Optimality Criteria for Matching with One-Sided Preferences

Richard Matthew McCutchen
Advisor: Samir Khuller
University of Maryland

Problem

- Given an *instance*:
  - Set of people
  - Set of positions available to them
  - Each person's preference ordering of the positions
    - (Positions don't have preferences; that would be two-way)
- Compute the "best" matching of people to positions
- Applications
  - TAs to classes
  - Netflix customers to their next DVDs

Approach

- Different matchings inevitably favor different people ⇒ no obvious "best" matching
- Need an *optimality criterion*
  - An "optimal" matching should exist for every instance
  - Should be "fair"
  - Should be resistant to manipulation by people
  - Should admit an efficient algorithm to compute an optimal matching

Goal

- A computer program to solve real-world matching problems according to a good optimality criterion!
- Advantages
  - Fast/easy
  - Objective
  - Makes no mistakes

Example

	Cooking	Laundry	Dishes
Alice	1	2	3
Bob	1	3	2
Carol	3	1	2

- Three people, three positions
- Numbers indicate preference ranks

Example

	Co	La	Di
Alice	1	<u>**2**</u>	3
Bob	<u>**1**</u>	3	2
Carol	3	1	<u>**2**</u>

	Co	La	Di
Alice	<u>**1**</u>	2	3
Bob	1	3	<u>**2**</u>
Carol	3	<u>**1**</u>	2

- Which is better?

Example

	Co	La	Di
Alice	1	**2**	3
Bob	**1**	3	2
Carol	3	1	**2**

	Co	La	Di
Alice	**1**	2	3
Bob	1	3	**2**
Carol	3	**1**	2

- Compare by majority vote
- Right matching is “popular”

Why voting?

- +1 or –1; ignores the distance between two positions on a preference list
  - Arguably less fair
  - Seems to be accepted for elections for public office
- Using difference of numerical ranks opens door to easy manipulation
  - Person can pad preference list with positions he/she won’t get to make algorithm pity him/her
  - Students once exploited MIT housing algorithm this way
- Until we have a safer way to consider distance, stick with voting

Finding a popular matching

(Abraham, Irving, Kavitha, Mehlhorn; SODA 2005)

- A person's *backup* position: her favorite position that isn't anyone's first choice
- Theorem: A matching is popular iff:
  - Every position that is someone's first choice is filled, *and*
  - Each person gets either her first choice or her backup

Example:

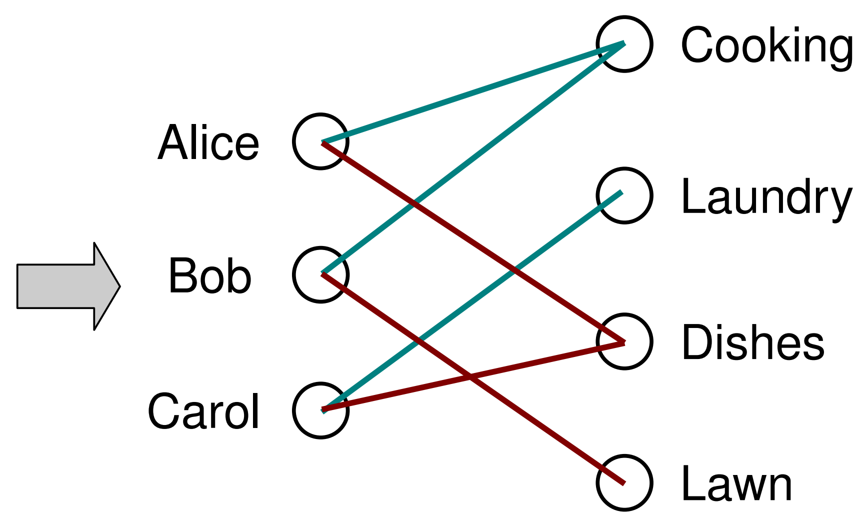
	Cooking	*Laundry*	Dishes	Lawn
Alice	***1***	*2*	**3***	4
Bob	***1***	*4*	3	**2***
Carol	*3*	***1***	**2***	4

Finding a popular matching

(Abraham, Irving, Kavitha, Mehlhorn; SODA 2005)

- Max-match in graph of first choices and backups, then promote people into any unfilled first choices

	Co	*Ld*	Di	Lw
Alice	*1*	*2*	3*	4
Bob	*1*	*4*	3	2*
Carol	*3*	*1*	2*	4

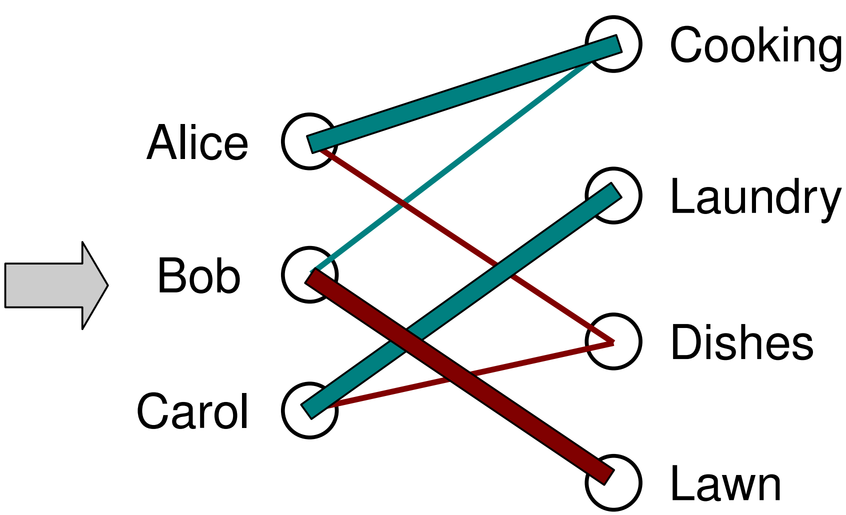


Finding a popular matching

(Abraham, Irving, Kavitha, Mehlhorn; SODA 2005)

- Max-match in graph of first choices and backups, then promote people into any unfilled first choices

	Co	*Ld*	Di	Lw
Alice	***1***	*2*	3*	4
Bob	*1*	*4*	3	**2***
Carol	*3*	***1***	2*	4



- More complicated algorithm works when preference orderings contain ties

No popular matching exists!

X

	Co	La	Di
Alice	**1**	2	3
Bob	1	**2**	3
Carol	1	2	**3**

Y

	Co	La	Di
Alice	1	2	**3**
Bob	**1**	2	3
Carol	1	**2**	3

Z

	Co	La	Di
Alice	1	**2**	3
Bob	1	2	**3**
Carol	**1**	2	3

Unpopularity factor

- Helps us choose decent matchings rather than terrible ones when no popular matching exists

Unpopularity factor

- Helps us choose decent matchings rather than terrible ones when no popular matching exists
- *N dominates M* by a factor of *u/v*, where:
  - *u* is # people better off in *N*
  - *v* is # people better off in *M*

Unpopularity factor

- Helps us choose decent matchings rather than terrible ones when no popular matching exists
- *N dominates M* by a factor of *u/v*, where:
  - *u* is # people better off in *N*
  - *v* is # people better off in *M*
- *Unpopularity factor* of *M*: Largest factor by which *M* is dominated by any other matching

Unpopularity factor

- Helps us choose decent matchings rather than terrible ones when no popular matching exists
- *N dominates M* by a factor of *u/v*, where:
  - *u* is # people better off in *N*
  - *v* is # people better off in *M*
- *Unpopularity factor* of *M*: Largest factor by which *M* is dominated by any other matching
- "Best" matching: least unpopularity factor
- Unpopularity factor ≤ 1 ⇔ popular

Example of U.F.

	Cooking	Laundry	Dishes	Cleaning
Alice	1	2	3	4
Bob	1	2	3	4
Carol	1	2	3	4
Dave	1	2	4	3

- No popular matching exists

Example of U.F.

M_1	Co	La	Di	Cl
Alice	**1**	2	3	4
Bob	1	2	**3**	4
Carol	1	2	3	**4**
Dave	1	**2**	4	3

N_1	Co	La	Di	Cl
Alice	1	2	3	**4**
Bob	1	**2**	3	4
Carol	1	2	**3**	4
Dave	**1**	2	4	3

- Unpopularity factor of $M_1 = 3$

Example of U.F.

M_2	Co	La	Di	Cl
Alice	**1**	2	3	4
Bob	1	**2**	3	4
Carol	1	2	**3**	4
Dave	1	2	4	**3**

N_2	Co	La	Di	Cl
Alice	1	2	**3**	4
Bob	**1**	2	3	4
Carol	1	**2**	3	4
Dave	1	2	4	**3**

- Unpopularity factor of $M_2 = 2$
- M_2 is better than M_1
- M_2 is in fact best

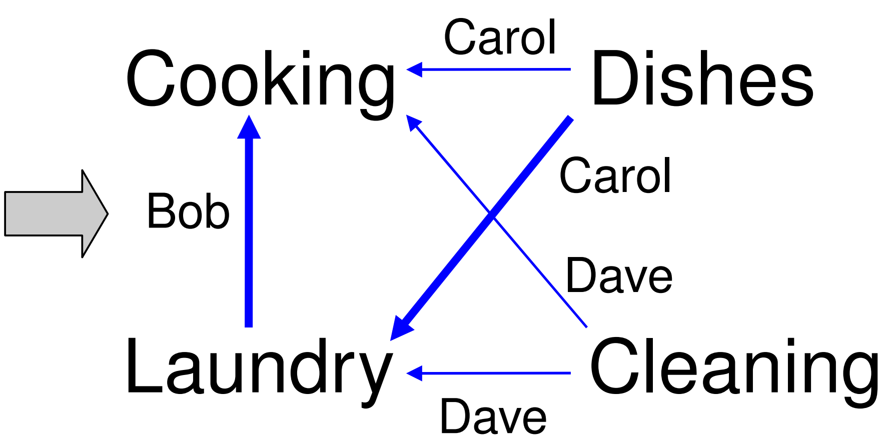
Results

- Easy to calculate unpopularity factor of a given matching
- NP-hard to find the “best” matching (least unpopularity factor)
  - Can still find it exhaustively for few people and positions

Pressures

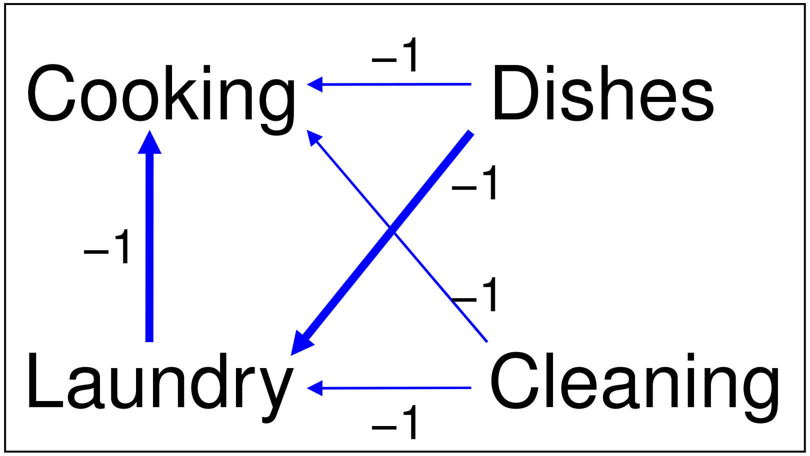
- *Pressure* of a position = # of people who can become better off if its occupant leaves
- Highest pressure = unpopularity factor

M_2	Co	La	Di	Cl
Alice	**1**	2	3	4
Bob	1	**2**	3	4
Carol	1	2	**3**	4
Dave	1	2	4	**3**



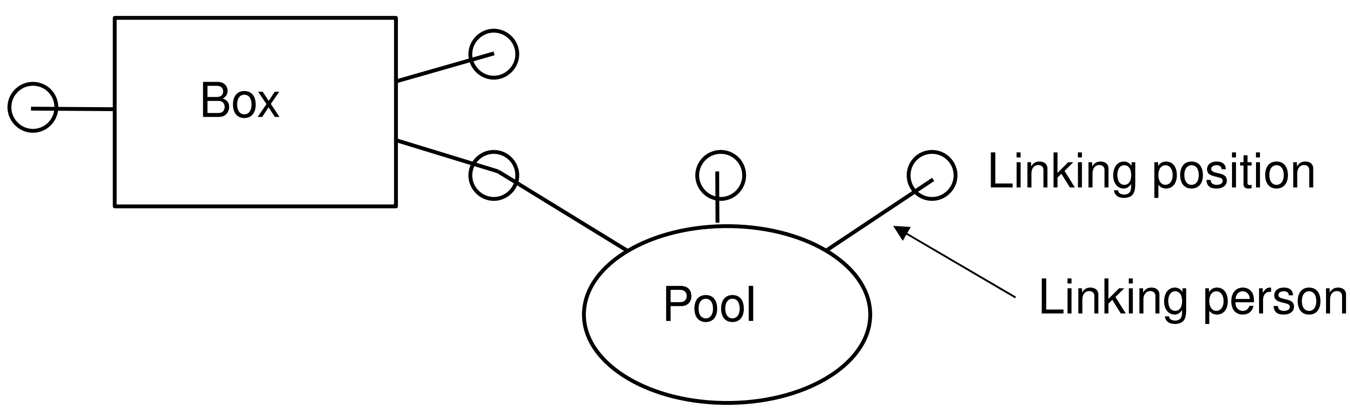
Finding U.F. of a matching

- Bellman-Ford shortest path algorithm
- Pressure edge: “length” –1
- “Shortest” path length to a position gives its pressure
- Remember, highest pressure = unpopularity factor



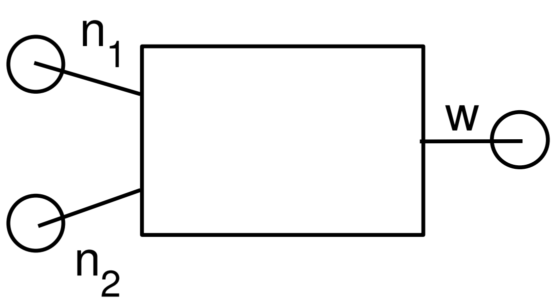
Finding matching of minimum U.F.

- Reduce 3SAT to the problem of finding the matching of minimum U.F. ⇒ it is NP-hard
- 3SAT solution ↔ matching of U.F. ≤ 2
- Gadgets confine pressures
- Analyze each gadget separately; a matching is acceptable iff it has pressure ≤ 2 in each



The reduction: Box

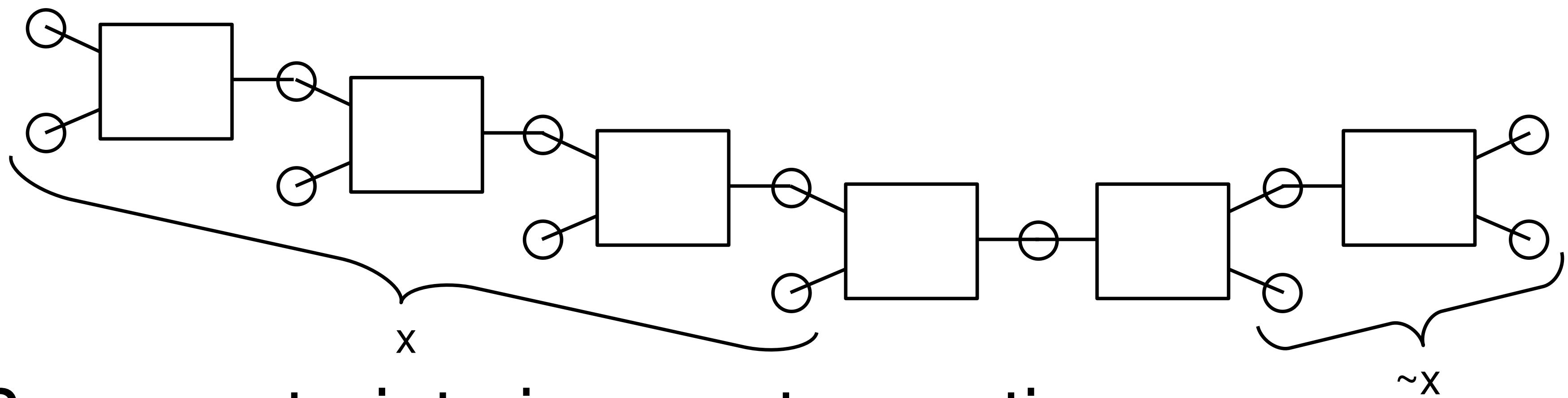
- To keep pressure ≤ 2, can assign either wide or narrow(s) (but not one of each) inside box



	x	y	z	u	l_w	l_{n1}	l_{n2}
i_1	1	2	3	4	–	–	–
i_2	1	2	3	4	–	–	–
i_3	1	2	3	4	–	–	–
w	2	3	5	4	1	–	–
n_1	–	–	–	2	–	1	–
n_2	–	–	–	2	–	–	1

The reduction: Variables

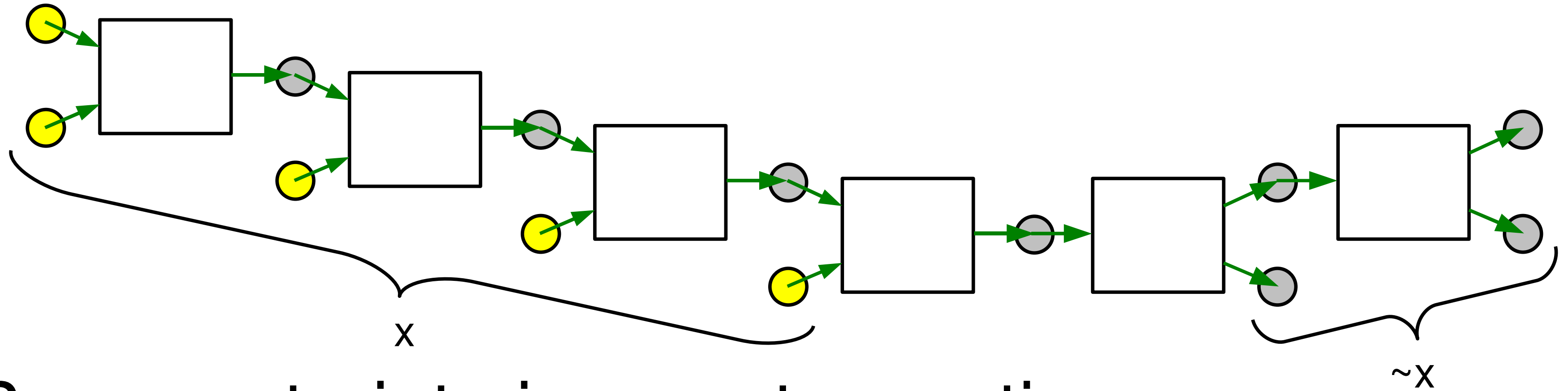
- Variable ↦ double-sided chain of boxes



- Box constraint gives us two options:
  - “True”: Assign “x” people inside boxes and “~x” people to linking positions
  - “False”: vice versa
- Leaves linking positions for satisfied variable references open

The reduction: Variables

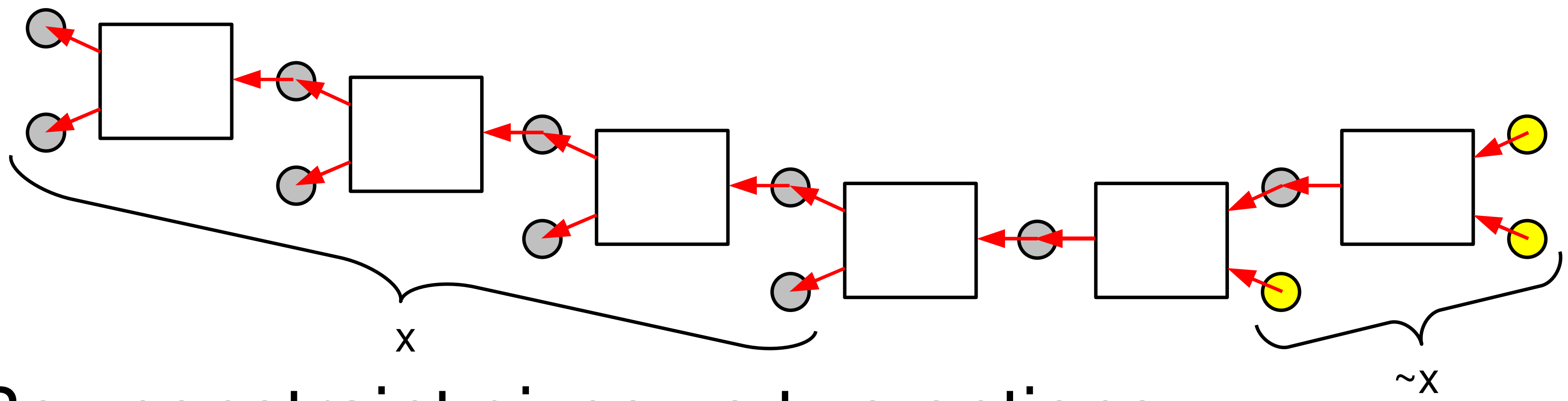
- Variable ↦ double-sided chain of boxes



- Box constraint gives us two options:
  - “True”: Assign “x” people inside boxes and “~x” people to linking positions
  - “False”: vice versa
- Leaves linking positions for satisfied variable references open

The reduction: Variables

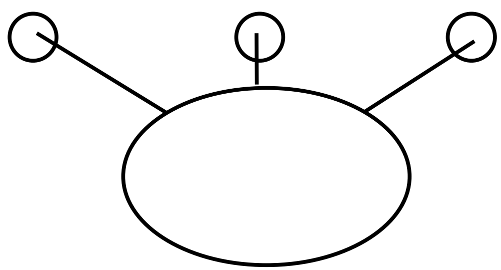
- Variable ↦ double-sided chain of boxes



- Box constraint gives us two options:
  - "True": Assign "x" people inside boxes and "~x" people to linking positions
  - "False": vice versa
- Leaves linking positions for satisfied variable references open

The reduction: Pool

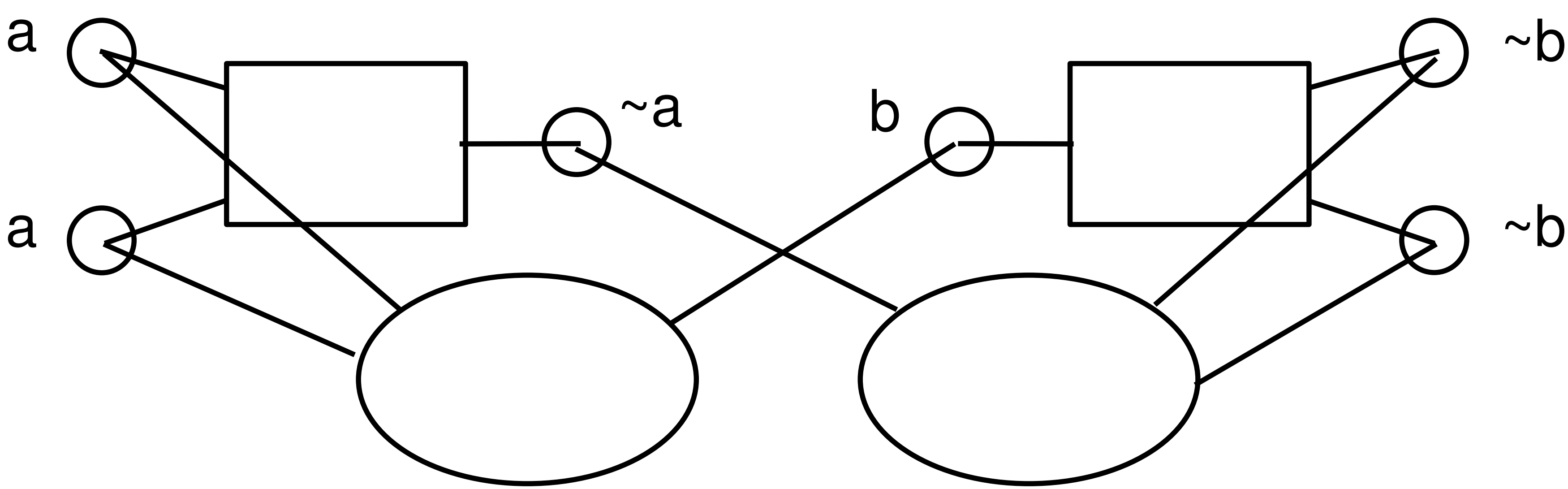
- To keep pressure ≤ 2, can assign at most two of the three linking people inside pool



	x	y	z	l_{f1}	l_{f2}	l_{f3}
f_1	2	3	4	1	–	–
f_2	2	3	4	–	1	–
f_3	2	3	4	–	–	1

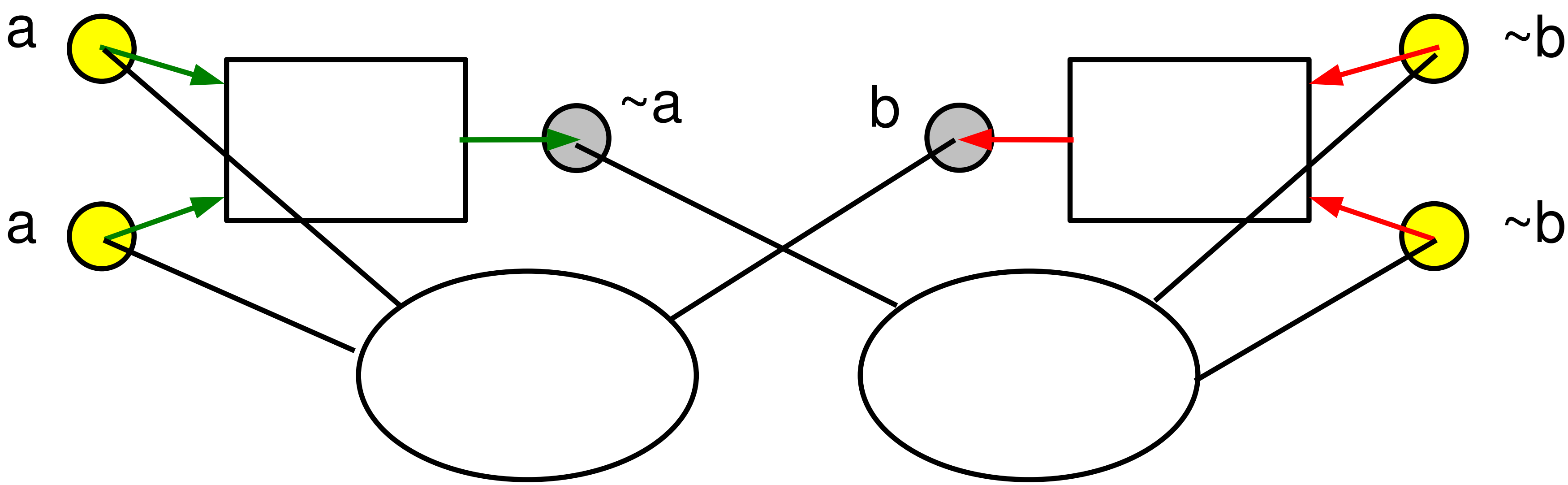
The reduction: Putting it together

- Clause ↦ pool
  - Identify linking positions with those of box chains according to variable references
- Example: a or b or a; (not b) or (not a) or (not b)



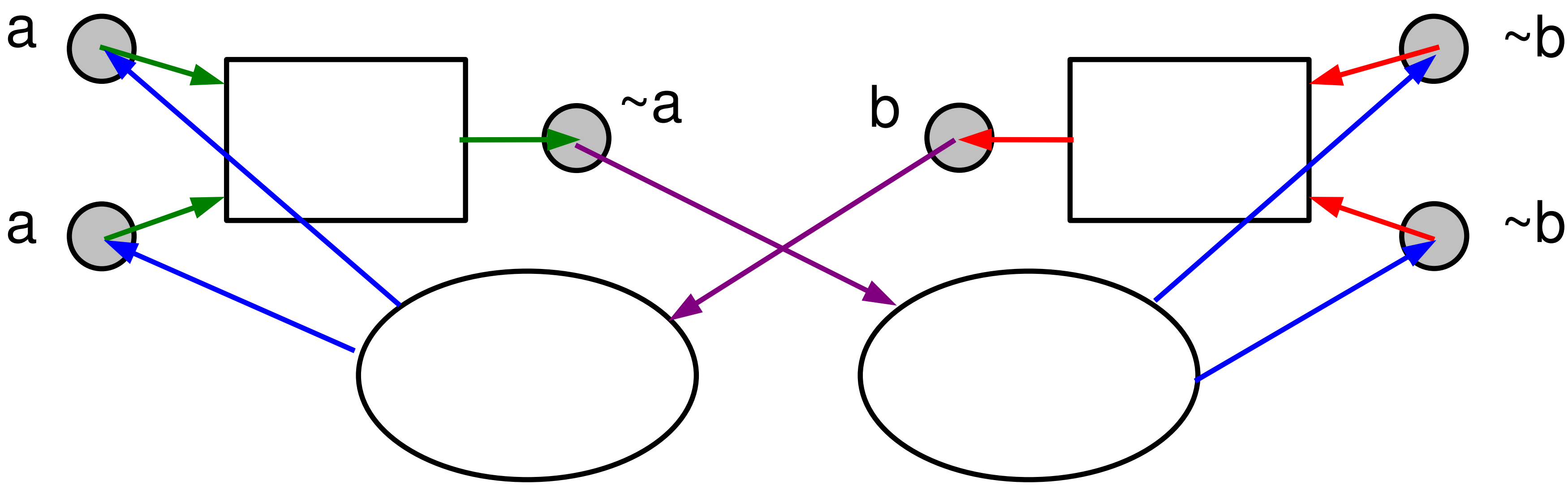
The reduction: Putting it together

- Clause ↦ pool
  - Identify linking positions with those of box chains according to variable references
- Example: a or b or a; (not b) or (not a) or (not b)
- Set a = true, b = false



The reduction: Putting it together

- Clause ↦ pool
  - Identify linking positions with those of box chains according to variable references
- Example: a or b or a; (not b) or (not a) or (not b)
- Set a = true, b = false; assign pool linking people



What to do about this?

- Can’t find matching of least unpopularity factor ⇒ the criterion is not useful for choosing matchings in practice
  – Open question: Is there an approximation algorithm?
- So try a different criterion!

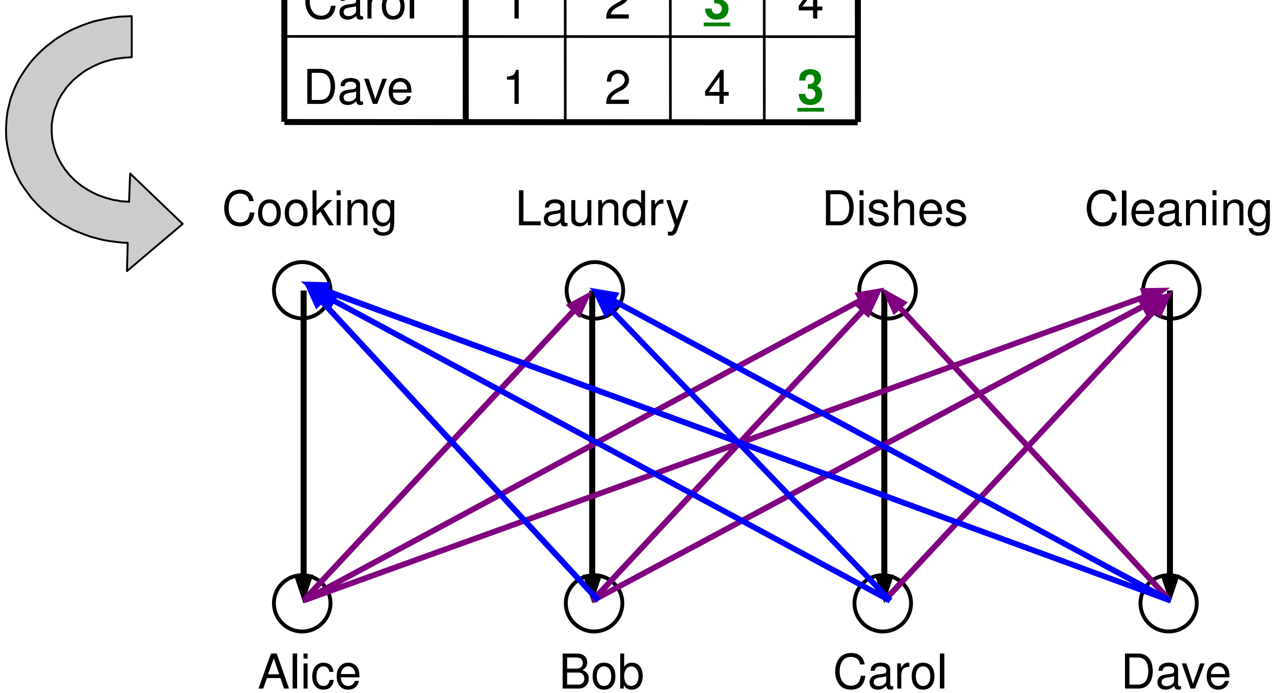
Unpopularity margin

- *N dominates M* by a *margin* of $u - v$ (instead of a factor of u/v); minimize the margin
- Differences:
  - Factor is based on worst pressure, a local property; margin is based (*roughly*) on the sum of all pressures, a global property
  - Originally liked factor criterion because it handles Pareto efficiency more nicely (positive/0 → infinite)
  - Margin criterion is better because one really bad, unfixable pressure doesn't deter it from optimizing the rest of the matching

Finding U.M. of a matching

- *Min-cost flow* models reassignment of unit-size people, resulting in –1 and +1 costs (votes)

M_2	Co	La	Di	Cl
Alice	**1**	2	3	4
Bob	1	**2**	3	4
Carol	1	2	**3**	4
Dave	1	2	4	**3**



All edge capacities are unit.

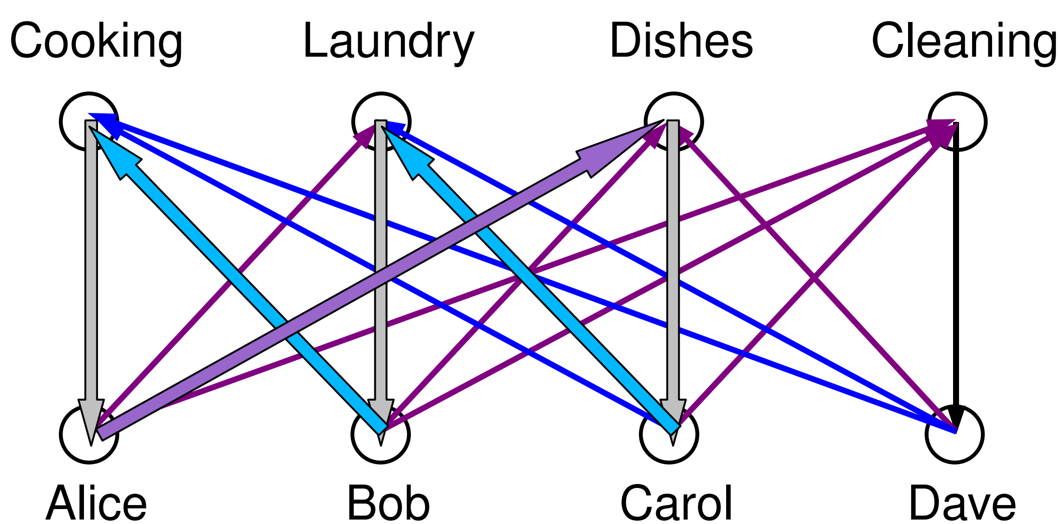
Colors give costs: **0**, **–1**, **+1**.

Finding U.M. of a matching

- Flow represents difference from M_2 to N_2
- Min. cost = –1 ⇒ unpopularity margin = 1

M_2	Co	La	Di	Cl
Alice	**1**	2	3	4
Bob	1	**2**	3	4
Carol	1	2	**3**	4
Dave	1	2	4	**3**

N_2	Co	La	Di	Cl
Alice	1	2	**3**	4
Bob	**1**	2	3	4
Carol	1	**2**	3	4
Dave	1	2	4	**3**



All edge capacities are unit.

Colors give costs: **0**, **–1**, **+1**.

Fat edges are used.

Finding matching of minimum U.M.

- Work in progress; neither algorithm nor NP-hardness proof yet
- Gadget-based reduction from 3SAT harder because we must account for *all* the pressures, not just the largest

Acknowledgments

- Samir Khuller, advisor
- Bobby Bhattacharjee
- Brian Dean
- Blair HS classmates, especially Nancy Zheng
- Dr. Torrence

Questions? Comments?