;
      rank,first_in_cycle,last_in_cycle : rank_type;
      soln_possible : boolean;
      tail : set_type;

procedure phase_1_reduce(var soln_possible : boolean);
var set_proposed_to : set_type;
      person,proposer,next_choice,current : person_type;
```

The Algorithm (Pascal)

Stable Roommates (SR)

What's the problem?

Comp_sdarticle.pdf - TeXworks

File Edit Search View Typeset Scripts Window Help

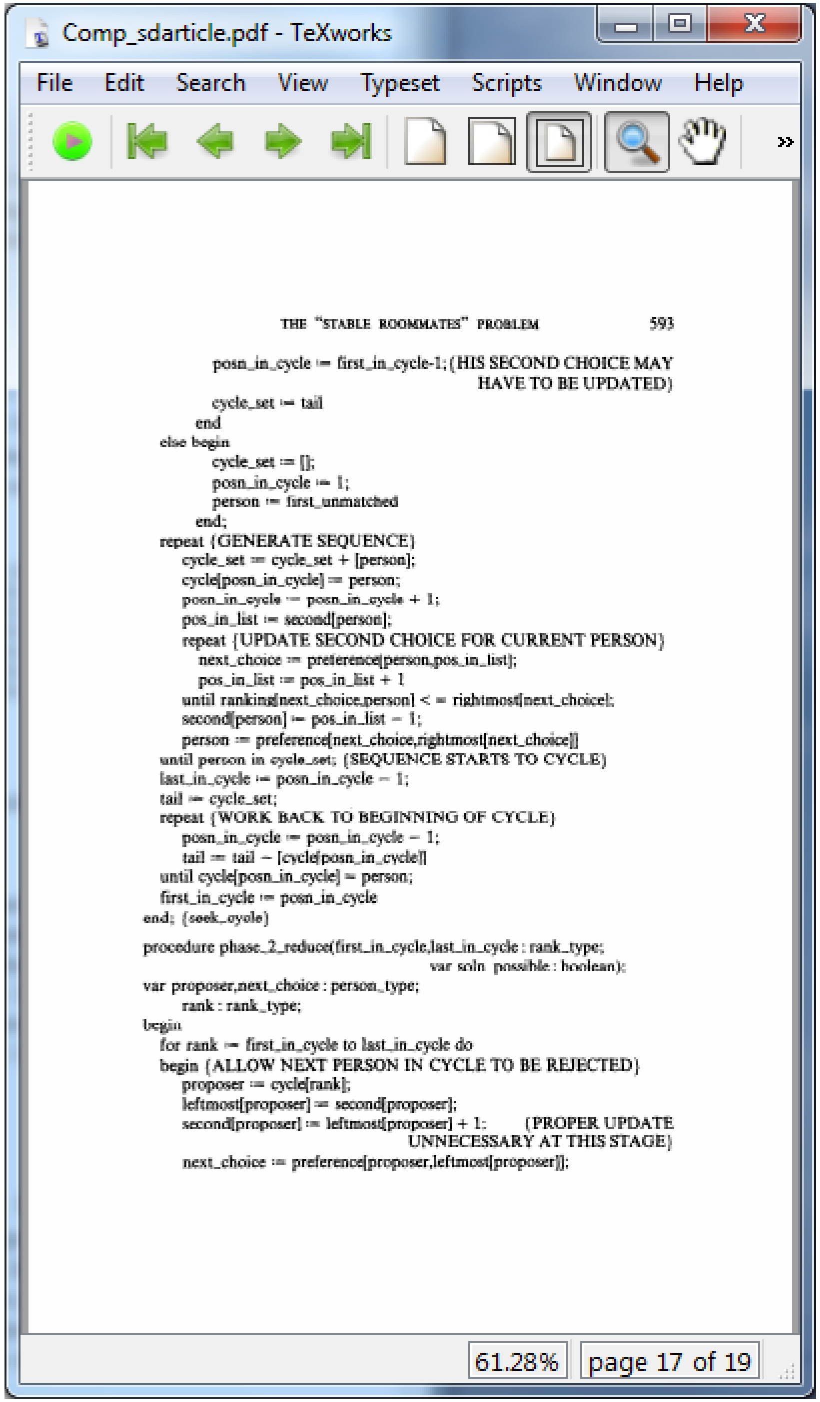

```
begin
   set_proposed_to := [];
   for person := 1 to n do
   begin
       proposer := person;
       repeat
         next_choice := preference[proposer,leftmost[proposer]];
                                              {BEST POTENTIAL PARTNER}
         current := preference[next_choice,rightmost[next_choice]];
                                              {NEXT_CHOICE HOLDS CURRENT}
         while  ranking[next_choice,proposer] > ranking[next_choice,current]
         do
         begin {PROPOSER IS REJECTED BY NEXT_CHOICE}
            leftmost[proposer] := leftmost[proposer] + 1;
            next_choice := preference[proposer,leftmost[proposer]];
            current := preference[next_choice,rightmost[next_choice]]
         end;
         rightmost[next_choice] := ranking[next_choice,proposer];
                                              {NEXT_CHOICE HOLDS PROPOSER}
         proposer := current
                                              {AND REJECTS CURRENT}
       until not (next_choice in set_proposed_to);
       set_proposed_to := set_proposed_to + [next_choice]
   end;
   soln_possible := proposer = next_choice
end; {phase_1_reduce}

procedure find(var first_unmatched : person_type);
begin {FINDS FIRST PERSON WITH > 1 POTENTIAL PARTNER}
   while leftmost[first_unmatched] = rightmost[first_unmatched] do
       first_unmatched := first_unmatched + 1
end; {find}

procedure seek_cycle(var first_in_cycle,last_in_cycle : rank_type;
                      first_unmatched : person_type; var tail : set_type);
var cycle_set : set_type;
       person,next_choice : person_type;
       posn_in_cycle,pos_in_list : rank_type;
begin
   if first_in_cycle > 1
   then begin
          person := cycle[first_in_cycle-1];          {LAST PERSON IN
                                                        PREVIOUS TAIL}
```

61.28% page 16 of 19

The Algorithm (Pascal)

Stable Roommates (SR)

What's the problem?



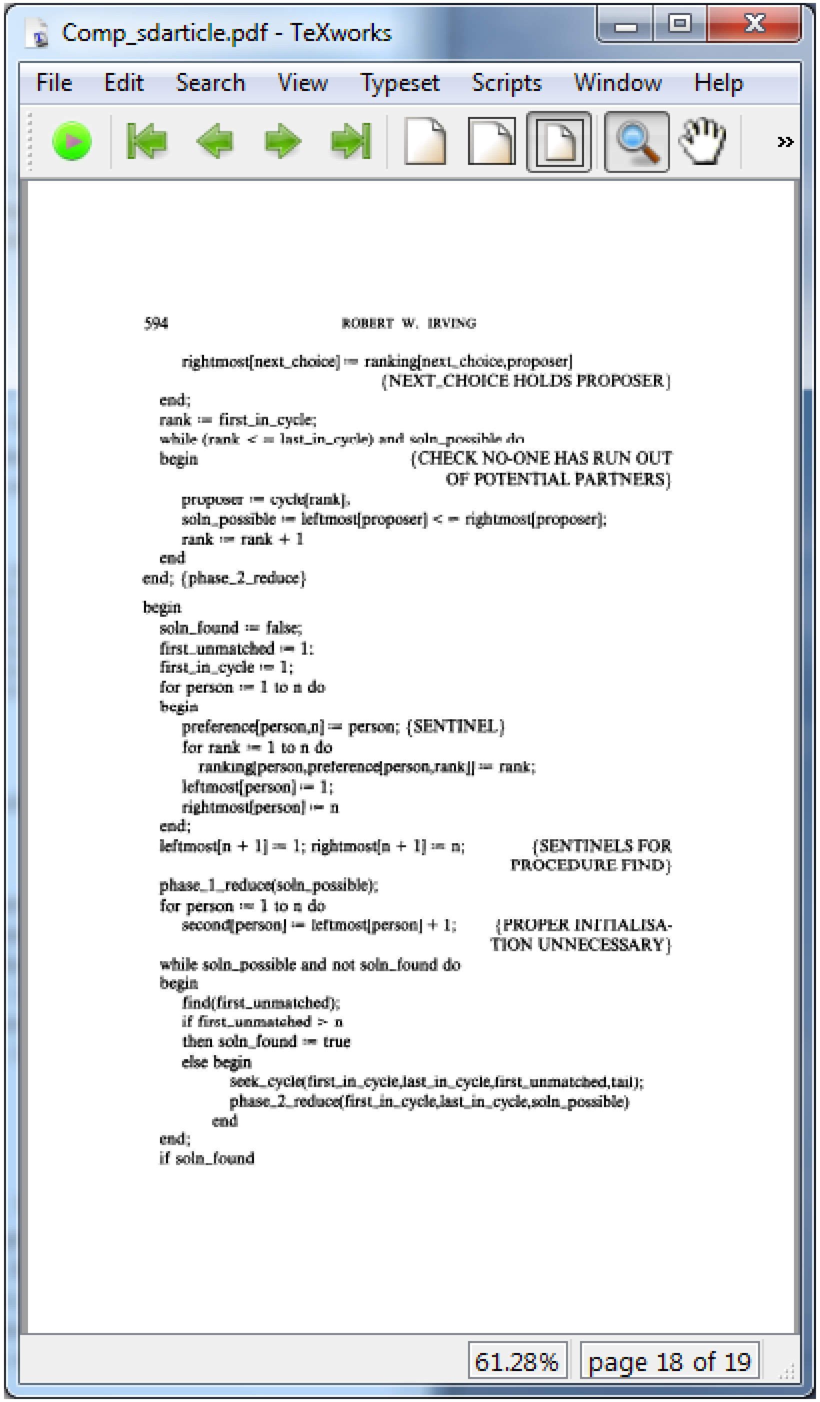
THE "STABLE ROOMMATES" PROBLEM 593

```
            posn_in_cycle := first_in_cycle-1; {HIS SECOND CHOICE MAY
                                              HAVE TO BE UPDATED}
            cycle_set := tail
         end
   else begin
            cycle_set := [];
            posn_in_cycle := 1;
            person := first_unmatched
         end;
   repeat {GENERATE SEQUENCE}
      cycle_set := cycle_set + [person];
      cycle[posn_in_cycle] := person;
      posn_in_cycle := posn_in_cycle + 1;
      pos_in_list := second[person];
      repeat {UPDATE SECOND CHOICE FOR CURRENT PERSON}
         next_choice := preference[person,pos_in_list];
         pos_in_list := pos_in_list + 1
      until ranking[next_choice,person] < = rightmost[next_choice];
      second[person] := pos_in_list - 1;
      person := preference[next_choice,rightmost[next_choice]]
   until person in cycle_set; {SEQUENCE STARTS TO CYCLE}
   last_in_cycle := posn_in_cycle - 1;
   tail := cycle_set;
   repeat {WORK BACK TO BEGINNING OF CYCLE}
      posn_in_cycle := posn_in_cycle - 1;
      tail := tail - [cycle[posn_in_cycle]]
   until cycle[posn_in_cycle] = person;
   first_in_cycle := posn_in_cycle
end; {seek_cycle}

procedure phase_2_reduce(first_in_cycle,last_in_cycle : rank_type;
                              var soln_possible : boolean);
var proposer,next_choice : person_type;
      rank : rank_type;
begin
   for rank := first_in_cycle to last_in_cycle do
   begin {ALLOW NEXT PERSON IN CYCLE TO BE REJECTED}
      proposer := cycle[rank];
      leftmost[proposer] := second[proposer];
      second[proposer] := leftmost[proposer] + 1;      {PROPER UPDATE
                                         UNNECESSARY AT THIS STAGE}
      next_choice := preference[proposer,leftmost[proposer]];
```

The Algorithm (Pascal)

Stable Roommates (SR)

What’s the problem?




```
      rightmost[next_choice] := ranking[next_choice,proposer]
                                        {NEXT_CHOICE HOLDS PROPOSER}
   end;
   rank := first_in_cycle;
   while (rank <= last_in_cycle) and soln_possible do
   begin                                   {CHECK NO-ONE HAS RUN OUT
                                              OF POTENTIAL PARTNERS}
      proposer := cycle[rank];
      soln_possible := leftmost[proposer] <= rightmost[proposer];
      rank := rank + 1
   end
end; {phase_2_reduce}

begin
   soln_found := false;
   first_unmatched := 1;
   first_in_cycle := 1;
   for person := 1 to n do
   begin
      preference[person,n] := person; {SENTINEL}
      for rank := 1 to n do
         ranking[person,preference[person,rank]] := rank;
      leftmost[person] := 1;
      rightmost[person] := n
   end;
   leftmost[n + 1] := 1; rightmost[n + 1] := n;        {SENTINELS FOR
                                                     PROCEDURE FIND}
   phase_1_reduce(soln_possible);
   for person := 1 to n do
      second[person] := leftmost[person] + 1;    {PROPER INITIALISA-
                                                   TION UNNECESSARY}
   while soln_possible and not soln_found do
   begin
      find(first_unmatched);
      if first_unmatched > n
      then soln_found := true
      else begin
            seek_cycle(first_in_cycle,last_in_cycle,first_unmatched,tail);
            phase_2_reduce(first_in_cycle,last_in_cycle,soln_possible)
         end
   end;
   if soln_found
```

The Algorithm (Pascal)

Stable Roommates (SR)

What's the problem?

Comp_sdarticle.pdf - TeXworks

File Edit Search View Typeset Scripts Window Help


```
    then for person := 1 to n do
            partner[person] := preference[person,leftmost[person]]
end; {room_mates}
```

Acknowledgment

The author is grateful to Professor D. E. Knuth for a number of helpful comments, particularly that which led to a more rigorous presentation of phase two of the algorithm, and that which improved the worst-case performance of procedure seek-cycle, and thereby of the whole algorithm, by an order of magnitude.

References

1. D. Gale and L. S. Shapley, College admissions and the stability of marriage, *Amer. Math. Monthly* **69** (1962), 9–15.
2. S. Y. Itoga, The upper bound for the stable marriage problem, *J. Oper. Res. Soc.* **29** (1978), 811–814.
3. D. E. Knuth, "Mariages Stables," Presses Univ. de Montréal, Montreal, 1976.
4. D.G. McVitie and L. B. Wilson, The stable marriage problem, *Comm. ACM* **14** (1971), 486–490.
5. D. G. McVitie and L. B. Wilson, Three procedures for the stable marriage problem, ACM Algorithm 411, *Comm. ACM* **14** (1971), 491–492.
6. L. B. Wilson, An analysis of the stable marriage assignment problem, *BIT* **12** (1972), 569–575.
7. L. B. Wilson, "A Survey of the Stable Marriage Assignment Problem," Department of Computing Science Technical Report 5, Univ. of Stirling, 1981.
8. N. Wirth, "Algorithms + Data Structures = Programs," Prentice-Hall, Englewood Cliffs, N.J., 1976.

61.28% page 19 of 19

The Algorithm (Pascal)

Stephan & Ciaran spotted something!

A simple constraint model

1	:	8	2	9	3	6	4	5	7	10
2	:	4	3	8	9	5	1	10	6	7
3	:	5	6	8	2	1	7	10	4	9
4	:	10	7	9	3	1	6	2	5	8
5	:	7	4	10	8	2	6	3	1	9
6	:	2	8	7	3	4	10	1	5	9
7	:	2	1	8	3	5	10	4	6	9
8	:	10	4	2	5	6	7	1	3	9
9	:	6	7	2	5	10	3	4	8	1
10	:	3	1	6	5	2	9	8	4	7

$pref_i$

Preference list for agent i

1	:	8	2	9	3	6	4	5	7	10
2	:	4	3	8	9	5	1	10	6	7
3	:	5	6	8	2	1	7	10	4	9
4	:	10	7	9	3	1	6	2	5	8
5	:	7	4	10	8	2	6	3	1	9
6	:	2	8	7	3	4	10	1	5	9
7	:	2	1	8	3	5	10	4	6	9
8	:	10	4	2	5	6	7	1	3	9
9	:	6	7	2	5	10	3	4	8	1
10	:	3	1	6	5	2	9	8	4	7

$pref_i$ Preference list for agent i

$pref_{i,k} = j$ agent j is agent i's kth choice

1 : 8 2 9 3 6 4 5 7 10
2 : 4 3 8 9 5 1 10 6 7
3 : 5 6 8 2 1 7 10 4 9
4 : 10 7 9 3 1 6 2 5 8
5 : 7 4 10 8 2 6 3 1 9
6 : 2 8 7 3 4 10 1 5 9
7 : 2 1 8 3 5 10 4 6 9
8 : 10 4 2 5 6 7 1 3 9
9 : 6 7 2 5 10 3 4 8 1
10 : 3 1 6 5 2 9 8 4 7

$pref_i$ Preference list for agent i

$pref_3 = 5,6,8,2,1,7,10,4,9$

Stable Roommates (SR)

A simple constraint model

1 : 8 2 9 3 6 4 5 7 10
2 : 4 3 8 9 5 1 10 6 7
3 : 5 6 8 2 $\boxed{1}$ 7 10 4 9
4 : 10 7 9 3 1 6 2 5 8
5 : 7 4 10 8 2 6 3 1 9
6 : 2 8 7 3 4 10 1 5 9
7 : 2 1 8 3 5 10 4 6 9
8 : 10 4 2 5 6 7 1 3 9
9 : 6 7 2 5 10 3 4 8 1
10 : 3 1 6 5 2 9 8 4 7

$pref_i$ Preference list for agent i

$pref_{3,5} = 1$ The 5th preference of agent 3 is agent 1

1 :	8 2 9 3 6 4 5 7 10
2 :	4 3 8 9 5 1 10 6 7
3 :	5 6 8 2 1 7 10 4 9
4 :	10 7 9 3 1 6 2 5 8
5 :	7 4 10 8 2 6 3 1 9
6 :	2 8 7 3 4 10 1 5 9
7 :	2 1 8 3 5 10 4 6 9
8 :	10 4 2 5 6 7 1 3 9
9 :	6 7 2 5 10 3 4 8 1
10 :	3 1 6 5 2 9 8 4 7

$pref_i$ Preference list for agent i

$pref_{i,k} = j$ agent j is agent i's kth choice

$rank_{i,j} = k$ agent j is agent i's kth choice

1 :	8 2 9 3 6 4 5 7 10
2 :	4 3 8 9 5 1 10 6 7
3 :	5 6 8 2 1 7 10 4 9
4 :	10 7 9 3 1 6 2 5 8
5 :	7 4 10 8 2 6 3 1 9
6 :	2 8 7 3 4 10 1 5 9
7 :	2 1 8 3 5 10 4 6 9
8 :	10 4 2 5 6 7 1 3 9
9 :	6 7 2 5 10 3 4 8 1
10 :	3 1 6 5 2 9 8 4 7

$pref_i$ Preference list for agent i

$pref_{i,k} = j$ agent j is agent i's kth choice

$rank_{i,j} = k$ agent j is agent i's kth choice

NOTE: a rank value that is low is a preferred choice (large numbers are bad)

1 : 8 2 9 3 6 4 5 7 10
2 : 4 3 8 9 5 1 10 6 7
3 : $\boxed{5}$ 6 8 2 1 7 10 4 9
4 : 10 7 9 3 1 6 2 5 8
5 : 7 4 10 8 2 6 3 1 9
6 : 2 8 7 3 4 10 1 5 9
7 : 2 1 8 3 5 10 4 6 9
8 : 10 4 2 5 6 7 1 3 9
9 : 6 7 2 5 10 3 4 8 1
10 : 3 1 6 5 2 9 8 4 7

$pref_i$ Preference list for agent i

$pref_{i,k} = j$ agent j is agent i's kth choice

$rank_{3,5} = 1$ agent 5 is agent 3's 1st choice

1 :	8 2 9 3 6 4 5 7 10
2 :	4 3 8 9 5 1 10 6 7
3 :	5 6 8 2 1 7 10 4 9
4 :	10 7 9 3 1 6 2 5 8
5 :	7 4 10 8 2 6 3 1 9
6 :	2 8 7 3 4 10 1 5 9
7 :	2 1 8 3 5 10 4 6 9
8 :	10 4 2 5 6 7 1 3 9
9 :	6 7 2 5 10 3 4 8 1
10 :	3 1 6 5 2 9 8 4 7

$pref_i$ Preference list for agent i

$pref_{i,k} = j$ agent j is agent i's kth choice

$rank_{3,5} = 1$ agent j is agent i's kth choice

$a_i \in \{1..n-1\}$ ***constrained integer variable*** agent i with a domain of ***ranks***

Stable Roommates (SR)

A simple constraint model

1	:	8	2	9	3	6	4	5	7	10
2	:	4	3	8	9	5	1	10	6	7
3	:	5	6	8	2	1	7	10	4	9
4	:	10	7	9	3	1	6	2	5	8
5	:	7	4	10	8	2	6	3	1	9
6	:	2	8	7	3	4	10	1	5	9
7	:	2	1	8	3	5	$\boxed{10}$	4	6	9
8	:	10	4	2	5	6	7	1	3	9
9	:	6	7	2	5	10	3	4	8	1
10	:	3	1	6	5	2	9	8	4	7

$pref_i$ Preference list for agent i

$pref_{i,k} = j$ agent j is agent i's kth choice

$rank_{3,5} = 1$ agent j is agent i's kth choice

$a_7 = 6$ agent 7 gets 6th choice and that is agent 10

Given two agents, i and j,
if agent i is matched to an agent he prefers less than agent j
then agent j must match up with an agent he prefers to agent i

Given two agents, i and j,
if agent i is matched to an agent he prefers less than agent j
then agent j must match up with an agent he prefers to agent i

$$\forall_{i \in [1..n]}\ a_i \in \{1..l_i + 1\} \tag{1}$$

$$\forall_{i \in [1..n]} \forall_{j \in pref_i}\ a_i > rank_{i,j} \implies a_j < rank_{j,i} \tag{2}$$

$$\forall_{i \in [1..n]} \forall_{j \in pref_i}\ a_i = rank_{i,j} \implies a_j = rank_{j,i} \tag{3}$$

Given two agents, i and j,
if agent i is matched to an agent he prefers less than agent j
then agent j must match up with an agent he prefers to agent i

$$\forall_{i \in [1..n]} \; a_i \in \{1..l_i + 1\} \tag{1}$$

$$\forall_{i \in [1..n]} \forall_{j \in pref_i} \; a_i > rank_{i,j} \implies a_j < rank_{j,i} \tag{2}$$

$$\forall_{i \in [1..n]} \forall_{j \in pref_i} \; a_i = rank_{i,j} \implies a_j = rank_{j,i} \tag{3}$$

(1) agent variables, actually we allow incomplete lists!

Given two agents, i and j,
if agent i is matched to an agent he prefers less than agent j
then agent j must match up with an agent he prefers to agent i

$$\forall_{i \in [1..n]} \; a_i \in \{1..l_i + 1\} \tag{1}$$

$$\forall_{i \in [1..n]} \forall_{j \in pref_i} \; a_i > rank_{i,j} \implies a_j < rank_{j,i} \tag{2}$$

$$\forall_{i \in [1..n]} \forall_{j \in pref_i} \; a_i = rank_{i,j} \implies a_j = rank_{j,i} \tag{3}$$

(1) agent variables, actually we allow incomplete lists!
(2) If agent i is matched to agent he prefers less than agent j
then agent j must match with someone better than agent i

Given two agents, i and j,
if agent i is matched to an agent he prefers less than agent j
then agent j must match up with an agent he prefers to agent i

$$\forall_{i \in [1..n]}\ a_i \in \{1..l_i + 1\} \tag{1}$$

$$\forall_{i \in [1..n]} \forall_{j \in pref_i}\ a_i > rank_{i,j} \implies a_j < rank_{j,i} \tag{2}$$

$$\forall_{i \in [1..n]} \forall_{j \in pref_i}\ a_i = rank_{i,j} \implies a_j = rank_{j,i} \tag{3}$$

(1) agent variables, actually we allow incomplete lists!
(2) If agent i is matched to agent he prefers less than agent j
then agent j must match with someone better than agent i
(3) If agent i is matched to agent j
then agent j is matched to agent i

1 : 8 2 9 3 $\boxed{6\ 4\ 5\ 7\ 10}$
2 : ~~4 3 8 9~~ 5 1 10 6 7
3 : $\boxed{5\ 6\ 8\ 2}$ 1 7 10 4 9
4 : 10 7 9 3 1 6 2 5 8
5 : 7 4 10 8 2 6 3 1 9
6 : 2 8 7 3 4 10 1 5 9
7 : 2 1 8 3 5 10 4 6 9
8 : 10 4 2 5 6 7 1 3 9
9 : 6 7 2 5 10 3 4 8 1
10 : 3 1 6 5 2 9 8 4 7

Given two agents, 1 and 3,
if agent 1 is matched to an agent he prefers less than agent 3
then agent 3 must match with an agent he prefers to agent 1

	1:	8	2	9	3	6	4	5	7	10
(2)	3:	5	6	8	2	1	7	10	4	9

$$\forall_{i\in[1..n]}\forall_{j\in pref_i}\ a_i > rank_{i,j} \implies a_j < rank_{j,i} \qquad (2)$$

1 : 8 2 9 3 6 4 5 7 10
2 : 4 3 8 9 5 1 10 6 7
3 : 5 6 8 2 1 7 10 4 9
4 : 10 7 9 3 1 6 2 5 8
5 : 7 4 10 8 2 6 3 1 9
6 : 2 8 7 3 4 10 1 5 9
7 : 2 1 8 3 5 10 4 6 9
8 : 10 4 2 5 6 7 1 3 9
9 : 6 7 2 5 10 3 4 8 1
10 : 3 1 6 5 2 9 8 4 7

Given two agents, 1 and 3,
if agent 1 is matched to agent 3
then agent 3 is matched to agent 1

(3)

1:	**8**	**2**	**9**	**3**	**6**	**4**	**5**	**7**	**10**
3:	**5**	**6**	**8**	**2**	**1**	**7**	**10**	**4**	**9**

$$\forall_{i \in [1..n]} \forall_{j \in pref_i} \; a_i = rank_{i,j} \implies a_j = rank_{j,i} \qquad (3)$$

choco

```java
public class StableRoommates {

  public static void main(String[] args) throws IOException {

      BufferedReader fin = new BufferedReader(new FileReader(args[0]));
      int n = Integer.parseInt(fin.readLine());
      int[][] pref = new int[n][n];
      int[][] rank = new int[n][n];
      int[] length = new int[n];
      for (int i=0;i<n;i++){
          StringTokenizer st = new StringTokenizer(fin.readLine()," ");
          int k = 0;
          length[i] = 0;
          while (st.hasMoreTokens()){
              int j = Integer.parseInt(st.nextToken()) - 1;
              rank[i][j] = k;
              pref[i][k] = j;
              length[i] = length[i] + 1;
              k = k + 1;
          }
          rank[i][i] = k;
          pref[i][k] = i;
      }
      fin.close();
      Model model = new CPModel();
      IntegerVariable[] a = new IntegerVariable[n];
      for (int i=0;i<n;i++) a[i] = makeIntVar("a_"+ i,0,length[i],"cp:enum");
      for (int i=0;i<n;i++)
          for (int j=0;j<length[i];j++){
              int k = pref[i][j];
              model.addConstraint(implies(gt(a[i],rank[i][k]),lt(a[k],rank[k][i])));
              model.addConstraint(implies(eq(a[i],rank[i][k]),eq(a[k],rank[k][i])));
          }
      Solver solver = new CPSolver();
      solver.read(model);
      if (solver.solve().booleanValue())
        for (int i=0;i<n;i++){
           int j = pref[i][solver.getVar(a[i]).getVal()];
           if (i<j) System.out.print("("+ (i+1) +","+ (j+1) +") ");
        }
      System.out.println();
  }
}
```

Listing 1. A simple encoding for SRI, StableRoommates.java

choco

```java
public class StableRoommates {

  public static void main(String[] args) throws IOException {

      BufferedReader fin = new BufferedReader(new FileReader(args[0]));
      int n = Integer.parseInt(fin.readLine());
      int[][] pref = new int[n][n];
      int[][] rank = new int[n][n];
      int[] length = new int[n];
      for (int i=0;i<n;i++){
          StringTokenizer st = new StringTokenizer(fin.readLine()," ");
          int k = 0;
          length[i] = 0;
          while (st.hasMoreTokens()){
              int j = Integer.parseInt(st.nextToken()) - 1;
              rank[i][j] = k;
              pref[i][k] = j;
              length[i]  = length[i] + 1;
              k = k + 1;
          }
          rank[i][i] = k;
          pref[i][k] = i;
      }
      fin.close();
      Model model = new CPModel();
      IntegerVariable[] a = new IntegerVariable[n];
      for (int i=0;i<n;i++) a[i] = makeIntVar("a_"+ i,0,length[i],"cp:enum");
      for (int i=0;i<n;i++)
          for (int j=0;j<length[i];j++){
              int k = pref[i][j];
              model.addConstraint(implies(gt(a[i],rank[i][k]),lt(a[k],rank[k][i])));
              model.addConstraint(implies(eq(a[i],rank[i][k]),eq(a[k],rank[k][i])));
          }
      Solver solver = new CPSolver();
      solver.read(model);
      if (solver.solve().booleanValue())
        for (int i=0;i<n;i++){
           int j = pref[i][solver.getVar(a[i]).getVal()];
           if (i<j) System.out.print("("+ (i+1) +","+ (j+1) +") ");
        }
      System.out.println();
  }
}
```

Read in the problem

Listing 1. A simple encoding for SRI, StableRoommates.java

choco

```java
public class StableRoommates {

  public static void main(String[] args) throws IOException {

      BufferedReader fin = new BufferedReader(new FileReader(args[0]));
      int n = Integer.parseInt(fin.readLine());
      int[][] pref = new int[n][n];
      int[][] rank = new int[n][n];
      int[] length = new int[n];
      for (int i=0;i<n;i++){
          StringTokenizer st = new StringTokenizer(fin.readLine()," ");
          int k = 0;
          length[i] = 0;
          while (st.hasMoreTokens()){
              int j = Integer.parseInt(st.nextToken()) - 1;
              rank[i][j] = k;
              pref[i][k] = j;
              length[i]  = length[i] + 1;
              k = k + 1;
          }
          rank[i][i] = k;
          pref[i][k] = i;
      }
      fin.close();
      Model model = new CPModel();
      IntegerVariable[] a = new IntegerVariable[n];
      for (int i=0;i<n;i++) a[i] = makeIntVar("a_"+ i,0,length[i],"cp:enum");
      for (int i=0;i<n;i++)
          for (int j=0;j<length[i];j++){
              int k = pref[i][j];
              model.addConstraint(implies(gt(a[i],rank[i][k]),lt(a[k],rank[k][i])));
              model.addConstraint(implies(eq(a[i],rank[i][k]),eq(a[k],rank[k][i])));
          }
      Solver solver = new CPSolver();
      solver.read(model);
      if (solver.solve().booleanValue())
        for (int i=0;i<n;i++){
           int j = pref[i][solver.getVar(a[i]).getVal()];
           if (i<j) System.out.print("("+ (i+1) +","+ (j+1) +") ");
        }
      System.out.println();
  }
}
```

Build the model

Listing 1. A simple encoding for SRI, StableRoommates.java

choco

```java
public class StableRoommates {

  public static void main(String[] args) throws IOException {

      BufferedReader fin = new BufferedReader(new FileReader(args[0]));
      int n = Integer.parseInt(fin.readLine());
      int[][] pref = new int[n][n];
      int[][] rank = new int[n][n];
      int[] length = new int[n];
      for (int i=0;i<n;i++){
          StringTokenizer st = new StringTokenizer(fin.readLine()," ");
          int k = 0;
          length[i] = 0;
          while (st.hasMoreTokens()){
              int j = Integer.parseInt(st.nextToken()) - 1;
              rank[i][j] = k;
              pref[i][k] = j;
              length[i] = length[i] + 1;
              k = k + 1;
          }
          rank[i][i] = k;
          pref[i][k] = i;
      }
      fin.close();
      Model model = new CPModel();
      IntegerVariable[] a = new IntegerVariable[n];
      for (int i=0;i<n;i++) a[i] = makeIntVar("a_"+ i,0,length[i],"cp:enum");
      for (int i=0;i<n;i++)
          for (int j=0;j<length[i];j++){
              int k = pref[i][j];
              model.addConstraint(implies(gt(a[i],rank[i][k]),lt(a[k],rank[k][i])));
              model.addConstraint(implies(eq(a[i],rank[i][k]),eq(a[k],rank[k][i])));
          }
      Solver solver = new CPSolver();
      solver.read(model);
      if (solver.solve().booleanValue())
        for (int i=0;i<n;i++){
           int j = pref[i][solver.getVar(a[i]).getVal()];
           if (i<j) System.out.print("("+ (i+1) +","+ (j+1) +") ");
        }
      System.out.println();
  }
}
```

Find and print first matching

Listing 1. A simple encoding for SRI, StableRoommates.java

Neat

☺

Can model SMI as SRI

Ant: Bea, Ann, Cat

Bob: Bea, Cat, Ann

Cal: Ann, Bea, Cat

Ann: Bob, Ant, Cal

Bea: Cal, Ant, Bob

Cat: Cal, Bob, Ant

SM

men

1 : **1** 3 6 2 4 5
2 : 4 6 1 **2** 5 3
3 : 1 **4** 5 3 6 2
4 : **6** 5 3 4 2 1
5 : 2 3 1 4 **5** 6
6 : **3** 1 2 6 5 4

women

1 : **1** 5 6 3 2 4
2 : **2** 4 6 1 3 5
3 : 4 3 **6** 2 5 1
4 : 1 **3** 5 4 2 6
5 : 3 2 6 1 4 **5**
6 : 5 1 3 6 **4** 2

SRI

women+6

1 : **7** 9 12 8 10 11
2 : 10 12 7 **8** 11 9
3 : 7 **10** 11 9 12 8
4 : **12** 11 9 10 8 7
5 : 8 9 7 10 **11** 12
6 : **9** 7 8 12 11 10
7 : **1** 5 6 3 2 4
8 : **2** 4 6 1 3 5
9 : 4 3 **6** 2 5 1
10 1 **3** 5 4 2 6
11 3 2 6 1 4 **5**
12 5 1 3 6 **4** 2

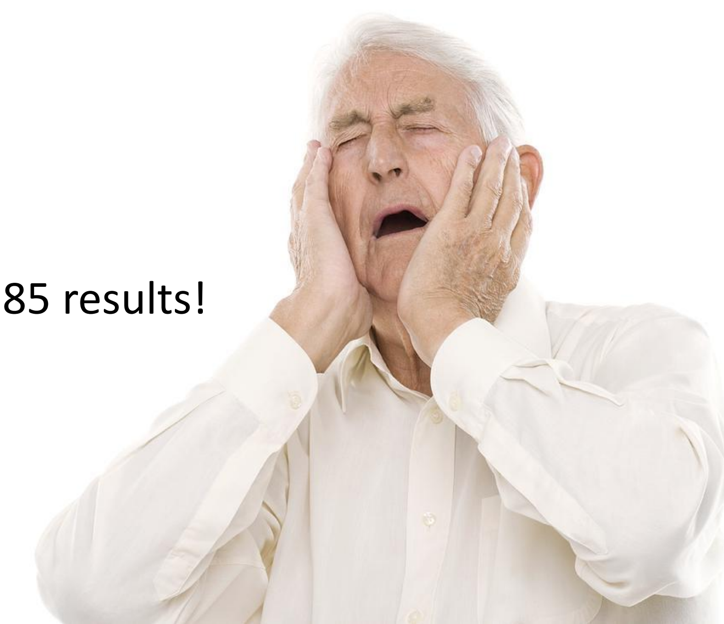
Yes, but what's new here?

Yes, but what's new here?

1. Model appeared twice in workshops
2. Applied to SM but not SR!
   (two sets of variables, more complicated)
3. One model for SM, SMI, SR & SRI
4. Simple & elegant

Yes, but what's new here?

1. Model appeared twice in workshops
2. Applied to SM but not SR!
   (two sets of variables, more complicated)
3. One model for SM, SMI, SR & SRI
4. Simple & elegant



But this is hard to believe … it is ***slower*** than Rob's 1985 results!

Yes, but what's new here?

1. Model appeared twice in workshops
2. Applied to SM but not SR!
   (two sets of variables, more complicated)
3. One model for SM, SMI, SR & SRI
4. Simple & elegant

But this is hard to believe … it is ***slower*** than Rob's 1985 results!

Cubic to achieve ***phase-1 table***

Not so neat

☹

A specialised constraint

When an agent's domain is filtered AC revises all constraints that involve that variable.

A specialised constraint

When an agent's domain is filtered AC revises all constraints that involve that variable.

In this case that is n constraints

A specialised constraint

When an agent's domain is filtered AC revises all constraints that involve that variable.

In this case that is n constraints

We can do better than this, reducing complexity of model by O(n)

I implemented a specialised binary SR constraint and an n-ary SR constraint
This deals with incomplete lists
This is presented in the paper
You can also download and run this

A specialised constraint

```java

public class SRN extends AbstractLargeIntSConstraint {

    private int n;
    private int[][] rank;
    private int[][] pref;
    private int[] length;
    private IStateInt[] uph;
    private IStateInt[] lwb;
    private IntDomainVar[] a;

    public SRN(Solver s,IntDomainVar[] a,int[][] pref,int[][] rank,int[] length) {
        super(a);
        n           = a.length;
        this.a      = a;
        this.pref   = pref;
        this.rank   = rank;
        this.length = length;
        uph         = new StoredInt[n];
        lwb         = new StoredInt[n];
        for (int i=0;i<n;i++){
            uph[i]  = s.getEnvironment().makeInt(length[i]);
            lwb[i]  = s.getEnvironment().makeInt(0);
        }
    }

    public void awake() throws ContradictionException {
        for (int i=0;i<n;i++) awakeOnInf(i);
    }

    public void propagate() throws ContradictionException {}

    public void awakeOnInf(int i) throws ContradictionException {
        int x = a[i].getInf(); // best (lowest) rank for a i
        int j = pref[i][x];
        a[j].setSup(rank[j][i]);
        for (int w=lwb[i].get();w<x;w++){
            int h = pref[i][w];
            a[h].setSup(rank[h][i]-1);
        }
        lwb[i].set(x);
    }

    public void awakeOnSup(int i) throws ContradictionException {
        int x = a[i].getSup(); // worst (largest) preference for a[i]
        for (int y=x+1;y<=uph[i].get();y++){
            int j = pref[i][y];
            a[j].remVal(rank[j][i]);
        }
        uph[i].set(x);
    }

    public void awakeOnRem(int i,int x) throws ContradictionException {
        int j = pref[i][x];
        a[j].remVal(rank[j][i]);
    }

    public void awakeOnInst(int i) throws ContradictionException {
        int y = a[i].getVal();
        for (int x = lwb[i].get();x<y;x++){
            int j = pref[i][x];
            a[j].setSup(rank[j][i]-1);
        }
        for (int x=y+1;x<=uph[i].get();x++){
            int j = pref[i][x];
            a[j].remVal(rank[j][i]);
        }
        lwb[i].set(y);
        uph[i].set(y);
    }
}
```

Listing 2. SRN.java

Here's the code.
Not much to it ☺

A specialised constraint

Constructor

```java
public class SRN extends AbstractLargeIntSConstraint {

    private int n;
    private int[][] rank;
    private int[][] pref;
    private int[] length;
    private IStateInt[] uph;
    private IStateInt[] lwh;
    private IntDomainVar[] a;

    public SRN(Solver s,IntDomainVar[] a,int[][] pref,int[][] rank,int[] length) {
        super(a);
        n           = a.length;
        this.a      = a;
        this.pref   = pref;
        this.rank   = rank;
        this.length = length;
        uph         = new StoredInt[n];
        lwh         = new StoredInt[n];
        for (int i=0;i<n;i++){
            uph[i]  = s.getEnvironment().makeInt(length[i]);
            lwh[i]  = s.getEnvironment().makeInt(0);
        }
    }

    public void awake() throws ContradictionException {
        for (int i=0;i<n;i++) awakeOnInf(i);
    }

    public void propagate() throws ContradictionException {}

    public void awakeOnInf(int i) throws ContradictionException {
        int x = a[i].getInf(); // best (lowest) rank for a i
        int j = pref[i][x];
        a[j].setSup(rank[j][i]);
        for (int w=lwh[i].get();w<x;w++){
            int h = pref[i][w];
            a[h].setSup(rank[h][i]-1);
        }
        lwh[i].set(x);
    }

    public void awakeOnSup(int i) throws ContradictionException {
        int x = a[i].getSup(); // worst (largest) preference for a[i]
        for (int y=x+1;y<=uph[i].get();y++){
            int j = pref[i][y];
            a[j].remVal(rank[j][i]);
        }
        uph[i].set(x);
    }

    public void awakeOnRem(int i,int x) throws ContradictionException {
        int j = pref[i][x];
        a[j].remVal(rank[j][i]);
    }

    public void awakeOnInst(int i) throws ContradictionException {
        int y = a[i].getVal();
        for (int x = lwh[i].get();x<y;x++){
            int j = pref[i][x];
            a[j].setSup(rank[j][i]-1);
        }
        for (int x=y+1;x<=uph[i].get();x++){
            int j = pref[i][x];
            a[j].remVal(rank[j][i]);
        }
        lwh[i].set(y);
        uph[i].set(y);
    }
}
```

Listing 2. SRN.java

A specialised constraint

```java

public class SRN extends AbstractLargeIntSConstraint {

    private int n;
    private int[][] rank;
    private int[][] pref;
    private int[] length;
    private IStateInt[] uph;
    private IStateInt[] lwb;
    private IntDomainVar[] a;

    public SRN(Solver s,IntDomainVar[] a,int[][] pref,int[][] rank,int[] length) {
        super(a);
        n           = a.length;
        this.a      = a;
        this.pref   = pref;
        this.rank   = rank;
        this.length = length;
        uph         = new StoredInt[n];
        lwb         = new StoredInt[n];
        for (int i=0;i<n;i++){
            uph[i]  = s.getEnvironment().makeInt(length[i]);
            lwb[i]  = s.getEnvironment().makeInt(0);
        }
    }

    public void awake() throws ContradictionException {
        for (int i=0;i<n;i++) awakeOnInf(i);
    }

    public void propagate() throws ContradictionException {}

    public void awakeOnInf(int i) throws ContradictionException {
        int x = a[i].getInf(); // best (lowest) rank for a i
        int j = pref[i][x];
        a[j].setSup(rank[j][i]);
        for (int w=lwb[i].get();w<x;w++){
            int h = pref[i][w];
            a[h].setSup(rank[h][i]-1);
        }
        lwb[i].set(x);
    }

    public void awakeOnSup(int i) throws ContradictionException {
        int x = a[i].getSup(); // worst (largest) preference for a[i]
        for (int y=x+1;y<=uph[i].get();y++){
            int j = pref[i][y];
            a[j].remVal(rank[j][i]);
        }
        uph[i].set(x);
    }

    public void awakeOnRem(int i,int x) throws ContradictionException {
        int j = pref[i][x];
        a[j].remVal(rank[j][i]);
    }

    public void awakeOnInst(int i) throws ContradictionException {
        int y = a[i].getVal();
        for (int x = lwb[i].get();x<y;x++){
            int j = pref[i][x];
            a[j].setSup(rank[j][i]-1);
        }
        for (int x=y+1;x<=uph[i].get();x++){
            int j = pref[i][x];
            a[j].remVal(rank[j][i]);
        }
        lwb[i].set(y);
        uph[i].set(y);
    }
}
```

Listing 2. SRN.java

awakening

A specialised constraint

lower bound changes

```java

public class SRN extends AbstractLargeIntSConstraint {

    private int n;
    private int[][] rank;
    private int[][] pref;
    private int[] length;
    private IStateInt[] uph;
    private IStateInt[] lwb;
    private IntDomainVar[] a;

    public SRN(Solver s,IntDomainVar[] a,int[][] pref,int[][] rank,int[] length) {
        super(a);
        n           = a.length;
        this.a      = a;
        this.pref   = pref;
        this.rank   = rank;
        this.length = length;
        uph         = new StoredInt[n];
        lwb         = new StoredInt[n];
        for (int i=0;i<n;i++){
            uph[i]  = s.getEnvironment().makeInt(length[i]);
            lwb[i]  = s.getEnvironment().makeInt(0);
        }
    }

    public void awake() throws ContradictionException {
        for (int i=0;i<n;i++) awakeOnInf(i);
    }

    public void propagate() throws ContradictionException {}

    public void awakeOnInf(int i) throws ContradictionException {
        int x = a[i].getInf(); // best (lowest) rank for a i
        int j = pref[i][x];
        a[j].setSup(rank[j][i]);
        for (int w=lwb[i].get();w<x;w++){
            int h = pref[i][w];
            a[h].setSup(rank[h][i]-1);
        }
        lwb[i].set(x);
    }

    public void awakeOnSup(int i) throws ContradictionException {
        int x = a[i].getSup(); // worst (largest) preference for a[i]
        for (int y=x+1;y<=uph[i].get();y++){
            int j = pref[i][y];
            a[j].remVal(rank[j][i]);
        }
        uph[i].set(x);
    }

    public void awakeOnRem(int i,int x) throws ContradictionException {
        int j = pref[i][x];
        a[j].remVal(rank[j][i]);
    }

    public void awakeOnInst(int i) throws ContradictionException {
        int y = a[i].getVal();
        for (int x = lwb[i].get();x<y;x++){
            int j = pref[i][x];
            a[j].setSup(rank[j][i]-1);
        }
        for (int x=y+1;x<=uph[i].get();x++){
            int j = pref[i][x];
            a[j].remVal(rank[j][i]);
        }
        lwb[i].set(y);
        uph[i].set(y);
    }
}
```

Listing 2. SRN.java

A specialised constraint

```java

public class SRN extends AbstractLargeIntSConstraint {

    private int n;
    private int[][] rank;
    private int[][] pref;
    private int[] length;
    private IStateInt[] uph;
    private IStateInt[] lwb;
    private IntDomainVar[] a;

    public SRN(Solver s,IntDomainVar[] a,int[][] pref,int[][] rank,int[] length) {
        super(a);
        n           = a.length;
        this.a      = a;
        this.pref   = pref;
        this.rank   = rank;
        this.length = length;
        uph         = new StoredInt[n];
        lwb         = new StoredInt[n];
        for (int i=0;i<n;i++){
            uph[i]  = s.getEnvironment().makeInt(length[i]);
            lwb[i]  = s.getEnvironment().makeInt(0);
        }
    }

    public void awake() throws ContradictionException {
        for (int i=0;i<n;i++) awakeOnInf(i);
    }

    public void propagate() throws ContradictionException {}

    public void awakeOnInf(int i) throws ContradictionException {
        int x = a[i].getInf(); // best (lowest) rank for a i
        int j = pref[i][x];
        a[j].setSup(rank[j][i]);
        for (int w=lwb[i].get();w<x;w++){
            int h = pref[i][w];
            a[h].setSup(rank[h][i]-1);
        }
        lwb[i].set(x);
    }

    public void awakeOnSup(int i) throws ContradictionException {
        int x = a[i].getSup(); // worst (largest) preference for a[i]
        for (int y=x+1;y<=uph[i].get();y++){
            int j = pref[i][y];
            a[j].remVal(rank[j][i]);
        }
        uph[i].set(x);
    }

    public void awakeOnRem(int i,int x) throws ContradictionException {
        int j = pref[i][x];
        a[j].remVal(rank[j][i]);
    }

    public void awakeOnInst(int i) throws ContradictionException {
        int y = a[i].getVal();
        for (int x = lwb[i].get();x<y;x++){
            int j = pref[i][x];
            a[j].setSup(rank[j][i]-1);
        }
        for (int x=y+1;x<=uph[i].get();x++){
            int j = pref[i][x];
            a[j].remVal(rank[j][i]);
        }
        lwb[i].set(y);
        uph[i].set(y);
    }
}
```

Listing 2. SRN.java

upper bound changes

A specialised constraint

removal of a value

```java

public class SRN extends AbstractLargeIntSConstraint {

    private int n;
    private int[][] rank;
    private int[][] pref;
    private int[] length;
    private IStateInt[] uph;
    private IStateInt[] lwb;
    private IntDomainVar[] a;

    public SRN(Solver s,IntDomainVar[] a,int[][] pref,int[][] rank,int[] length) {
        super(a);
        n           = a.length;
        this.a      = a;
        this.pref   = pref;
        this.rank   = rank;
        this.length = length;
        uph         = new StoredInt[n];
        lwb         = new StoredInt[n];
        for (int i=0;i<n;i++){
            uph[i]  = s.getEnvironment().makeInt(length[i]);
            lwb[i]  = s.getEnvironment().makeInt(0);
        }
    }

    public void awake() throws ContradictionException {
        for (int i=0;i<n;i++) awakeOnInf(i);
    }

    public void propagate() throws ContradictionException {}

    public void awakeOnInf(int i) throws ContradictionException {
        int x = a[i].getInf(); // best (lowest) rank for a i
        int j = pref[i][x];
        a[j].setSup(rank[j][i]);
        for (int w=lwb[i].get();w<x;w++){
            int h = pref[i][w];
            a[h].setSup(rank[h][i]-1);
        }
        lwb[i].set(x);
    }

    public void awakeOnSup(int i) throws ContradictionException {
        int x = a[i].getSup(); // worst (largest) preference for a[i]
        for (int y=x+1;y<=uph[i].get();y++){
            int j = pref[i][y];
            a[j].remVal(rank[j][i]);
        }
        uph[i].set(x);
    }

    public void awakeOnRem(int i,int x) throws ContradictionException {
        int j = pref[i][x];
        a[j].remVal(rank[j][i]);
    }

    public void awakeOnInst(int i) throws ContradictionException {
        int y = a[i].getVal();
        for (int x = lwb[i].get();x<y;x++){
            int j = pref[i][x];
            a[j].setSup(rank[j][i]-1);
        }
        for (int x=y+1;x<=uph[i].get();x++){
            int j = pref[i][x];
            a[j].remVal(rank[j][i]);
        }
        lwb[i].set(y);
        uph[i].set(y);
    }
}
```

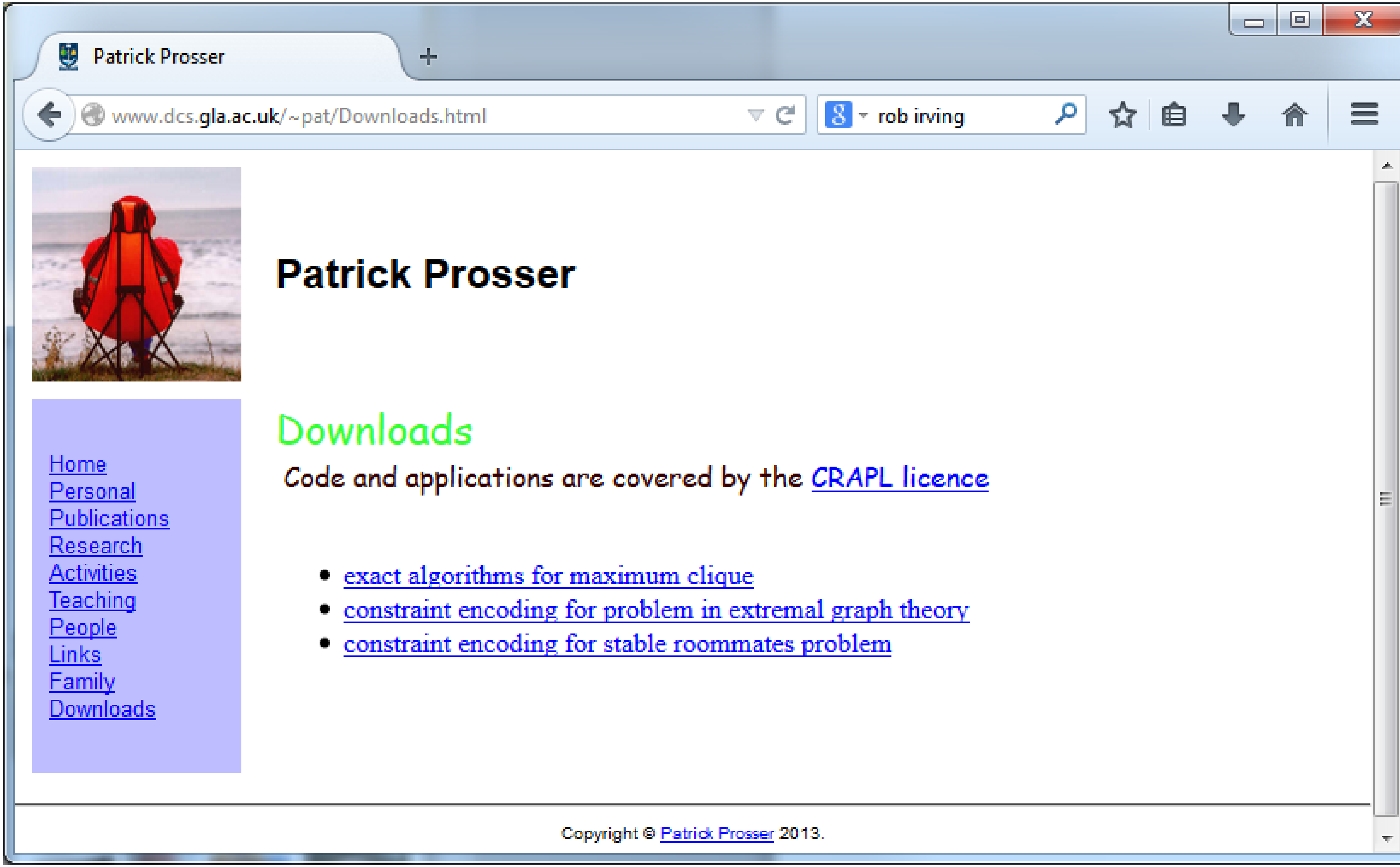
Listing 2. SRN.java

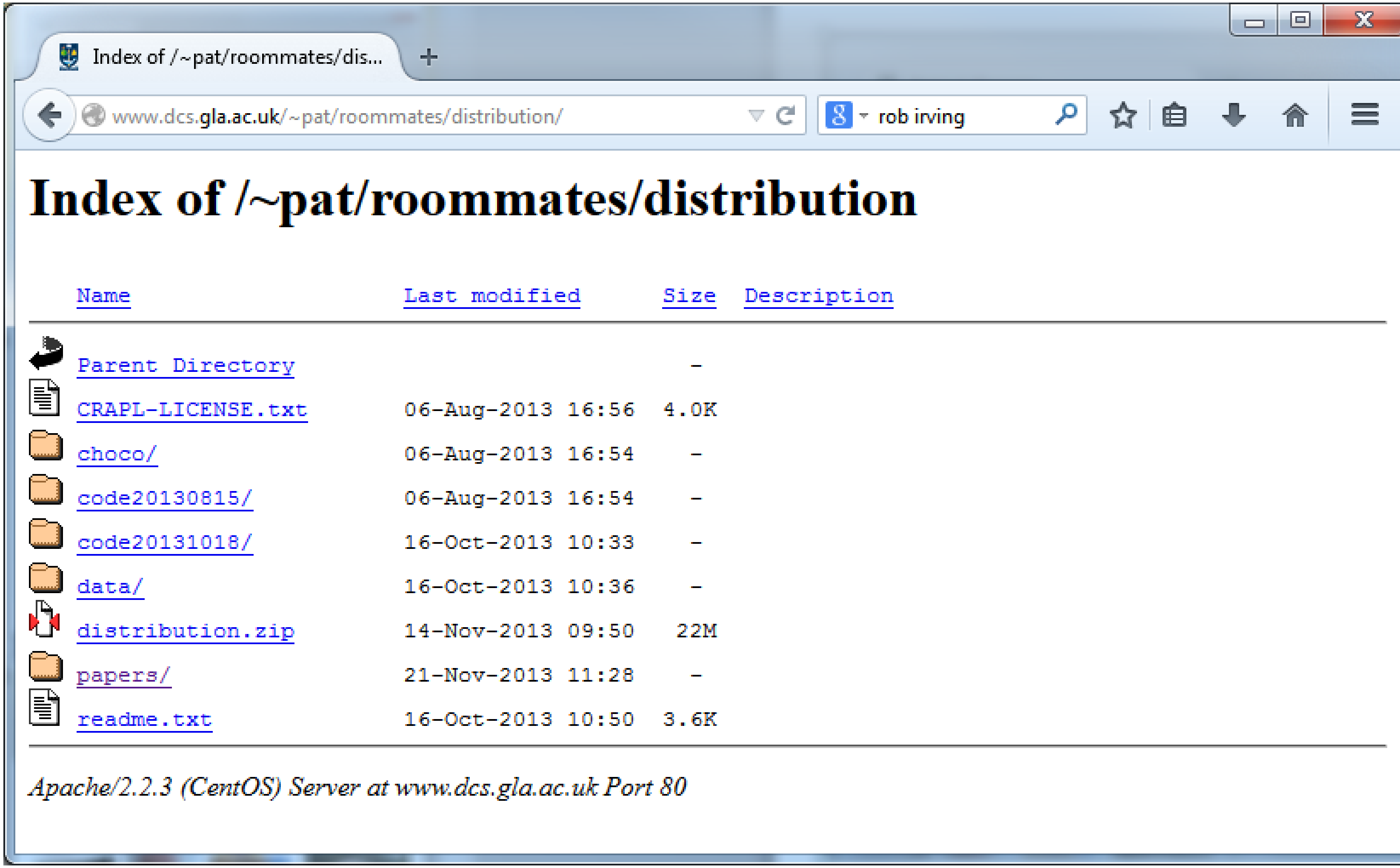
A specialised constraint

instantiate

```java

public class SRN extends AbstractLargeIntSConstraint {

    private int n;
    private int[][] rank;
    private int[][] pref;
    private int[] length;
    private IStateInt[] uph;
    private IStateInt[] lwb;
    private IntDomainVar[] a;

    public SRN(Solver s,IntDomainVar[] a,int[][] pref,int[][] rank,int[] length) {
        super(a);
        n           = a.length;
        this.a      = a;
        this.pref   = pref;
        this.rank   = rank;
        this.length = length;
        uph         = new StoredInt[n];
        lwb         = new StoredInt[n];
        for (int i=0;i<n;i++){
            uph[i]  = s.getEnvironment().makeInt(length[i]);
            lwb[i]  = s.getEnvironment().makeInt(0);
        }
    }

    public void awake() throws ContradictionException {
        for (int i=0;i<n;i++) awakeOnInf(i);
    }

    public void propagate() throws ContradictionException {}

    public void awakeOnInf(int i) throws ContradictionException {
        int x = a[i].getInf(); // best (lowest) rank for a i
        int j = pref[i][x];
        a[j].setSup(rank[j][i]);
        for (int w=lwb[i].get();w<x;w++){
            int h = pref[i][w];
            a[h].setSup(rank[h][i]-1);
        }
        lwb[i].set(x);
    }

    public void awakeOnSup(int i) throws ContradictionException {
        int x = a[i].getSup(); // worst (largest) preference for a[i]
        for (int y=x+1;y<=uph[i].get();y++){
            int j = pref[i][y];
            a[j].remVal(rank[j][i]);
        }
        uph[i].set(x);
    }

    public void awakeOnRem(int i,int x) throws ContradictionException {
        int j = pref[i][x];
        a[j].remVal(rank[j][i]);
    }

    public void awakeOnInst(int i) throws ContradictionException {
        int y = a[i].getVal();
        for (int x = lwb[i].get();x<y;x++){
            int j = pref[i][x];
            a[j].setSup(rank[j][i]-1);
        }
        for (int x=y+1;x<=uph[i].get();x++){
            int j = pref[i][x];
            a[j].remVal(rank[j][i]);
        }
        lwb[i].set(y);
        uph[i].set(y);
    }
}
```

Listing 2. SRN.java


Patrick Prosser
www.dcs.gla.ac.uk/~pat/Downloads.html
rob irving
Patrick Prosser
Home
Personal
Publications
Research
Activities
Teaching
People
Links
Family
Downloads
Downloads
Code and applications are covered by the CRAPL licence
exact algorithms for maximum clique
constraint encoding for problem in extremal graph theory
constraint encoding for stable roommates problem
Copyright © Patrick Prosser 2013.


Index of /~pat/roommates/dis...
www.dcs.gla.ac.uk/~pat/roommates/distribution/
rob irving
Index of /~pat/roommates/distribution
Name Last modified Size Description
Parent Directory -
CRAPL-LICENSE.txt 06-Aug-2013 16:56 4.0K
choco/ 06-Aug-2013 16:54 -
code20130815/ 06-Aug-2013 16:54 -
code20131018/ 16-Oct-2013 10:33 -
data/ 16-Oct-2013 10:36 -
distribution.zip 14-Nov-2013 09:50 22M
papers/ 21-Nov-2013 11:28 -
readme.txt 16-Oct-2013 10:50 3.6K
Apache/2.2.3 (CentOS) Server at www.dcs.gla.ac.uk Port 80

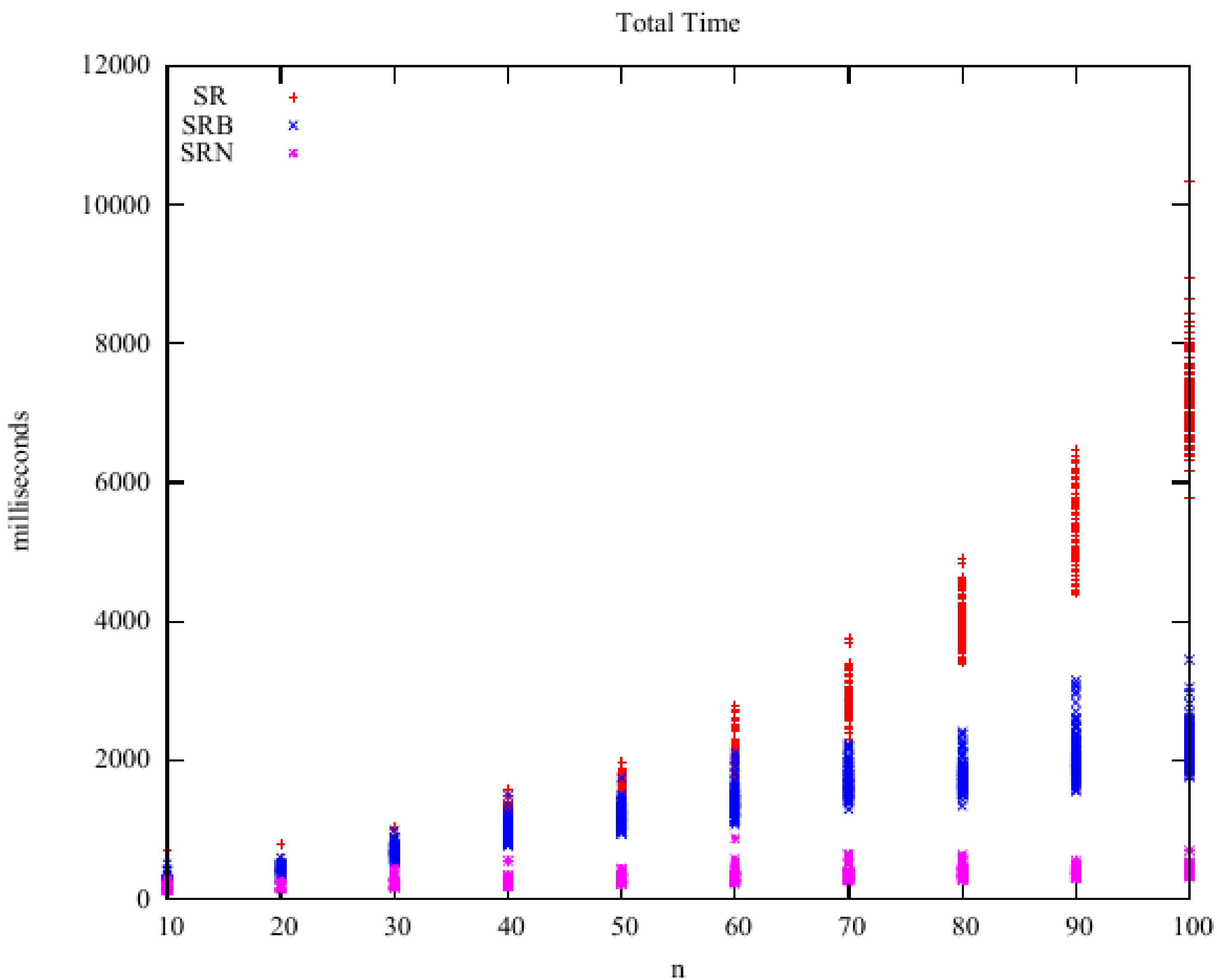
Empirical study

When I was younger, my mother did things to annoy me

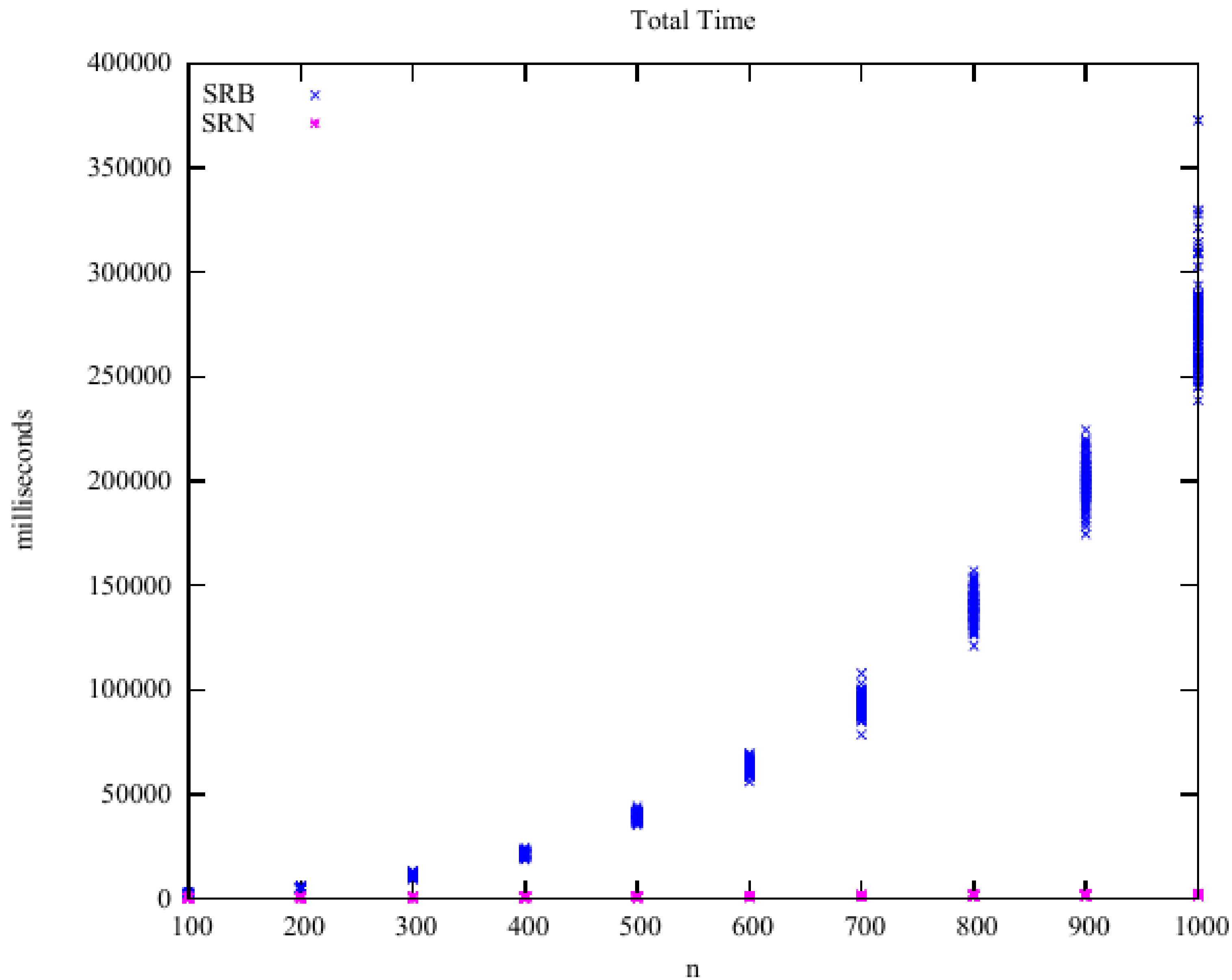
Empirical study

SR: ***simple*** constraint model, ***enumerated*** domains
SRB: ***simple*** constraint model, **bound** domains
SRN: ***specialised*** n-ary constraint, ***enumerated*** domains

10 < n < 100: read, build, find all stable matchings



100 < n < 1000: read, build, find all stable matchings



This is new (so says Rob and David)

n	cpu time	nodes	matched	max matchings
100	0.423	4 (17)	0.63	9
200	0.511	6 (34)	0.52	16
300	0.645	7 (33)	0.53	16
400	0.768	7 (25)	0.38	10
500	0.950	7 (35)	0.45	16
600	1.094	7 (27)	0.41	14
700	1.290	7 (31)	0.42	12
800	1.555	8 (50)	0.44	24
900	1.786	8 (29)	0.39	12
1,000	2.046	8 (85)	0.40	40

n, average run time, nodes (maximum), proportion with matchings, maximum number of matchings

So?

Well, think on this …

There are hard variants of SR. One example is egalitarian SR where a matching is to be found that minimizes the sum of the ranks, and this has been shown to be NP-hard [9]. In our constraint model an egalitarian matching is one that minimizes $\sum a_i$. Therefore we can model this problem by adding one more variable ($totalCost$), one more constraint ($totalCost = \sum a_i$) and a change from solving to minimization (line 36 of Listing 1). Naively, to find an egalitarian matching we could consider all matchings. As we see from Table 2 no instance had more than 40 matchings, no search took more than 85 nodes and the longest run time (not tabulated) was 2.6 seconds. Therefore, although NP-hard we would fail to encounter a hard instance in the problems sampled. So, (as Cheeseman, Kanefsky and Taylor famously asked [3]) where are the hard problems? As yet I do not know.

What’s still to do?

Prove that the model finds a stable matching in quadratic time …

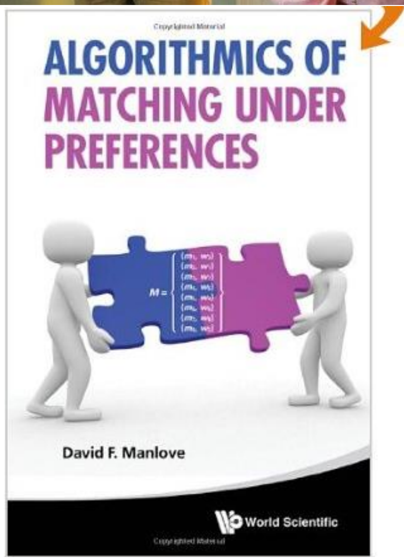
This was all my own work …

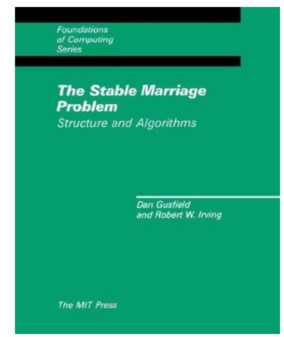
… well, with some help from

David Manlove
Rob Irving,
Jeremy Singer
Ian Gent
Chris Unsworth
Stephan Mertens
Ciaran McCreesh
Paul Cockshott
Joe Sventek
Augustine Kwanashie
Andrea






ALGORITHMICS OF MATCHING UNDER PREFERENCES
David F. Manlove
World Scientific



Foundations of Computing Series
The Stable Marriage Problem
Structure and Algorithms
Dan Gusfield and Robert W. Irving
The MIT Press