CSE 5311 Notes 10: Matching Under Preferences

(Last updated 1/12/21 11:47 AM)

A recent book - sample chapter 1 is useful (1.1.2.5, 1.3.5, 1.3.6, 1.4.5, 1.5.4-1.5.7 may be ignored):
`http://www.worldscientific.com/worldscibooks/10.1142/8591`

Book by Gusfield and Irving is on SEL reserve.

HIGH-LEVEL OVERVIEW

Matching under "agent" preferences under several broad scenarios will be considered :

1. Bipartite with preferences on both sides (arranging traditional marriages)
2. Non-bipartite matching with preferences (assigning roommates)
3. Bipartite one-sided preferences (housing allocation, DVD rentals, paper reviewing)

Goals: Understand mathematical properties and obtain centralized solution that is not trivially compromised.

Assumption: No exchange of money. (i.e. No salary negotiation. See p. 4 of
`http://www.nrmp.org/wp-content/uploads/2015/09/Applicant-Survey-Report-2015.pdf`)

STABLE MARRIAGES PROBLEM

Classical Problem Instance:

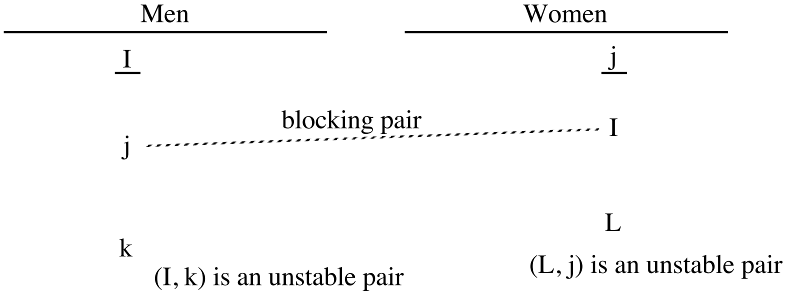
n men (A, B, C, . . .) with preference lists (ordered from most-preferred to least)

n women (1, 2, 3, . . .) with preference lists

Goal: Produce matching with *n* stable marriages.

A matching is *unstable* if there is a *blocking pair*:

Consider a matching with the pairs (I, k) and (L, j) based on preference lists:



I and j prefer each over their partners in the suggested matching . . . unstable situation

Applications:

Matching new M.D.s to internships (many-to-one, `http://www.nrmp.org/`)
Matching lawyers to federal clerkships (one-to-one)
Matching students to classes (many-to-many)
Centralized admissions decisions for universities (many-to-one)

Gale-Shapley (Deferred Acceptance) Algorithm

Corresponds to most societies. (No `https://en.wikipedia.org/wiki/Sadie_Hawkins_Day`)

Men propose from the beginning of their lists.

Women always accept the first proposal, but may break the engagement later.

Example (from Sedgewick)

A	B	C	D	E	1	2	3	4	5
2	1	2	1	5	E	D	A	C	D
5	2	3	**3**	3	**A**	**E**	**D**	**B**	B
1	3	**5**	2	**2**	D	B	B	D	**C**
3	**4**	4	4	1	B	A	C	A	E
4	5	1	5	4	C	C	E	E	A

Observations:

1. There is at least one stable solution.

   (Once engaged, a woman is always engaged. A man could eventually propose to all women and can't be rejected by all of them.)

2. The set of currently engaged couples is stable.

3. As stated, Gale-Shapley algorithm gives *male-optimal* matching. Switching roles in algorithm gives *female-optimal* matching. (Example of rotations includes female-optimal matching for Sedgewick's example)

4. If male-optimal solution is the same as female-optimal solution, the solution is unique.

5. The order of proposals by the available men *makes no difference* in the outcome . . . leading to:

The "Rural Hospitals" Theorem: When the number of men and women differ (or preference lists may be incomplete), the set of agents included in every stable matching *is the same*.

Also possible to maintain n^2 nodes in *reduced* data structure instead of $2n^2$ nodes (i.e. each node is in two doubly-linked lists) - known as the *Extended Gale-Shapley* algorithm (MEGS = man-oriented, WEGS = woman-oriented).

Uses node deletion strategy to avoid some pain of rejection! For MEGS:

Man proposes from current beginning of reduced list . . . always accepted!

When woman receives proposal . . . she will always accept and also delete the nodes for all less-preferable men.

For the current set of engagements:

A man is engaged to the woman at the beginning of his list.

A woman is engaged to the man at the end of her list.

A	B	C	D	E	1	2	3	4	5
2	1	2	1	5	E	D	A	C	D
5	2	3	**3**	3	**A**	**E**	**D**	**B**	B
1	3	**5**	2	**2**	D	B	B	D	**C**
3	**4**	4	4	1	B	A	C	A	E
4	5	1	5	4	C	C	E	E	A

Rotations and Lattice of Stable Marriage Solutions

A *rotation* takes two or more men, breaks their engagements, and engages them with the next (remaining) choice on their preference lists.

(An implementation may find the men after the first one by looking at the ends of the women's lists.)

Stability is maintained since all involved women become engaged with more preferable men.

Example: (truncated output from `http://ranger.uta.edu/~weems/NOTES5311/rotations.c`)

```
cat sedgewick.dat
5
2 5 1 3 4
1 2 3 4 5
2 3 5 4 1
1 3 2 4 5
5 3 2 1 4
5 1 4 2 3
4 5 2 1 3
1 4 2 3 5
3 2 4 1 5
4 2 3 5 1
a.out<sedgewick.dat
male preference lists are:
1: 1 3
2: 4 5
3: 5 4
4: 3 2 5
5: 2 1
female preference lists are:
1: 5 1
2: 4 5
3: 1 4
4: 3 2
5: 4 2 3
Male optimal solution:
1 1
2 4
3 5
4 3
5 2
Found a rotation:
(1,1)
(4,3)
(5,2)
Delete male=1 female=1
Delete male=4 female=3
Delete male=5 female=2
```

```
Revised preference lists:
male preference lists are:
1: 3
2: 4 5
3: 5 4
4: 2 5
5: 1
female preference lists are:
1: 5
2: 4
3: 1
4: 3 2
5: 4 2 3
Next matching:
1 3
2 4
3 5
4 2
5 1
Found a rotation:
(2,4)
(3,5)
Delete male=2 female=4
Delete male=3 female=5
Revised preference lists:
male preference lists are:
1: 3
2: 5
3: 4
4: 2 5
5: 1
female preference lists are:
1: 5
2: 4
3: 1
4: 3
5: 4 2
Next matching:
1 3
2 5
3 4
4 2
5 1
```

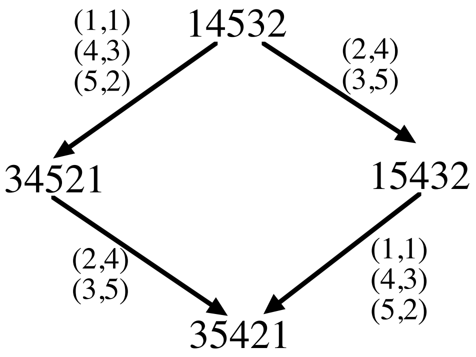
Last matching when iterating through rotations will be the female-optimal matching.

Given any pair of stable marriage matchings, another stable matching may be found by taking either:

1. The more preferred woman for every man (the "meet").

2. The less preferred woman for every man (the "join").

Mathematically, the result is a *distributive lattice*. (Also, note that any path from the male-optimal matching to the female-optimal matching includes each rotation exactly once.)

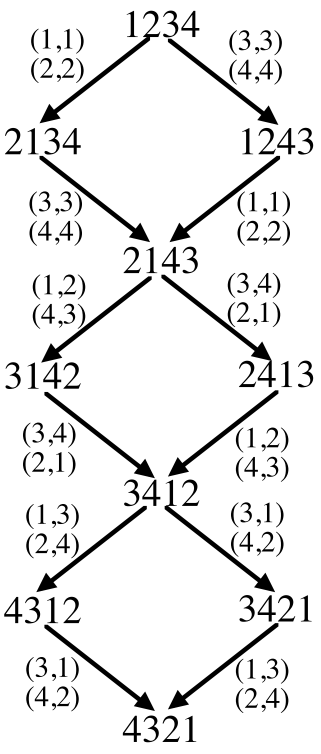


Example:

```
cat sm22.dat
4
1 2 3 4
2 1 4 3
3 4 1 2
4 3 2 1
4 3 2 1
3 4 1 2
2 1 4 3
1 2 3 4
a.out<sm22.dat
male preference lists are:
1: 1 2 3 4
2: 2 1 4 3
3: 3 4 1 2
4: 4 3 2 1
female preference lists are:
1: 4 3 2 1
2: 3 4 1 2
3: 2 1 4 3
4: 1 2 3 4
Male optimal solution:
1 1
2 2
3 3
4 4
Found a rotation:
(1,1)
(2,2)
Delete male=1 female=1
Delete male=2 female=2
Revised preference lists:
male preference lists are:
1: 2 3 4
2: 1 4 3
3: 3 4 1 2
4: 4 3 2 1
female preference lists are:
1: 4 3 2
2: 3 4 1
3: 2 1 4 3
4: 1 2 3 4
```

```
Next matching:
1 2
2 1
3 3
4 4
Found a rotation:
(3,3)
(4,4)
Delete male=3 female=3
Delete male=4 female=4
Revised preference lists:
male preference lists are:
1: 2 3 4
2: 1 4 3
3: 4 1 2
4: 3 2 1
female preference lists are:
1: 4 3 2
2: 3 4 1
3: 2 1 4
4: 1 2 3
Next matching:
1 2
2 1
3 4
4 3
Found a rotation:
(1,2)
(4,3)
Delete male=1 female=2
Delete male=4 female=3
Revised preference lists:
male preference lists are:
1: 3 4
2: 1 4 3
3: 4 1 2
4: 2 1
female preference lists are:
1: 4 3 2
2: 3 4
3: 2 1
4: 1 2 3
```

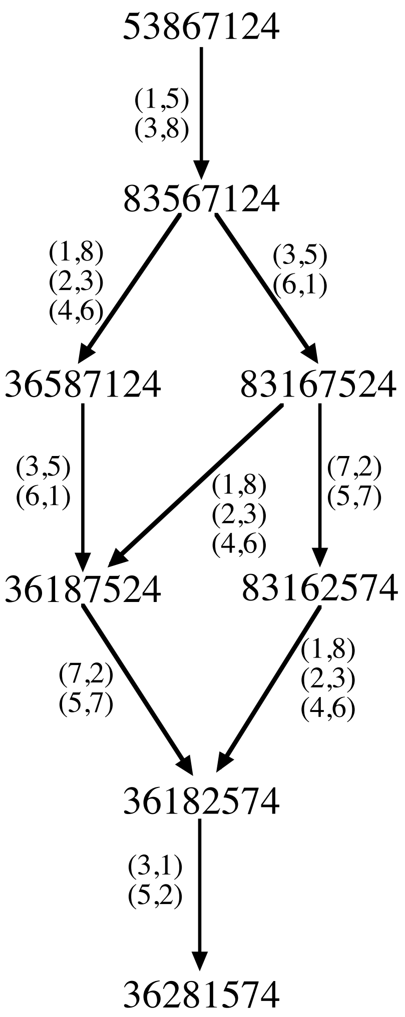
```
Next matching:
1 3
2 1
3 4
4 2
Found a rotation:
(3,4)
(2,1)
Delete male=3 female=4
Delete male=2 female=1
Revised preference lists:
male preference lists are:
1: 3 4
2: 4 3
3: 1 2
4: 2 1
female preference lists are:
1: 4 3
2: 3 4
3: 2 1
4: 1 2
Next matching:
1 3
2 4
3 1
4 2
Found a rotation:
(1,3)
(2,4)
Delete male=1 female=3
Delete male=2 female=4
Revised preference lists:
male preference lists are:
1: 4
2: 3
3: 1 2
4: 2 1
female preference lists are:
1: 4 3
2: 3 4
3: 2
4: 1
Next matching:
1 4
2 3
3 1
4 2
Found a rotation:
(3,1)
(4,2)
Delete male=3 female=1
Delete male=4 female=2
Revised preference lists:
male preference lists are:
1: 4
2: 3
3: 2
4: 1
female preference lists are:
1: 4
2: 3
3: 2
4: 1
Next matching:
1 4
2 3
3 2
4 1
```



Example:

```
cat sm12.dat
8
5 7 1 2 6 8 4 3
2 3 7 5 4 1 8 6
8 5 1 4 6 2 3 7
3 2 7 4 1 6 8 5
7 2 5 1 3 6 8 4
1 6 7 5 8 4 2 3
2 5 7 6 3 4 8 1
3 8 4 5 7 2 6 1
5 3 7 6 1 2 8 4
8 6 3 5 7 2 1 4
1 5 6 2 4 8 7 3
8 7 3 2 4 1 5 6
6 4 7 3 8 1 2 5
2 8 5 3 4 6 7 1
7 5 2 1 8 6 4 3
7 4 1 5 2 3 6 8
male preference lists are:
1: 5 8 3
2: 3 8 6
3: 8 5 1 6 2
4: 6 8 5
5: 7 2 1 3 6 8
6: 1 5 2 3
7: 2 5 7 8 1
8: 4 5 2 6
female preference lists are:
1: 5 3 7 6
2: 8 6 3 5 7
3: 1 5 6 2
4: 8
5: 6 4 7 3 8 1
6: 2 8 5 3 4
7: 7 5
8: 7 4 1 5 2 3
Male optimal solution:
1 5
2 3
3 8
4 6
5 7
6 1
7 2
8 4
Found a rotation:
(1,5)
(3,8)
Delete male=1 female=5
Delete male=8 female=5
Delete male=3 female=8
Delete male=2 female=8
Delete male=5 female=8
Revised preference lists:
male preference lists are:
1: 8 3
2: 3 6
3: 5 1 6 2
4: 6 8 5
5: 7 2 1 3 6
6: 1 5 2 3
7: 2 5 7 8 1
8: 4 2 6
female preference lists are:
1: 5 3 7 6
2: 8 6 3 5 7
3: 1 5 6 2
4: 8
5: 6 4 7 3
6: 2 8 5 3 4
7: 7 5
8: 7 4 1
Next matching:
1 8
2 3
3 5
4 6
5 7
6 1
7 2
8 4
Found a rotation:
(1,8)
(2,3)
(4,6)
Delete male=1 female=8
Delete male=2 female=3
Delete male=6 female=3
Delete male=5 female=3
Delete male=4 female=6
Delete male=3 female=6
Delete male=5 female=6
Delete male=8 female=6
Revised preference lists:
male preference lists are:
1: 3
2: 6
3: 5 1 2
4: 8 5
5: 7 2 1
6: 1 5 2
7: 2 5 7 8 1
8: 4 2
female preference lists are:
1: 5 3 7 6
2: 8 6 3 5 7
3: 1
4: 8
5: 6 4 7 3
6: 2
7: 7 5
8: 7 4
```

```
Next matching:
1 3
2 6
3 5
4 8
5 7
6 1
7 2
8 4
Found a rotation:
(3,5)
(6,1)
Delete male=3 female=5
Delete male=7 female=5
Delete male=4 female=5
Delete male=6 female=1
Delete male=7 female=1
Revised preference lists:
male preference lists are:
1: 3
2: 6
3: 1 2
4: 8
5: 7 2 1
6: 5 2
7: 2 7 8
8: 4 2
female preference lists are:
1: 5 3
2: 8 6 3 5 7
3: 1
4: 8
5: 6
6: 2
7: 7 5
8: 7 4
Next matching:
1 3
2 6
3 1
4 8
5 7
6 5
7 2
8 4
Found a rotation:
(7,2)
(5,7)
Delete male=7 female=2
Delete male=5 female=7
Revised preference lists:
male preference lists are:
1: 3
2: 6
3: 1 2
4: 8
5: 2 1
6: 5 2
7: 7 8
8: 4 2
female preference lists are:
1: 5 3
2: 8 6 3 5
3: 1
4: 8
5: 6
6: 2
7: 7
8: 7 4
Next matching:
1 3
2 6
3 1
4 8
5 2
6 5
7 7
8 4
Found a rotation:
(3,1)
(5,2)
Delete male=3 female=1
Delete male=5 female=2
Revised preference lists:
male preference lists are:
1: 3
2: 6
3: 2
4: 8
5: 1
6: 5 2
7: 7 8
8: 4 2
female preference lists are:
1: 5
2: 8 6 3
3: 1
4: 8
5: 6
6: 2
7: 7
8: 7 4
Next matching:
1 3
2 6
3 2
4 8
5 1
6 5
7 7
8 4
```



Generalizing to Hospitals/Residents (One-to-Many) or Many-to-Many

Gale-Shapley is straightforward to extend to support capacities

Rural Hospitals Theorem is more meaningful

Incomplete preference lists

Preference lists with ties

STABLE ROOMMATES - introductory concepts (aside)

Classical Problem Instance:

n persons with preference lists including the other *n* - 1 persons

n is assumed to be even

Goal: Produce list of $\frac{n}{2}$ stable pairs. (Still need to avoid blocking pairs.)

Generalizes stable marriages. An instance of S.M. is easily translated to an instance of S.R.

Two-phase algorithm is more complicated, since a solution is *not guaranteed*. (Aside: large, random instances seem to converge with < 25% having a solution.)

Phase 1: Like Gale-Shapley (with reduced lists), but based on finding *n* asymmetric *semi-engagements*.

x is semi-engaged to y means that x has issued a proposal to y, which y accepted. (`semiEngaged[i]=j` means that `j` is semi-engaged to `i`)

Phase 2: Uses rotations to assure that x is semi-engaged to y iff y is semi-engaged to x.

If some list becomes empty, then no solution.
Eliminating a rotation never precludes finding a solution.

(`http://ranger.uta.edu/~weems/NOTES5311/roommate.c`)

In rare cases of there being exactly one solution, the second phase may be skipped:

```
a.out<fig4.4.dat
Input:
1: 4 2 3
2: 1 3 4
3: 2 4 1
4: 2 1 3
debug: semiEngaged[2]=4
debug:  after processing semiengagement
1: 4 2 3
2: 1 3 4
3: 2 4 1
4: 2 1 3
debug: semiEngaged[2]=3
debug: delete {2 4} from lists
debug:  after processing semiengagement
1: 4 2 3
2: 1 3
3: 2 4 1
4: 1 3
debug: semiEngaged[1]=4
debug: delete {1 2} from lists
debug: delete {1 3} from lists
debug:  after processing semiengagement
1: 4
2: 3
3: 2 4
4: 1 3
debug: semiEngaged[3]=2
debug: delete {3 4} from lists
debug:  after processing semiengagement
1: 4
2: 3
3: 2
4: 1
debug: semiEngaged[4]=1
debug:  after processing semiengagement
1: 4
2: 3
3: 2
4: 1
After phase 1:
1: 4
2: 3
3: 2
4: 1
phase 2 not needed
```

In some cases there is no solution:

```
a.out<fig4.3.dat
Input:
1: 3 2 4
2: 1 3 4
3: 2 1 4
4: 1 2 3
debug: semiEngaged[1]=4
debug:  after processing semiengagement
1: 3 2 4
2: 1 3 4
3: 2 1 4
4: 1 2 3
debug: semiEngaged[2]=3
debug: delete {2 4} from lists
debug:  after processing semiengagement
1: 3 2 4
2: 1 3
3: 2 1 4
4: 1 3
debug: semiEngaged[1]=2
debug: delete {1 4} from lists
debug:  after processing semiengagement
1: 3 2
2: 1 3
3: 2 1 4
4: 3
```

```
debug: semiEngaged[3]=4
debug:  after processing semiengagement
1: 3 2
2: 1 3
3: 2 1 4
4: 3
debug: semiEngaged[3]=1
debug: delete {3 4} from lists
debug:  after processing semiengagement
1: 3 2
2: 1 3
3: 2 1
4:
phase 1 has empty list for 4 - no
solution exists
1: 3 2
2: 1 3
3: 2 1
4:
```

General case:

```
a.out<fig4.5.dat
Input:
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
debug: semiEngaged[3]=10
debug: delete {3 4} from lists
debug: delete {3 9} from lists
debug:  after processing semiengagement
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: 10 7 9 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 4 8 1
10: 3 1 6 5 2 9 8 4 7
debug: semiEngaged[6]=9
debug:  after processing semiengagement
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: 10 7 9 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 4 8 1
10: 3 1 6 5 2 9 8 4 7
debug: semiEngaged[10]=8
debug: delete {10 4} from lists
debug: delete {10 7} from lists
debug:  after processing semiengagement
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: 7 9 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 4 6 9
8: 10 4 2 5 6 7 1 3 9
9: 6 7 2 5 10 4 8 1
10: 3 1 6 5 2 9 8
debug: semiEngaged[2]=7
debug:  after processing semiengagement
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: 7 9 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 4 6 9
8: 10 4 2 5 6 7 1 3 9
9: 6 7 2 5 10 4 8 1
10: 3 1 6 5 2 9 8
debug: semiEngaged[2]=6
debug: delete {2 7} from lists
debug:  after processing semiengagement
1: 8 2 9 3 6 4 5 7 10
2: 4 3 8 9 5 1 10 6
3: 5 6 8 2 1 7 10
4: 7 9 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: 1 8 3 5 4 6 9
8: 10 4 2 5 6 7 1 3 9
9: 6 7 2 5 10 4 8 1
10: 3 1 6 5 2 9 8
debug: semiEngaged[1]=7
debug: delete {1 10} from lists
debug:  after processing semiengagement
1: 8 2 9 3 6 4 5 7
2: 4 3 8 9 5 1 10 6
3: 5 6 8 2 1 7 10
4: 7 9 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: 1 8 3 5 4 6 9
8: 10 4 2 5 6 7 1 3 9
9: 6 7 2 5 10 4 8 1
10: 3 6 5 2 9 8
debug: semiEngaged[7]=5
debug: delete {7 4} from lists
debug: delete {7 6} from lists
debug: delete {7 9} from lists
debug:  after processing semiengagement
1: 8 2 9 3 6 4 5 7
2: 4 3 8 9 5 1 10 6
3: 5 6 8 2 1 7 10
```

```
4: 9 1 6 2 5 8
5: 7 4 10 8 2 6 3 1 9
6: 2 8 3 4 10 1 5 9
7: 1 8 3 5
8: 10 4 2 5 6 7 1 3 9
9: 6 2 5 10 4 8 1
10: 3 6 5 2 9 8
debug: semiEngaged[9]=4
debug: delete {9 8} from lists
debug: delete {9 1} from lists
debug:  after processing semiengagement
1: 8 2 3 6 4 5 7
2: 4 3 8 9 5 1 10 6
3: 5 6 8 2 1 7 10
4: 9 1 6 2 5 8
5: 7 4 10 8 2 6 3 1 9
6: 2 8 3 4 10 1 5 9
7: 1 8 3 5
8: 10 4 2 5 6 7 1 3
9: 6 2 5 10 4
10: 3 6 5 2 9 8
debug: semiEngaged[5]=3
debug: delete {5 1} from lists
debug: delete {5 9} from lists
debug:  after processing semiengagement
1: 8 2 3 6 4 7
2: 4 3 8 9 5 1 10 6
3: 5 6 8 2 1 7 10
4: 9 1 6 2 5 8
5: 7 4 10 8 2 6 3
6: 2 8 3 4 10 1 5 9
7: 1 8 3 5
8: 10 4 2 5 6 7 1 3
9: 6 2 10 4
10: 3 6 5 2 9 8
debug: semiEngaged[4]=2
debug: delete {4 5} from lists
debug: delete {4 8} from lists
debug:  after processing semiengagement
1: 8 2 3 6 4 7
2: 4 3 8 9 5 1 10 6
3: 5 6 8 2 1 7 10
4: 9 1 6 2
5: 7 10 8 2 6 3
6: 2 8 3 4 10 1 5 9
7: 1 8 3 5
8: 10 2 5 6 7 1 3
9: 6 2 10 4
10: 3 6 5 2 9 8
debug: semiEngaged[8]=1
debug: delete {8 3} from lists
debug:  after processing semiengagement
1: 8 2 3 6 4 7
2: 4 3 8 9 5 1 10 6
3: 5 6 2 1 7 10
4: 9 1 6 2
5: 7 10 8 2 6 3
6: 2 8 3 4 10 1 5 9
7: 1 8 3 5
8: 10 2 5 6 7 1
9: 6 2 10 4
```

```
10: 3 6 5 2 9 8
After phase 1:
1: 8 2 3 6 4 7
2: 4 3 8 9 5 1 10 6
3: 5 6 2 1 7 10
4: 9 1 6 2
5: 7 10 8 2 6 3
6: 2 8 3 4 10 1 5 9
7: 1 8 3 5
8: 10 2 5 6 7 1
9: 6 2 10 4
10: 3 6 5 2 9 8
DEBUG-rotation: (1,8)(6,2)
debug: delete {2 6} from lists
debug: delete {2 10} from lists
debug: delete {8 1} from lists
debug: delete {8 7} from lists
1: 2 3 6 4 7
2: 4 3 8 9 5 1
3: 5 6 2 1 7 10
4: 9 1 6 2
5: 7 10 8 2 6 3
6: 8 3 4 10 1 5 9
7: 1 3 5
8: 10 2 5 6
9: 6 2 10 4
10: 3 6 5 9 8
DEBUG-rotation: (1,2)(10,3)(9,6)
debug: delete {6 9} from lists
debug: delete {6 5} from lists
debug: delete {6 1} from lists
debug: delete {3 10} from lists
debug: delete {3 7} from lists
debug: delete {2 1} from lists
debug: delete {2 5} from lists
1: 3 4 7
2: 4 3 8 9
3: 5 6 2 1
4: 9 1 6 2
5: 7 10 8 3
6: 8 3 4 10
7: 1 5
8: 10 2 5 6
9: 2 10 4
10: 6 5 9 8
DEBUG-rotation: (1,3)(2,4)
debug: delete {4 2} from lists
debug: delete {4 6} from lists
debug: delete {3 1} from lists
1: 4 7
2: 3 8 9
3: 5 6 2
4: 9 1
5: 7 10 8 3
6: 8 3 10
7: 1 5
8: 10 2 5 6
9: 2 10 4
10: 6 5 9 8
DEBUG-rotation: (8,10)(9,2)
debug: delete {2 9} from lists
```

```
debug: delete {10 8} from lists
1: 4 7
2: 3 8
3: 5 6 2
4: 9 1
5: 7 10 8 3
6: 8 3 10
7: 1 5
8: 2 5 6
9: 10 4
10: 6 5 9
DEBUG-rotation: (1,4)(5,7)(9,10)
debug: delete {10 9} from lists
debug: delete {7 5} from lists
debug: delete {4 1} from lists
1: 7
2: 3 8
3: 5 6 2
4: 9
5: 10 8 3
6: 8 3 10
7: 1
8: 2 5 6
9: 4
10: 6 5
DEBUG-rotation: (2,3)(6,8)
debug: delete {8 6} from lists
debug: delete {8 5} from lists
debug: delete {3 2} from lists
1: 7
2: 8
3: 5 6
4: 9
5: 10 3
6: 3 10
7: 1
8: 2
9: 4
10: 6 5
DEBUG-rotation: (3,5)(10,6)
debug: delete {6 10} from lists
debug: delete {5 3} from lists
phase 2 has solution
After phase 2:
1: 7
2: 8
3: 6
4: 9
5: 10
6: 3
7: 1
8: 2
9: 4
10: 5
```

This instance has a total of 7 solutions, based on processing rotations in different orders.

Underlying mathematical structure is more complicated, but efficient to deal with. (In 1976, Knuth speculated that deciding whether a S.R. instance had a solution might be NP-complete).

HOUSE ALLOCATION PROBLEM

Like Stable Marriages, two types of agents - *applicants* (with preferences) and *houses* (without)

No notion of blocking pair

Expects some applicants' preference lists to be incomplete

Several optimization criteria ("solution concepts") have been proposed (last three are asides):

Pareto optimality - A matching M is pareto optimal if it excludes any *Pareto improvements*:

Matching an unmatched applicant with an acceptable unmatched house

Changing an applicant to a more-preferable house without changing another applicant to a less-preferable house (or leaving out entirely)

Popularity - specializes Pareto optimal matchings by comparing the number of applicants who prefer one matching over another. Not guaranteed to exist.

Profile-based optimality - Each matching has a profile (vector) where the *k*th position is the number of applicants matched with their *k*th choice in their preference list. Matchings are compared by lexicographic comparison of their profiles. Leads to trade-offs on matchings' sizes.

Maximum utility - simple preference lists are replaced by a more general weighting scheme

Simplest way to find a Pareto optimal matching is the *Serial Dictatorship Mechanism*:

1. Choose an arbitrary order for the applicants.

2. Use the order to have each applicant choose their most-preferred house among the remaining unmatched houses.

Since the method is exhaustive and an early chooser would never trade with a later chooser, must be Pareto optimal.

Due to different orderings and incomplete preference lists, different size matchings may occur!

Finding a maximum cardinality Pareto optimal matching:

1. Find a maximum cardinality bipartite matching (e.g. using flow techniques or CLRS problem 26-6, p. 763), but *ignore the applicants' preferences*. (The next two phases are constructive proof that a Pareto optimal matching of this size exists. This phase is the most expensive. It is not worthwhile to clutter it with details of the two later phases.)

2. Make the matching "trade-in free" - iteratively, find a matched applicant having a more-preferred house that is available and promote the applicant.

3. Address "cyclic coalitions" in a manner similar to rotations for S.M. and S.R. The technique is known as Gale's Top Trading Cycles algorithm:

    a. Delete all unmatched houses.

    b. While applicants remain:

        1. Iteratively, find applicants matched with their most-preferred house. Make each match permanent and delete the applicant and house from data structures. (This may expose other most-preferred matches in the reduced instance.)

        2. Create directed graph:

            a. Vertex for each applicant.
            b. Edge from vertex for applicant x to the vertex of the applicant who is (tentatively) matched with x's <u>most-preferred</u> house.

        3. Properties of the generated graph:

            a. At least one cycle. (Similar to Boruvka's MST.)
            b. Cycles do not intersect! Simply eliminate each cycle.

Example 1:

After Phase 1

A1: H1 H3 $\boxed{\text{H2}}$

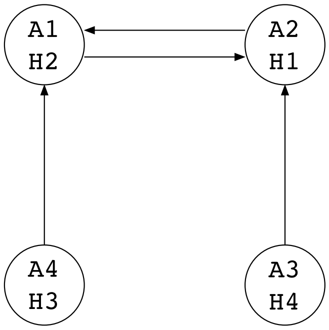
A2: H2 H4 $\boxed{\text{H1}}$

A3: H1 H2 $\boxed{\text{H4}}$

A4: H2 $\boxed{\text{H3}}$ H1

Phase 2 - no changes

Phase 3 graph



Final Result:

A1: $\boxed{\text{H1}}$

A2: $\boxed{\text{H2}}$

A3: $\boxed{\text{H4}}$

A4: $\boxed{\text{H3}}$

Example 2:

After Phase 1

A1: H3 H5 $\boxed{\text{H1}}$

A2: H1 H3 H4 $\boxed{\text{H2}}$

A3: H1 H4 H2 $\boxed{\text{H3}}$

A4: H1 H2 H3 $\boxed{\text{H4}}$

After Phase 2

A1: $\boxed{\text{H3}}$ H5 H1

A2: $\boxed{\text{H1}}$ H3 H4 H2

A3: H1 $\boxed{\text{H4}}$ H2 H3

A4: H1 $\boxed{\text{H2}}$ H3 H4

After Phase 3a

A1: $\boxed{\text{H3}}$ H1

A2: $\boxed{\text{H1}}$ H3 H4 H2

A3: H1 $\boxed{\text{H4}}$ H2 H3

A4: H1 $\boxed{\text{H2}}$ H3 H4

Phase 3b

The applicants will be deleted in the order: A1 A2 A3 A4

Final Result:

A1: $\boxed{\text{H3}}$

A2: $\boxed{\text{H1}}$

A3: $\boxed{\text{H4}}$

A4: $\boxed{\text{H2}}$

Example 3:

Find all maximum cardinality Pareto optimal matchings for:

A1: H1 H4 H2

A2: H2 H3 H1

A3: H1 H3 H4

A4: H2 H4 H3

Maximum cardinality matchings (Phase 1) - Pareto optimal ones are highlighted. The number below the non-Pareto-optimal ones is the number of Pareto-optimal ones that may be reached using later phases.

H1	H1	**H1**	H2	H2	H2	H4	**H4**	**H4**
H2	H2	**H3**	H1	H1	H3	H1	**H2**	**H3**
H3	H4	**H4**	H3	H4	H1	H3	**H1**	**H1**
H4	H3	**H2**	H4	H3	H4	H2	**H3**	**H2**
	1		1	1*	1	1		

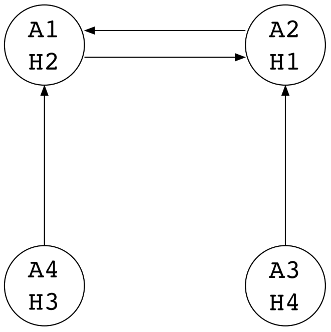
*Situation leading to 3.b iteration:

A1: H1 H4 $\boxed{\text{H2}}$

A2: H2 H3 $\boxed{\text{H1}}$

A3: H1 H3 $\boxed{\text{H4}}$

A4: H2 H4 $\boxed{\text{H3}}$

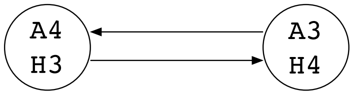


A1: $\boxed{\text{H1}}$

A2: $\boxed{\text{H2}}$

A3: H3 $\boxed{\text{H4}}$

A4: H4 $\boxed{\text{H3}}$



A1: $\boxed{\text{H1}}$

A2: $\boxed{\text{H2}}$

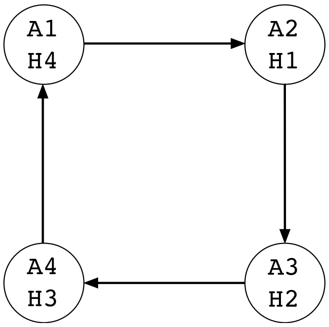
A3: $\boxed{\text{H3}}$

A4: $\boxed{\text{H4}}$

Example 4:

After Phases 1, 2, 3.a, 3.b.1, 3.b.2

A1:	H1	H2	H3	$\boxed{\text{H4}}$
A2:	H2	H3	H4	$\boxed{\text{H1}}$
A3:	H3	H4	H1	$\boxed{\text{H2}}$
A4:	H4	H1	H2	$\boxed{\text{H3}}$



ANSWER SET PROGRAMMING (aside)

A declarative programming paradigm with connections to logic programming, deductive databases, and knowledge representation. (`http://dl.acm.org.ezproxy.uta.edu/citation.cfm?doid=2043174.2043195`)

Potassco Implementation of ASP: `https://potassco.org/clingo/run/`

Examples: `http://ranger.uta.edu/~weems/NOTES5311/NOTES10.ASP/`

1. Replace declarations in first text area by an instance (e.g. `sm.sedgewick.lp`) and the matching rule encoding (e.g. `smi.enc.lp`).
2. Set “reasoning mode” to “enumerate all”.
3. Click “Run!”.