Preferences: modelling frameworks, reasoning tools, and multi-agent scenarios

Francesca Rossi
K. Brent Venable
University of Padova, Italy

Toby Walsh
NICTA and UNSW, Australia

Outline

- Part 1
  - Preferences
  - Soft constraints and CP nets
- Part 2
  - Uncertainty in preference reasoning
  - Multi-agent preference scenarios
  - Voting theory, fairness and manipulation
- Part 3
  - Computational aspects of preference aggregation and manipulation
  - Matching problems

Preferences vs. constraints

- Constraints are strict requirements
- Preferences as a way to provide more "tolerant" statements

Constraints

- Many real-life problems can be modelled via constraints
  - "I need at least two bedrooms"
  - "I don't want to spend more than 100K"
- Constraint = requirement = relation among objects (values for variables) of the problem
- Solution of a constraint problem = object choice (variable assignment) such that all constraints are satisfied
- Constraint programming offers
  - Natural modelling frameworks
  - Efficient solvers
  - Many application domains
    - Scheduling, timetabling, resource allocation, vehicle routing, ...

[Dechter 2000; Rossi, Van Beek, Walsh, 2006]
Constraints are not flexible

- Constraints are useful when we have a clear yes/no idea
  - A constraint can either be satisfied or violated
- Sometimes, we have a less precise model of the real-life problem
  - Ex.: “Both a skiing and a beach vacation are fine, but I prefer skiing”
- If all constraints, possibly
  - No solution, or
  - Too many solutions, and equally satisfiable

Preferences are everywhere

- Under-constrained problems ➔ many solutions ➔ we want to choose among solutions
- Over-constrained problems ➔ no solution ➔ we want to find an acceptable assignment
- Problems which are naturally modelled with preferences
- Constraints and preferences may occur together
  - Ex.: configuration, timetabling

Example: University timetabling

- Professor
  - I cannot teach on Wednesday afternoon.
  - I prefer not to teach early in the morning, nor on Friday afternoon.
- Lab C can fit only 120 students.
- Better to not leave 1-hour holes in the day schedule.

Several kinds of preferences

- Positive (degrees of acceptance)
  - “I like ice cream”
- Negative (degrees of rejection)
  - “I don’t like strawberries”
- Unconditional
  - “I prefer taking the bus”
- Conditional
  - “I prefer taking the bus if it’s raining”
- Multi-agent
  - “I like blue, my husband likes green, what color do we buy the car?”
Two main ways to model preferences

- **Quantitative**
  - Numbers or ordered set of objects
  - "My preference for ice cream is 0.8, and for cake is 0.6"
  - E.g., soft constraints

- **Qualitative**
  - Pairwise comparisons:
    - "Ice cream is better than cake"
  - E.g., CP-nets

Modelling preferences compactly

- **Preference ordering**: an ordering over the whole set of solutions (or candidates, or outcomes, ...)
- Solution space with a combinatorial structure \(\Rightarrow\) preferences over partial assignments, from which to generate the preference ordering over the solution space

Ultimate goal

- A formalism to model compactly problems with many kinds of preferences and to solve them efficiently

- **Uncertainty**
- **Multiple agents**

Formalisms to model preferences

- **Soft Constraints**
  - Quantitative formalism
  - Negative preferences
  - CP-nets (Conditional Preference Networks)
    - Qualitative formalism
    - Positive preferences

Two different ways to model compactly a preference ordering over a set of objects with a combinatorial structure
Soft Constraints:
the c-semiring framework

- Variables \( \{X_1, \ldots, X_n\} = X \)
- Domains \( D(X_j) = D \)
- Soft constraints
  - each constraint involves some of the variables
  - a preference is associated with each assignment of the variables
- Set of preferences \( \mathcal{A} \)
  - Totally or partially ordered (induced by +)
  - Combination operator (x)
  - Top and bottom element \( (1,0) \)
  - Formally defined by a c-semiring \( \langle \mathcal{A}, +, x, 0, 1 \rangle \)


Soft constraints

- Soft constraint: a pair \( c = (f, \text{con}) \) where:
  - Scope: \( \text{con} = \{X_{c_1}, \ldots, X_{c_k}\} \) subset of \( X \)
  - Preference function:
    - \( f: D(X_{c_1}) \times \ldots \times D(X_{c_k}) \rightarrow \mathcal{A} \)
    - \( \text{tuple } (v_1, \ldots, v_k) \rightarrow p \) preference
- Hard constraint: a soft constraint where for each tuple \( (v_1, \ldots, v_k) \)
  - \( f(v_1, \ldots, v_k) = 1 \) the tuple is allowed
  - \( f(v_1, \ldots, v_k) = 0 \) the tuple is forbidden

Complete assignments and their evaluation

- Complete assignment: one value for each variable
- Global evaluation: preference associated to a complete assignment
- How to obtain a global evaluation?
  - By combining (via x) the preferences of the partial assignments given by the constraints
Example: weighted constraints

- \( \langle A = \mathbb{N} \cup\{+\infty\}, + = \min, x = +, 0 = +\infty, 1 = 0 \rangle \)
- Values in \([0, +\infty]\)
  - Best value = 0
  - Worst value = +\infty
- Comparison with min
  - A better than B iff min(A, B) = A
- Composition with +
  - Goal is to minimize sum

Example: fuzzy constraints

- \( \langle A = [0, 1], + = \max, x = \min, 0 = 0, 1 = 1 \rangle \)
  - Preferences between 0 and 1
  - Higher values denote better preferences
  - 0 is the worst preference
  - 1 is the best preference
  - Combination is taking the smallest value
  - Optimization criterion = maximize the minimum preference

Pessimistic approach, useful in critical application (e.g., space and medical settings)

Fuzzy-SCSP example

<table>
<thead>
<tr>
<th>Lunch</th>
<th>Swim</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fish</td>
<td>White</td>
</tr>
<tr>
<td>12 pm, 1 pm</td>
<td>2 pm, 3 pm</td>
</tr>
</tbody>
</table>

Fuzzy semiring

\[ S = \langle A = [0, 1], + = \max, x = \min, 0 = 0, 1 = 1 \rangle \]

<table>
<thead>
<tr>
<th>Lunch</th>
<th>Wine</th>
<th>Swim</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 pm</td>
<td>meat</td>
<td>0.1</td>
</tr>
<tr>
<td>1 pm</td>
<td>white</td>
<td>0.7</td>
</tr>
</tbody>
</table>

Solution S

Lunch = 1 pm
Main course = meat
Wine = white
Swim = 2 pm
pref(S) = min(0.3, 0) = 0

<table>
<thead>
<tr>
<th>Lunch</th>
<th>Wine</th>
<th>Swim</th>
</tr>
</thead>
<tbody>
<tr>
<td>12 pm</td>
<td>fish</td>
<td>1</td>
</tr>
<tr>
<td>1 pm</td>
<td>white</td>
<td>2 pm</td>
</tr>
</tbody>
</table>

pref(S) = min(1, 1) = 1

Solution S'

Instances of semiring-based soft constraints

- Each instance is characterized by a c-semiring \( \langle A, +, x, 0, 1 \rangle \)
- Classical constraints: \( \langle \{0, 1\}, \text{logical or}, \text{logical and}, 0, 1 \rangle \)
  - Satisfy all constraints
- Fuzzy constraints: \( \langle [0, 1], \max, \min, 0, 1 \rangle \)
  - Maximize the minimum preference
- Lexicographic CSPs: \( \langle [0, 1], \lex \max, \min, 0, 1 \rangle \)
  - Order the preferences lexicographically and then maximize the minimum preference
- Weighted constraints (N): \( \langle \mathbb{N} \cup [0, +\infty], +, +, +\infty, 0 \rangle \)
  - Minimize the sum of the costs (naturals)
- Weighted constraints (R): \( \langle \mathbb{R} \cup [0, +\infty], +, +, +\infty, 0 \rangle \)
  - Minimize the sum of the costs (reals)
- Max CSP: weight = 1 if constraint is not satisfied and 0 if satisfied
  - Minimize the number of violated constraints
- Probabilistic constraints: \( \langle [0, 1], \max, x, 0, 1 \rangle \)
  - Maximize the joint probability of being a constraint of the real problem
- Valued CSPs: any totally ordered c-semiring
- Multi-criteria problems: Cartesian product of semirings
Multi-criteria problems

- One semiring for each criteria
- Given n c-semirings $S_i = <A_i, +, x_i, 0_i, 1_i>$, we can build the c-semiring $<A_1, ..., A_n, +, x, 0, 1>$
- $+$ and $x$ obtained by pointwise application of $+$ and $x_i$ on each semiring
- A tuple of values associated with each variable instantiation
- A partial order even if all the criteria are totally ordered
  - Pareto-like approach

Example

- The problem: choosing a route between two cities
- Each piece of highway has a preference and a cost
- We want to both minimize the sum of the costs and maximize the preference
- Semiring: by putting together one fuzzy semiring and one weighted semiring:
  - $<[0,1], \max, \min, 0, 1>$
  - $<\mathbb{N}, \min, +, +, 0>$
- Best solutions: routes such that there is no other route with a better semiring value
  - $<0.8, 10>$ is better than $<0.7, 15>$
- Two total orders, but the resulting order is partial:
  - $<0.6, 10>$ and $<0.4, 5>$ are not comparable

Solution ordering

- A soft CSP induces an ordering over the solutions, from the ordering of the semiring
- Totally ordered semiring $\Rightarrow$ total order over solutions (possibly with ties)
- Partially ordered semiring $\Rightarrow$ total or partial order over solutions (possibly with ties)
- Any ordering can be obtained!

Expressive power

- $A \leftrightarrow B$ iff from a problem $P$ in $A$ it is possible to build in polynomial time a problem $P'$ in $B$ s.t. the optimal solutions are the same (but not necessarily the solution ordering!)
  - $B$ is at least as expressive as $A$
- $A \rightarrow B$ iff from a problem $P$ in $A$ it is possible to build in polynomial time a problem $P'$ in $B$ s.t. $\text{opt}(P') \subseteq \text{opt}(P)$
Expressive power

![Diagram showing expressive power](image)

Interesting questions for soft CSPs

- Find an optimal solution
- Find the next solution in a linearization of the solution ordering
- Is s an optimal solution?
- Is s better than s’?

Finding an optimal solution

- Difficult in general
  - Branch and bound + constraint propagation
  - Local search
  - Bucket elimination
  - ...
- Easy for tree-shaped problems
  - Bucket elimination: directional arc-consistency + backtrack-free search
  - Also for problems with bounded treewidth

Finding the next solution

- Next where? In a linearization of the solution ordering
- Ties and incomparable sets should be linearized (any way is fine)
- Difficult for CSPs in general (so also for SCSPs)
- At least as difficult as finding an optimal solution
- Easy for tree-shaped CSPs and tree-shaped fuzzy CSPs
- Difficult for tree-shaped weighted CSPs

[Brafman, Rossi, Venable, Walsh, 2009]
Is \( s \) an optimal solution?

- **Difficult in general**: same complexity as finding an optimal solution
  - we have to find the optimal preference level
  - Easy for classical CSPs (optimal preference level is 1)

Is \( s \) better than \( s' \)?

- **Easy**: Linear in the number of constraints
  - Compute the two preference levels and compare them
  - Assumption: \( + \) and \( x \) easy to compute

Inference: Constraint propagation

- Constraint propagation (ex. arc-consistency):
  - Deletes an element \( a \) from the domain of a variable \( x \) if, according to a constraint between \( x \) and \( y \), it does not have any compatible element \( b \) in the domain of \( y \)
  - Iterate until stability
- Polynomial time
- Very useful at each node of the search tree to prune subtrees

Example

No matter what the other constraints are, \( X=b \) cannot participate in any solution. So we can delete it without changing the set of solutions.
Properties

- Equivalence: each step preserves the set of solutions
- Termination (with finite domains)
- Order-independence

Fundamental operations with soft constraints

- **Projection**: eliminate one or more variables from a constraint obtaining a new constraint preserving all the information on the remaining variables
  Formally: If \( c = \langle f, \text{con} \rangle \), then \( c|_{\text{con}} = \langle f', \text{con} \cap \text{con} \rangle \) such that \( f'(t') = \max(f(t)) \) over tuples of values \( t \) s.t. \( t|_{\text{con}} = t' \)

- **Combination**: combine two or more soft constraints obtaining a new soft constraint "synthesizing" all the information of the original ones
  Formally: If \( c_i = \langle f_i, \text{con}_i \rangle \), then \( c_1 \times c_2 = \langle f, \text{con}_1 \cup \text{con}_2 \rangle \) such that \( f(t) = \min(f_1(t|_{\text{con}_1}), f_2(t|_{\text{con}_2})) \)

**Projection: fuzzy example**

If \( c = \langle f, \text{con} \rangle \), then \( c|_{\text{mc}} = \langle f, \text{mc} \cap \text{con} \rangle \)

\[
f(t') = \max(f(t)) \text{ over tuples of values } t \text{ s.t. } t|_{\text{con}} = t'
\]

**Combination: fuzzy example**

If \( c_i = \langle f_i, \text{con}_i \rangle \), then \( c_1 \times c_2 = \langle f, \text{con}_1 \cup \text{con}_2 \rangle \)

\[
f(t) = \min(f_1(t|_{\text{con}_1}), f_2(t|_{\text{con}_2}))
\]
### Soft constraint propagation

- Deleting a value means passing from 1 to 0 in the semiring \(<{0,1}, \text{or, and}, 0,1>\).
- In general, constraint propagation can change preferences to lower values in the ordering.
- **Soft arc-consistency**: given \(c_x, c_{xy}, \text{and} \ c_y\), compute \(c_y := (c_x \times c_{xy} \times c_y)_x\).
- Iterate until stability.

### Properties

- If \(x\) idempotent (ex.: fuzzy, classical):
  - Equivalence
  - Termination
  - Order-independence
- If \(x\) not idempotent (ex.: weighted CSPs, prob.), we could count more than once the same constraint \(\Rightarrow\) we need to compensate by subtracting appropriate quantities somewhere else \(\Rightarrow\) we need an additional property (fairness=presence of -)
  - Equivalence
  - Termination
  - Not order-independence
  
  [Schiex, CP 2000]

### Bucket elimination

- Generalization of adaptive consistency to soft constraints.
- Choose a linear ordering of the variables: \(x_1, \ldots, x_n\).
- From \(x_n\) to \(x_1\), take \(x_i\):
  - Combine all constraints involving \(x_i\).
  - Project this new constraint over frontier(\(x_i\)).
  - Add the constraint to the SCSP.
- At the end, the highest preference of \(x_1\) is the preference of the optimal solutions.
- An optimal solution can be found by instantiating \(x_1, \ldots, x_n\) taking for each variable an optimal value from its domain which is compatible with the values chosen for the previous variables.
  
  [Dechter, AI Journal 1999]
Complexity

- As many steps as the number of variables ($n$)
- At each step, time exponential in the size of the frontier $Y$ of the current variable plus one (and space exponential in size of $Y$)
- $n$ steps to find an optimal solution
- Time: $O(n \times \exp(|Y|) +n)$
- But space is the main problem with this method

Qualitative and conditional preferences

- Soft constraints model quantitatively unconditional preferences
- Many problems need statements like
  - "I like white wine if there is fish" (conditional)
  - "I like white wine better than red wine" (qualitative)
- Quantitative $\Rightarrow$ a level of preference for each assignment of the variables in a soft constraint $\Rightarrow$ possibly difficult to elicitate preferences from user

Preference statements in CP nets

- Conditional preference statements
  - "If it is fish, I prefer white wine to red wine"
  - syntax:
    - fish: white wine $>$ red wine
- Ceteris paribus interpretation
  - all else being equal
  - (fish, white wine, ice cream) $>$ (preferred to)
    - (fish, red wine, ice cream)
  - (fish, white wine, ice cream) $<$
    - (fish, red wine, fruit)

CP nets

- Variables $\{X_1, \ldots, X_n\}$ with domains
- For each variable, a total order over its values
- Independent variable:
  - $X=v_1 > X=v_2 > \ldots > X=v_k$
- Conditioned variable: a total order for each combination of values of some other variables (conditional preference table)
  - $Y=a, Z=b; X=v_1 > X=v_2 > \ldots > X=v_k$
  - $X$ depends on $Y$ and $Z$ (parents of $X$)
- Graphically: directed graph over $X_1, \ldots, X_n$
  - Possibly cyclic
CP nets: an example

<table>
<thead>
<tr>
<th>Independent feature</th>
<th>Main course</th>
<th>Wine</th>
</tr>
</thead>
<tbody>
<tr>
<td>fish &gt; meat</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Conditional Preference Table

<table>
<thead>
<tr>
<th>Main course</th>
<th>Wine</th>
</tr>
</thead>
<tbody>
<tr>
<td>fish</td>
<td>white &gt; red</td>
</tr>
<tr>
<td>meat</td>
<td>red &gt; white</td>
</tr>
</tbody>
</table>

Dependent feature

Fruit

peaches > strawberries

CP-net semantics

- **Worsening flip**: changing the value of an attribute in a way that is less preferred in some statement. Example:
  - (fish, white wine, peaches)
  - worsening flip
  - (fish, red wine, peaches)

- An outcome $O_1$ is preferred to $O_2$ iff there is a sequence of worsening flips from $O_1$ to $O_2$.
- **Optimal outcome**: if no other outcome is preferred

Preorder over solutions

- A CP net induces an ordering over the solutions (directly)
- In general, a preorder
- Some solutions can be in a cycle: for each of them, there is another one which is better
- **Acyclic CP net**: one optimal solution
- Not all orderings can be obtained with CP nets
  - Outcomes which are one flip apart must be ordered

Solution ordering

A CP net induces a preorder over the solutions.

- Optimal solution: fish, white, peaches
  - fish, red, peaches
  - fish, white, berries
  - fish, red, berries
  - meat, red, peaches
  - meat, white, peaches
  - meat, red, berries
  - meat, white, berries
Interesting questions in CP nets

- Find an optimal outcome
  - In general, difficult (as solving a CSP)
  - Easy for acyclic networks
    - always have exactly one optimal solution
    - sweep forward in linear time
- Find the next solution in a linearization of the solution ordering
  - Easy for acyclic CP-nets
- Does O1 dominate O2?
  - Difficult even for acyclic CP nets
- Is O optimal?
  - Easy: test O against a CSP

Expressive power

If interested in the optimal solutions:

- Classical
  - Semiring-based
    - Valued
    - weighted_N
    - weighted_R
    - Prob
- Fuzzy
  - Lexicographic
  - weighted_R

If interested in maintaining the solution ordering:

- CP nets
  - Classical
  - Semiring-based
    - Valued
    - weighted_N
    - weighted_R
    - Prob
- Fuzzy
  - Lexicographic
  - weighted_R
CP nets vs. Soft Constraints
(solution ordering)

- There are CP nets whose ordering cannot be modelled (in poly time) by a soft CSP
  - Otherwise dominance testing would be easy in CP-nets

- There are soft CSPs whose orderings cannot be modelled by a CP net
  - Not all orderings can be represented by CP nets

---

How to find optimal solutions in CP nets

- Acyclic CP-nets: sweep forward algorithm
  - Follow the dependency graph
  - For each variable, assign the most preferred value in the context of the parents’ assignment

---

Sweep forward algorithm

1. F = peaches
2. M = fish
3. Since M=fish, W=white

---

Cyclic CP nets

- Given a (cyclic) CP net, we can generate in polynomial time a set of constraints P such that the solutions of P coincides with the set of optimal solutions of the CP net
  - For each Y=a, Z=b: X=v₁ > X=v₂ > ... > X=vₖ, we build the constraint Y=a, Z=b \rightarrow X=v₁
Optimal solutions in cyclic CP nets

**Constraints:**
- F = peaches
- M = fish
- W = white
- W = red
- W = meat
- M = fish
- M = meat
- peaches > strawberries

**Main course**
- Fish: white > red
- Meat: red > white
- White: fish > meat
- Red: meat > fish

Optimal solutions:
- Fish, white, peaches
- Meat, red, peaches
- Fish, white, berries
- Meat, red, berries
- Fish, red, berries
- Meat, white, berries

Approximating CP nets via Soft Constraints

- We can approximate the ordering of a CP net via a soft constraint problem
  - Weighted or fuzzy soft constraints
  - For ordered outcomes, same ordering
  - For incomparable outcomes, tie or order \( \to \) more ordered
  - Easy dominance test

CP statements \( \rightarrow \) Soft constraints

Soft constraint solver

optimal solutions/approximate dominance test

[Domshlak, Rossi, Venable, Walsh, IJCAI 2003]

Constrained CP-net

A **Constrained CP-net** on variables \( X = \{X_1, \ldots, X_n\} \) is a pair \( <N, C> \) where:
- N is a CP-net on variables X
- C is a set of Hard or Soft Constraints on X

Constrained CP-net semantics:
- \( O_1 \geq O_2 \) iff
  - there is a chain of worsening flips from \( O_1 \) to \( O_2 \).
  - each outcome in the chain is optimal for \( C \) (feasible for hard constraints)

- \( O \) optimal if feasible and undominated in the CP net (not necessarily optimal in the CP net)

Softly Constrained CP net : example

**Optimal solutions**
- Fish, white, peaches
- Fish, white, berries
- Meat, red, peaches
- Meat, white, berries

**Soft Constraint**
- Wine
- White \( \rightarrow \) 0.2
- Red \( \rightarrow \) 1
How to obtain an optimal outcome of a constrained CP net \(<N,C>\):

- From N to optimality constraints OC
- If \(\text{Sol}(OC \cup C)\) is not empty, then they are (some of the) optimal outcomes \(\Rightarrow\) take one of them
  \(\Rightarrow\) only hard constraint solving
- Otherwise, dominance testing between feasible outcomes (more costly)

(Conditional + qualitative + quantitative) preferences + constraints

<table>
<thead>
<tr>
<th>Preferences and Uncertainty</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sources of uncertainty</td>
</tr>
<tr>
<td>----------------------------</td>
</tr>
<tr>
<td>Preferences rather than hard constraints</td>
</tr>
<tr>
<td>Uncontrollable variables</td>
</tr>
<tr>
<td>Events that will be decided by Nature or some other agent</td>
</tr>
<tr>
<td>Missing preferences</td>
</tr>
<tr>
<td>Giving all the preferences may be too much work for a user</td>
</tr>
<tr>
<td>The user might prefer not reveal some preferences</td>
</tr>
<tr>
<td>Imprecise preferences</td>
</tr>
<tr>
<td>Ranges</td>
</tr>
</tbody>
</table>
Uncontrollable variables

- Example: clouds' starting and ending times in a satellite scheduling system
- Possibilistic or probabilistic information over their domains
- Aim: finding solutions "robust" w.r.t. the uncontrollable part
- Also in temporal constraint/preference reasoning
  - Several levels of controllability (strong, weak, dynamic)
- Dynamic programming, bucket elimination
- Solvers that generalize soft constraint solvers
  - Eliminate the uncontrollable part by transforming it into new soft constraints on the frontier
  - Transformation assures certain lower bounds on the robustness of optimal solutions

Uncontrollable variables in soft constraint problems

Open Constraint Optimization problems (1)

- OCOP
  - an unbounded sequence of COPs: \{COP(0), COP(1), \ldots\}
  - COP(i) = \{X, D(i), C(i)\}
    - X set of variables
    - D(i): variable domains of instance i
    - C(i): preference functions of instance i
- COP(i) < COP(j)
  - If the domains in D(j) are supersets of those in D(i)
Open Constraint Optimization Problems (2)

- **Monotonicity assumption:** better values are provided first.
- Otherwise, the best may be revealed last, requiring all values and preferences to be queried by any algorithm.

### OCOP Algorithm schema for fuzzy preferences

1. **Threshold** $t=0$
2. Find a solution with preference $t$
3. If a solution with preference $t$ is found increase $t$
4. if all domains are exhausted or $t$ is the optimal preference of a sub-problem return current solution
5. else
   1. obtain next values with preference lower than $t$
   2. obtain next values with preference higher than $t$ only for critical variables
   3. go back to 2

### ISCPS

<table>
<thead>
<tr>
<th>(plane, ship)</th>
<th>(plain, Mexico)</th>
<th>(ship, Mexico)</th>
<th>(ship, Caribbean)</th>
</tr>
</thead>
<tbody>
<tr>
<td>plane ... 0.8</td>
<td>plain ... 0.7</td>
<td>ship ... 0.9</td>
<td>ship ... 0.8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(room, suite, bungalow)</th>
<th>(Caribbean, Mexico)</th>
</tr>
</thead>
<tbody>
<tr>
<td>room ... 1</td>
<td>Caribbean ... 0.7</td>
</tr>
<tr>
<td>suite ... 1</td>
<td>Mexico ... 0.9</td>
</tr>
<tr>
<td>bungalow ... 1</td>
<td></td>
</tr>
</tbody>
</table>

**Fuzzy** $\langle 0.1, \text{max}, 0.1 \rangle$

A travel agency is planning Alice and Bob's vacation knowing only some of their preferences about transport, destination, and accommodation.

### Completions of an ISCSP

- **Completion** of an ISCSP $P$: SCSP obtained from $P$ by adding the missing preferences.
- **0-completion** of $P$: completion of $P$ where each $?$ is replaced by $0$
  - Worst possible scenario
- **1-completion** of $P$: completion of $P$ where each $?$ is replaced by $1$
  - Best possible scenario
Possibly and necessarily optimal solutions

- **Possibly optimal solutions** POS(P): assignments to all the variables that are undominated in some completion of P
  - i.e., in some completion of P there is no sol. s’ s.t. pref(s’)>pref(s)
  - Solutions that are realizable in some scenario
- **Necessarily optimal solutions** NOS(P): assignments s to all the variables that are undominated in all the completions of P
  - i.e., in all the completions of P there is no sol. s’ s.t. pref(s’)>pref(s)
  - Robust w.r.t. the missing part
- **Losers** L(P): solutions that are neither possibly optimal nor necessarily optimal
  - No chance of being optimal

- NOS(P) ⊆ POS(P)
- POS(P) ∩ L(P) = ∅, POS(P) ∪ L(P) = Sol(P)

What to look for?

- Aim: find a necessarily optimal solution of the given problem, or of a (partial) completion of it
- A possible method:
  - If there are necessarily optimal solutions, find one
  - Otherwise, elicit some of the missing preferences and start again
    - Elicit where it looks more promising (no losers)

How to characterize necessarily optimal solutions

- We can determine the necessarily optimal solutions, if they exist, without eliciting preferences

  How? By comparing the 0-completion and the 1-completion
  - pref0: preference of an optimal solution of P0
  - pref1: preference of an optimal solution of P1
  - Since 0s1 and x is monotone, then pref0 ≤ pref1
  - If pref1= pref0, NOS(P) = {optimal solutions of P0}
  - Otherwise:
    - NOS(P) = ∅
    - POS(P) (solutions with pref. between pref0 and pref1)
    - L(P) = {solutions with pref. below pref0}
  - Either we find the pref. of the necessary optimal solutions, or we have a range where for the pref. of the possibly optimal solutions

Possibly and necessarily optimal solutions

<table>
<thead>
<tr>
<th>Fuzzy</th>
<th>[0,1]_max,min,0,1</th>
</tr>
</thead>
</table>

The optimal solution of P0 is (ship, Caribbean, room) with preference pref= 0.3

The optimal solutions of P1 including (ship, Caribbean, suite) have preference pref=0.7

NOS(P) = ∅
POS(P) = {all assignments with pref. in [0.3, 0.7]}
Solvers: a general schema

- Input: an ISCSP P (over a totally ordered c-semiring)
- Output:
  - Solution \( s \in \text{NOS}(P) \) if it exists, \( \text{pref}(s), P \)
  - Otherwise, \( s \in \text{NOS}(Q) \), \( \text{pref}(s), Q \)
  - Q is obtained from P after eliciting some preferences

Main idea:
- First solve the 0-completion
- Then solve the 1-completion interleaving branch and bound search with preference elicitation
- Elicit the most promising "object" (solution preference or partial solution preference)

Three elicitation parameters

- When elicitation happens
  - At the end of every BB run (tree level)
  - At the end of every complete branch
  - At every node

- What we ask the user to provide
  - All the missing prefs
  - The worst pref

- Who decides what values to give to the next variable
  - The system
  - In decreasing preference order in P1 (dp)
  - In decreasing preference ordering in P0 (dip)
  - Fixed values considered first
  - The user
  - Just looking at the domain preferences (lazy user: lu)
  - Looking also at the preferences in the constraints between the next var and the previous vars (smart user: su)

Incomplete fuzzy constraints

- Unstable preference = interval

Interval preferences

- Unstable preference = interval

The cost of this component will be between 10 and 20.

I like that at least x

[Gelain, Pini, Rossi, Venable, Wilson Pref'08]
Interval-valued constraints

- **Interval-valued constraint**: a pair \( c = \langle f, \text{con} \rangle \) where:
  - Scope: \( \text{con} = \{ X_1, \ldots, X_n \} \) subset of \( X \)
  - Preference function: \( f : D(X_1) \times \cdots \times D(X_n) \rightarrow A \times A \)

\[ \text{tuple } (v_1, \ldots, v_k) \rightarrow (l, u), l \leq u, \]
- \( l \): lower bound of the preference interval
- \( u \): upper bound of the preference interval

**Solutions of IVCSPs**

- **Complete assignment** \( s \): one value for each variable
- **Global evaluation of** \( s \): preference interval \( [L(s), U(s)] \)
  - \( L(S) \): combination (via \( x \)) of the lower bounds of the preferences of the partial assignments given by the constraints
  - \( U(S) \): combination (via \( x \)) of the upper bounds of the preferences of the partial assignments given by the constraints

**Example: solution of a Fuzzy IVCSP**

Fuzzy c-semiring: \( S_{FCSP} = <[0,1], \max, \min, 0, 1> \)

\( (a,a,a) \rightarrow [0.6, 0.9] \)

\( \min(1, 0.8, 0.6, 0.8, 0.9) \)

**Scenarios**

- **Scenario of an IVCSP** \( P \)
  - \( \text{SCSP} \) \( Q \) obtained replacing each imprecise preference with a value in its interval
  - worst scenario: \( l \) everywhere
  - best scenario: \( u \) everywhere
- \( S(P) \): set of all scenarios of IVCSP \( P \)
Example: Scenarios

Optimality notions for IVCSP

- **Lower/upper optimal solutions** (L/U O)
  - maximal lb/ub
  - optimal in the worst/best scenario
- **Interval optimal solutions** (IO)
  - maximal lb or ub
- **Weakly interval dominant solutions** (WLO)
  - maximal lb and ub
- **Interval dominant solutions** (ID)
  - lb greater or equal than the ub of all others

Summarizing: Interval optimality (1)
Multi-agent preferences

- Several agents (people, software agents, etc.) expressing their preferences over a set of scenarios (solutions, outcomes, etc.)
- We need to aggregate their preferences to obtain a result which satisfies all
- Result can be:
  - A preference ordering over the scenarios
  - A set of scenarios (optimal, winners, etc.)
- Preferences (one agent, or result) are expressed via partial orders

Why partial orders?

- When combining the preferences of different agents, incomparability as a means to resolve conflicts
- For a single agent:
  - some objects may naturally be incomparable
  - several possibly conflicting criteria
  - incomparability to model uncertainty
- Many AI formalisms to represent preferences generate partial orders or preorders
- POs for describing both the preferences of an agent and the results of preference aggregation
Preference aggregation

- We need to aggregate the preferences to
  - Test optimality
  - Find an optimal outcome
  - Order two outcomes
- Our proposal: ask each agent dominance queries and then collect the votes as in an election → voting theory

Brief overview of classical voting theory

Terminology

- **Agent**
  - Usually assume odd number of agents to reduce ties
- **Vote**
  - Total order over outcomes (or candidates)
  - Extensions include indifference, incomparability, incompleteness
- **Profile**
  - Vote for each agent

Voting rule

- **Social choice**: mapping of a profile onto a winner(s)
- **Social welfare**: mapping of a profile onto a total ordering over the candidates

Voting rules: plurality

- Otherwise known as “majority”
  - Candidate who is the most preferred for the majority of agents wins
- With just 2 candidates, this is a very good rule to use
  - (See May’s theorem)
Criticisms of plurality

- Ignores preferences other than favourite
- Similar candidates can "split" the vote
- Encourages voters to vote tactically
  - "My candidate cannot win so I'll put my second favorite first"

Voting rules: plurality with runoff

- Two rounds
  - Eliminate from the profiles all but the 2 candidates with most votes
  - Use plurality to choose the winner among the remaining 2 candidates
- Drawback: Requires voters to list all preferences or to vote twice

Plurality with run off is not monotonic

- Moving a candidate up your ballot may not help them

<table>
<thead>
<tr>
<th>Example</th>
<th>39 A&gt;B&gt;C</th>
<th>49 A&gt;B&gt;C</th>
<th>25 B&gt;C&gt;A</th>
<th>26 C&gt;A&gt;B</th>
<th>46 votes: C&gt;A&gt;B</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A wins 65:35</td>
<td>B is eliminated</td>
<td>C wins 51:49 !</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Plurality with runoff may incentivize abstention

- Two voters disliking C don’t vote
  - 23 votes: A>B>C
  - 24 votes: B>C>A
  - 46 votes: C>A>B
  - C wins

- Consider again
  - 25 votes: A>B>C
  - 24 votes: B>C>A
  - 46 votes: C>A>B
  - C wins

- Consider
  - 25 votes: A>B>C
  - 24 votes: B>C>A
  - 46 votes: C>A>B
  - 1st round: B knocked out
  - 2nd round: C>A by 70:25
  - C wins

- Different result
  - 1st round: A knocked out
  - 2nd round: B>C by 47:46
  - B wins

If 10 B supporters would have put A first...
Voting rules: single transferable vote

- **STV**
  - If one candidate has >50% vote then he is elected
  - Otherwise the candidate with least votes is eliminated
  - His votes transferred (2nd placed candidate becomes 1st, etc.)
  - Identical to plurality with runoff for 3 candidates

- Example:
  - 39 votes: A>B>C>D
  - 20 votes: B>A>C>D
  - 20 votes: B>C>A>D
  - 11 votes: C>B>A>D
  - 10 votes: D>A>B>C
  - Result: B wins!

Voting rules: Borda

- Given m candidates
  - ith ranked candidate score m-i
  - Candidate with greatest sum of scores wins

- Example
  - 42 votes: A>B>C>D
  - 26 votes: B>C>D>A
  - 15 votes: C>D>B>A
  - 17 votes: D>C>B>A
  - B wins

Jean Charles de Borda, 1733-1799

Voting rules: positional rules

- Given vector of weights, <s_1,...,s_m>
  - Candidate scores s_i for each vote in ith position
  - Candidate with greatest total score wins

- Generalizes many rules
  - Borda is <m-1,m-2,...,0>
  - Plurality is <1,0,...,0>

More voting rules

- **Approval**
  - Each voter approves between 1 and m-1 candidates
  - Candidate with most votes of approval wins

- **Cup (aka knockout)**
  - Tree of pairwise majority elections

- **Copeland**
  - The winner is the candidate that wins the most pairwise competitions
…Voting rules

- So many voting rules to choose from..
- Which is best?
  - Social choice theory looks at the (desirable and undesirable) properties they possess (e.g. monotonicity)
  - Bottom line: with more than 2 candidates, there is no best voting rule

Axiomatic approach

- Define desired properties
  - E.g. monotonicity: improving votes for a candidate can only help them win
- Prove whether voting rule has this property
  - In some cases, as we shall see, we'll be able to prove impossibility results (no voting rule has this combination of desirable properties)

Anonymity and Neutrality

- Some desirable properties of voting rule
  - Anonymous: names of voters irrelevant
  - Neutral: name of candidates irrelevant

Monotonicity

- Another desirable property of a voting rule
  - Monotonic: if a particular candidate wins, and a voter improves his vote in favor of this candidate, then he still win
- We have already seen that plurality with run-off is not monotonic
May’s theorem

- Thm: With 2 candidates, a voting rule is anonymous, neutral and monotonic if it is the plurality rule
  - Since these properties are uncontroversial, this about decides what to do with 2 candidates!

Condorcet’s paradox

- Collective preference may be cyclic
  - Even when individual preferences are not
  - Consider 3 votes
    - A>B>C
    - B>C>A
    - C>A>B
  - Condorcet cycle
    - Majority prefer A to B, and prefer B to C, and prefer C to A!

Condorcet principle

- Turn this on its head
  - Condorcet winner
    - Candidate that beats every other in pairwise elections
    - In general, Condorcet winner may not exist
    - When he exists, he must be unique
  - Condorcet consistent
    - Voting rule that elects the Condorcet winner when he exists (e.g. Copeland rule)

Condorcet principle

- Plurality rule is not Condorcet consistent
  - 35 votes: A>B>C
  - 34 votes: C>B>A
  - 31 votes: B>C>A
  - B is the Condorcet winner, but plurality elects A
Other desirable properties

- **Free**
  - Every result is possible

- **Unanimous**
  - If everyone votes for the same candidate, he wins

- **Independent to irrelevant alternatives**
  - Result between A and B only depends on the agents’ preferences between A and B (and not A and C and C and B...)

- **Non-dictatorial**
  - Absence of a dictator
  - Dictator: voter whose vote always coincides with the result

---

Arrow’s theorem

- Thm: If there are at least two voters and three or more candidates, then it is impossible for any voting rule to be at the same time:
  - Free
  - Unanimous
  - Independent to irrelevant alternatives
  - Monotonic
  - Non-dictatorial

---

Arrow’s theorem: stronger version

- Weaker conditions
  - Pareto property
    - If everyone prefers A to B then A is preferred to B in the result
  - If free & monotonic & IIA then Pareto
  - If free & Pareto & IIA then not necessarily monotonic

- Thm: If there are at least two voters and three or more candidates, then it is impossible for any voting rule to be:
  - Pareto
  - Independent to irrelevant alternatives
  - Non-dictatorial

---

Arrow’s theorem: ways around

- With two candidates, majority rule is:
  - Pareto
  - Independent to irrelevant alternatives
  - Non-dictatorial

- So, one way “around” Arrow’s theorem is to restrict to two candidates
Arrow’s theorem: ways around

- How do we get “around” this impossibility
  - Limit domain
    - Only two candidates
  - Limit votes
    - Single peaked votes
  - Limit properties
    - Drop IIA
  - ...
  - What happens if we allow the voters to express incomparability between candidates?

Dictators with partial orders

- Strong dictator: a voter such that his ordering is the result
- Dictator: if he says A better than B, then the result is A better than B
  - But if he says that A and B are incomparable/indifferent, then they can be ordered in the result
- Weak dictator: if he says A better than B, then the result cannot be B better than A
  - But it can be A incomparable/indifferent to B
- At most one strong dictator or dictator, possibly many weak dictators
  - Strong dictator $\Rightarrow$ dictator $\Rightarrow$ weak dictator

Arrow’s theorem with partial orders

- It is possible for a rule to be free, monotonic, independent, and not to have any strong dictator
- It is possible for a rule to be free, monotonic, independent, and not to have any dictator
- Given some restrictions on the partial orders it is impossible for a rule to be free, monotonic, independent, and not to have any strong dictator

Manipulation

- Constructive
  - Can we change result so a given candidate wins
- Destructive
  - Can we change result so a given candidate does not win

[Pini, Rossi, Venable, Walsh, JLC 2009]
Manipulation

- Means to manipulate
  - Our vote
  - A coalition of voters
  - Other voters
    - Bribery
  - Chair person (control)
    - Agenda
    - Adding/deleting candidates
    - Adding/deleting votes
    - ...

An example

- Consider the following vote
  - 49%: A>B>C
  - 20%: B>C>A
  - 20%: B>A>C
  - 11%: C>B>A → B>C>A → Now B wins!

- A wins a plurality vote
  - B is the Condorcet winner (pairwise winner)
  - C’s supporters can “manipulate” vote and get a “better” result by voting for B

The Gibbard-Satterthwaite theorem

- All “reasonable” voting rules are manipulable under weak assumptions
  - One of social choice’s most fundamental results
  - Only limited ways to escape GS
    - Restrict how people can vote
    - Ensure it is (computationally) difficult to manipulate result
    - ...

The Gibbard-Satterthwaite Theorem

- Assumptions
  - 2 or more agents
  - 3 or more candidates
  - Voting rule is onto
    - Every candidate is able to win
  - Voting rule is strategy-proof
    - Voting insincerely does not help
    - More precisely, an agent does not improve the result by mis-reporting their preferences
Gibbard-Satterthwaite

- **Assumptions**
  - 2 or more agents
  - 3 or more candidates
  - Voting rule is onto
  - Voting rule is strategy-proof

- **Conclusion**
  - Voting rule is dictatorial
    - One agent dictates the result

Circumventing Gibbard Satterthwaite

- **Limit candidates**
  - With 2 candidates, plurality is strategy-proof and lacks a dictator
- **Restrict vote**
  - For example, only permit single peaked votes
  - Then "median" rule is
    - Onto
    - Strategy-proof
    - Non-dictatorial

Control: Manipulating the agenda

- Consider the cup rule
- Suppose we have a Condorcet cycle:
  - Agent1: A>B>C
  - Agent2: B>C>A
  - Agent3: C>A>B
- By choosing agenda, Chair can make anyone win
  - A win: play B against C, winner plays A
  - B win: play C against A, winner plays B
  - C win: play A against B, winner plays C

Strategy proofness with POs

- Agents should not be able to make an outcome win by lowering its position in their preference ordering
- For every agent i, for every two profiles p and p’, which differ on pi only, for every a in f(p)-f(p’), for every b in f(p’),
  - a \not< pi b \Rightarrow a \not< pi b or a < pi b
  - a < pi b \Rightarrow a < pi b
- There is at least an element b in f(p’) such that
  - a > pi b \Rightarrow a \not< pi b or a < pi b
  - a \not< pi b \Rightarrow a < pi b
- One agent can remove an element (a) from the set of winners only by worsening it with respect to at least one of the new winners (b)

[Pini, Rossi, Venable, Walsh, JLC 2009]
Gibbard-Satterthwaite thm. with POs

- Social choice function from POs to PO
- Strategy proofness $\rightarrow$ monotonicity
- Onto + monotonicity $\rightarrow$ unanimity
  $\rightarrow$ Strategy proofness + onto $\rightarrow$ unanimity + monotonicity
  $\rightarrow$ Strategy proofness + onto $\rightarrow$ at least one weak dictator
Thus, if $f$ is onto, either a cheater or a weak dictator (or both)!

[Pin, Rossi, Venable, Walsh, JLC 2009]

Computational aspects of preference aggregation and manipulation

Preventing manipulation?

- A successful manipulation is a way of misreporting one’s preferences that leads to a better result for oneself
- Gibbard-Satterthwaite only tells us that for successful manipulations exist
  - It does not tell us what these manipulations are
- Perhaps we can use complexity as a barrier?
  - Do voting rules exist for which manipulations are computationally hard to find? [Bartholdi, Tovey, Trick 1989]

A formal computational problem

- The simplest version of the manipulation problem:

  CONSTRUCTIVE-MANIPULATION:
  - We are given a voting rule $R$, the (unweighted) votes of the other voters, and a candidate $p$.
  - We are asked if we can cast our (single) vote to make $p$ win.
  - E.g. for the Borda rule:
    - Voter 1 votes $A > B > C$
    - Voter 2 votes $B > A > C$
    - Voter 3 votes $C > A > B$
  - Borda scores are now: $A$: 4, $B$: 3, $C$: 2
  - Can we make $B$ win with our single vote?
  - Answer: YES. Vote $B > C > A$ (Borda scores: $A$: 4, $B$: 5, $C$: 3)
Constructive manipulation

- Manipulation by one voter
  - If this is hard, then it is also with more voters
- Manipulation by coalition of voters
  - More likely to be able to change result
  - More relevant to small committees than general elections?

Bad news: plurality is easy to manipulate by coalition (or single voter)

- If want $p$ to win, the best thing to do is vote for $p$
  - If $p$ then wins, we have manipulated vote
  - If $p$ does not win, there is no manipulation
- Hence, we can decide if plurality can be manipulated in polynomial time

Bad news: Borda is easy to manipulate

- *Greedy* algorithm which finds a manipulation (if one exists)
  - Place $p$ at top of your vote
  - (Repeat) Check every other candidate to see if they can placed next in order without defeating $p$. If so, place them next otherwise declare no manipulation exists
- Hence, we can decide if Borda can be manipulated in polynomial time

Good news: there exist rules which are hard to manipulate

- Theorem. CONSTRUCTIVE-MANIPULATION is NP-complete for the second-order Copeland rule. [Bartholdi, Tovey, Trick 1989]
  - Copeland score = number of victories − number of defeats in pairwise contests
  - Second order Copeland = tiebreak with sum of Copeland scores of alternatives that are defeated
  - Once used by NFL for tie-breaking, used in chess by US Chess Federation and Federation Internationale Des Echecs
Good news: there exist rules which are hard to manipulate

- **Theorem.** CONSTRUCTIVE-MANIPULATION is NP-complete for the STV rule. [Bartholdi, Orlin 1991]
  - Single Transferable Vote repeatedly eliminates the least popular candidate
  - Votes for the least popular candidate are transferred to the next most preferred candidate

- Most other rules are easy to manipulate (in P)

---

“Tweaking” voting rules to make them hard to manipulate

- It would be nice to be able to tweak rules:
  - Change the rule slightly so that
    - Hardness of manipulation is increased (significantly)
    - Many of the original rule’s properties still hold
  - It would also be nice to have a single, universal tweak for all (or many) rules
  - One such tweak: add a **preround** [Conitzer & Sandholm IJCAI ’03]

---

Adding a preround

- A **preround** proceeds as follows:
  - Pair the candidates
  - Each candidate faces its opponent in a pairwise knockout election
  - The winners proceed to the original rule

---

How hard is manipulation when a preround is added?

- Depends on the order of preround matching and vote collection:
  - **Theorem.** NP-hard if preround matching is done first
  - **Theorem.** #P-hard if vote collection is done first
  - **Theorem.** PSPACE-hard if the two are interleaved (for a complicated interleaving protocol)

- In each case, the tweak introduces the hardness for any rule satisfying certain sufficient conditions
  - All of Plurality, Borda, Maximin, STV satisfy the conditions in all cases, so they are hard to manipulate with the preround
What if there are few candidates?

- Hardness to manipulate STV/2nd order Copeland relies on the number of candidates \(m\) being unbounded.
- There is a recursive algorithm for manipulating STV with \(O(1.62^m)\) calls (and usually much fewer).
- E.g. 20 candidates: \(1.62^{20} = 15500\) \[Conitzer PhD 2006\]
- Sometimes the candidate space is large.
  - Voting over allocations of goods/tasks.
  - California governor elections.
- But what if it is not?
  - A typical election for a representative will only have a few.

Manipulation with few candidates

- Ideally, would like hardness results for constant number of candidates.
- But then the manipulator can simply evaluate each possible vote assuming the others’ votes are known.
- Even for coalitions of manipulators, there are only polynomially many effectively different votes.
- However, if we place weights on votes, complexity may return...
  - Weighted case informs case where uncertainty about votes.

<table>
<thead>
<tr>
<th>Unbounded #candidates</th>
<th>Constant #candidates</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unweighted voters</td>
<td>Weighted voters</td>
</tr>
<tr>
<td>Easy</td>
<td>Easy</td>
</tr>
<tr>
<td>Unweighted voters</td>
<td>Weighted voters</td>
</tr>
<tr>
<td>Easy</td>
<td>Potentially hard</td>
</tr>
</tbody>
</table>

Weighted votes

- Used in some elections.
  - Shareholders.
  - Parliaments.
  - ...
- Perhaps more interestingly.
  - Weighted case informs case where uncertainty about the votes.
  - (Informally) weights play role of probabilities.
  - More on this shortly!

Constructive manipulation with weighted votes

- Given weights and votes of the other voters and the weights of a coalition of voters who want to manipulate result.
- Can the coalition make their preferred candidate win?
  - E.g. Borda example:
    - Voter 1 (weight 4): A>B>C, voter 2 (weight 7): B>A>C
    - Manipulators: one with weight 4, one with weight 9
    - Can we make C win?
    - Yes! Solution: weight 4 voter votes C>B>A, weight 9 voter votes C>A>B
Inverse plurality is NP-hard to manipulate with 3 or more candidates

- Plurality
  - each voter has one vote, candidate with most votes wins
- Inverse plurality
  - each voter has one veto, candidate with fewest vetoes wins
  - Sometimes called anti-plurality or negative voting

**Why are many rules easy to manipulate?**

- The best strategy for the manipulators is often to vote identically
- Then the voting rule is easy to manipulate when the number of candidates is fixed
  - Simply check all possible orderings of the candidates (constant)

**Results for constructive manipulation**

<table>
<thead>
<tr>
<th>Number of candidates</th>
<th>2</th>
<th>3</th>
<th>4,5,6</th>
<th>≥ 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Borda</td>
<td>P</td>
<td>NP-c</td>
<td>NP-c</td>
<td>NP-c</td>
</tr>
<tr>
<td>Veto</td>
<td>P</td>
<td>NP-c*</td>
<td>NP-c*</td>
<td>NP-c*</td>
</tr>
<tr>
<td>STV</td>
<td>P</td>
<td>NP-c</td>
<td>NP-c</td>
<td>NP-c</td>
</tr>
<tr>
<td>Plurality with runoff</td>
<td>P</td>
<td>NP-c*</td>
<td>NP-c*</td>
<td>NP-c*</td>
</tr>
<tr>
<td>Copeland</td>
<td>P</td>
<td>P*</td>
<td>NP-c</td>
<td>NP-c</td>
</tr>
<tr>
<td>Minimax</td>
<td>P</td>
<td>P*</td>
<td>NP-c</td>
<td>NP-c</td>
</tr>
<tr>
<td>Randomized cup</td>
<td>P</td>
<td>P*</td>
<td>NP-c</td>
<td>NP-c</td>
</tr>
<tr>
<td>Regular cup</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>P</td>
</tr>
<tr>
<td>Plurality</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>P</td>
</tr>
</tbody>
</table>

Complexity of CONSTRUCTIVE CW-MANIPULATION

[Conitzer, Sandholm, Lang, JACM 53 (3), 2007]
Destructive manipulation with weighted votes

- Exactly the same, except:
- Instead of a preferred candidate
- We now have a hated candidate
- Our goal is to make sure that the hated candidate does not win (whoever else wins)
  - Destructive manipulation can be easy even though constructive manipulation is hard
  - If destructive manipulation is hard then so is constructive manipulation
  - Reverse does not hold
    - E.g. Borda is polynomial to manipulate destructively but NP-hard constructively for 3 or more candidates

Results for destructive manipulation

<table>
<thead>
<tr>
<th>Number of candidates</th>
<th>2</th>
<th>≥ 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>STV</td>
<td>P</td>
<td>NP-(c^*)</td>
</tr>
<tr>
<td>plurality with runoff</td>
<td>P</td>
<td>NP-(c^*)</td>
</tr>
<tr>
<td>randomized cup</td>
<td>P</td>
<td>?</td>
</tr>
<tr>
<td>Borda</td>
<td>P</td>
<td>P</td>
</tr>
<tr>
<td>veto</td>
<td>P</td>
<td>P*</td>
</tr>
<tr>
<td>Copeland</td>
<td>P</td>
<td>P</td>
</tr>
<tr>
<td>maximin</td>
<td>P</td>
<td>P</td>
</tr>
<tr>
<td>regular cup</td>
<td>P</td>
<td>P</td>
</tr>
<tr>
<td>plurality</td>
<td>P</td>
<td>P</td>
</tr>
</tbody>
</table>

Complexity of destructive cw-manipulation

[Conitzer, Sandholm, Lang, JACM 53 (3), 2007]

Uncertainty about votes

- Suppose we have some probability distribution over votes
- Weighted manipulation informs us about complexity of reasoning about such uncertainty
  - Thm: Constructive manipulation with weighted votes is NP-hard implies computing probability of candidate winning given uncertain votes is NP-hard

Preference elicitation

- Some preferences may be missing
- Elicitation closely related to manipulation
- Time consuming, costly, difficult, ...
  - Famous 7 questions!
- Want to terminate elicitation as soon as winner fixed
  - Obama must now win however remaining states vote

[Conitzer, Sandholm, Lang, JACM 53 (3), 2007]
Possible and necessary winners

- **Necessary winner**
  - However remaining votes are cast, they must win
- **Possible winner**
  - There is a way for remaining votes to be cast so that they win

[Konczak and Lang, IJCAI-05 preference workshop]

Possible and necessary winners

- Closely connected to manipulation
  - $p$ is possible winner iff there is a constructive manipulation for $p$
    - *Clinton is a possible winner and so can still manipulate a future in which she wins!*
  - $p$ is a necessary winner iff there is not a destructive manipulation for $p$
    - *Once Obama wins Pennsylvania and is a necessary winner, there is no way for the vote to be manipulated destructively so he is not chosen*

[Konczak and Lang, IJCAI-05 preference workshop]

Possible and necessary Condorcet winner

- Closely connected to preference elicitation
  - Elicitation can only be terminated iff possible winners = necessary winner
  - Deciding elicitation is over is in P => computing possible (and necessary) winners is also

[Walsh, AAMAS 2008]

Possible and necessary Condorcet winner

- Condorcet winner
  - Beats all others in pairwise contests
- Possible Condorcet winner
  - Some way to complete votes so Condorcet winner
- Necessary Condorcet winner
  - Condorcet winner however votes completed

[Walsh, AAMAS 2008]
Possible and necessary Condorcet winner

- Polynomial to compute
  - Even if votes are weighted and large number of candidates
  - To find necessary Condorcet winners, see if one candidate has at least half votes against every other candidate
  - To find possible Condorcet winners, put each candidate at top of incomplete votes
  - Hence can decide in polynomial time when to terminate preference elicitation when electing Condorcet winner

[Walsh, AAMAS 2008]

Possible and necessary Condorcet winner

- Polynomial to compute
  - Good news
    - Many authorities have argued that Condorcet winners should be elected when they exist

[Walsh, AAMAS 2008]

Manipulating Condorcet winner

- Polynomial to decide if coalition of voters can manipulate Condorcet winner
  - Each member of coalition just puts desired candidate top of their vote!
  - Bad news: we don’t want voting to be (easy to be) manipulable
  - Slightly good news: Condorcet consistent rules can still be hard to manipulate (e.g. 2nd order Copeland) but only in what they do when there is no Condorcet winner

[Walsh, AAMAS 2008]

Computing possible & necessary winners

- Consider specific voting rules
  - Unweighted votes
    - Arbitrary number of candidates
      - For STV, computing possible winners is NP-hard, and necessary winners is coNP-hard
      - Even NP-hard to approximate set of possible winners within constant factor in size
      - Many other rules easy!

[Pini, Rossi, Venable, Walsh, IJCAI 2007]
Computing possible & necessary winners

- Weighted votes
- Fixed number of candidates
  - NP-hard for Borda, veto, STV with 3 or more votes
  - NP-hard for Copeland & Simpson with 4 or more candidates
  ...

Cup rule

- Binary voting tree $T \rightarrow$ voting rule $r_T$
- $r_T$: majority graph $G \rightarrow$ candidate (winner)
- Sequence of pairwise comparisons (also called agenda) between candidates

Cup rule

- Easy to manipulate by coalition
  - Constructively or destructively
  - Weighted or unweighted votes
  - Introduce randomness (and 7 candidates) to make it NP-hard
- NP-hard to manipulate individual preferences
  - 3 or more candidates, weighted votes
  - May not be able to change whole vote but just preferences between particular candidates

Cup rule

- Easy to manipulate by coalition
  - For simplicity, consider balanced tree and $p$ is leftmost leaf
  - In each subtree, to make $p$ win, must be a winner of left subtree, and beat one of winners of right subtree
    - Then coalition put all candidates in left subtree above those in right
  - Simple recursive algorithm (remember depth is log of candidates) is polynomial

[Conitzer, Sandholm, Lang JACM 2007], [Walsh, AAMAS 2008]
Cup rule

- Preference elicitation
- Two different sources of uncertainty
  - Votes
  - Agenda

Incomplete preferences

- **Weak possible (WP) winner A**: ∃ completion of profile, ∃ voting tree s.t. A wins
- **Strong possible (SP) winner A**: ∀ completion of profile, ∃ voting tree s.t. A wins
- **Weak Condorcet (WC) winner A**: ∃ completion of profile, ∀ voting tree s.t. A wins
- **Strong Condorcet (SC) winner A**: ∀ completion of profile, ∀ voting tree s.t. A wins

\[
\text{SC} \subseteq \text{WC} \cap \text{SP} \quad \text{WC} \cup \text{SP} \subseteq \text{WP}
\]

Fair Weak and Strong Possible Winners

- Some possible winners may win only on very unbalanced trees, competing only few times. **UNFAIR**!

- **Fair weak possible (FWP) winner A**: ∃ completion of maj. graph/profile, ∃ balanced voting tree s.t. A wins
- **Fair strong possible (FSP) winner A**: ∀ completion of maj. graph/profile, ∃ balanced voting tree s.t. A wins

Fixed trees: weak and strong winners

- **T**: binary voting tree
- **A**: a candidate
  - **Strong winner (SW) winner A**: ∀ completion of maj. graph/profile, A wins in the fixed tree T
  - **Weak winner (WW) winner A**: ∃ completion of maj. graph/profile, A wins in the fixed tree T
Manipulation with single peaked votes

- With single peaked votes, necessary and possible Condorcet winners are polynomial
  - Find leftmost & rightmost possible winner
  - If they’re the same, this is necessary winner
  - Possible winners are all candidates between leftmost and rightmost possible winners

- Possible and necessary winners for STV
  - Remains NP-hard with just 3 candidates and weighted votes

- Constructive and destructive manipulation of STV
  - Remains NP-hard with just 3 candidates and weighted votes

[Walsh, AAMAS 2008]
Pre-rounds

- Plurality rule
  - Polynomial to decide when to terminate elicitation (good)
  - Polynomial to manipulate (bad)
- Pre-round then plurality
  - Remains polynomial to decide when to terminate elicitation (good)
  - Becomes NP-hard to manipulate (good)
  - Illustrates tension between complexity of manipulation and deciding the termination of preference elicitation

Matching Problems

Motivation

- Agents may express preferences for issues other than a collective decision
  - room-mates, work assignments, …
- All examples of matching problems
  - Students with Rooms, Doctors with Hospitals, …

Stable marriage

- Mathematical abstraction
- Two sets of agents: men and women
- Idealized model
  - All men totally order all women, and vice-versa
Stable marriage

- Given preferences of \( n \) men
  - Greg: Amy > Bertha > Clare
  - Harry: Bertha > Amy > Clare
  - Ian: Amy > Bertha > Clare

- Given preferences of \( n \) women
  - Amy: Harry > Greg > Ian
  - Bertha: Greg > Harry > Ian
  - Clare: Greg > Harry > Ian

- Find a *stable marriage*

Stable marriage

- Given the preferences of the \( n \) men over the \( n \) women, and of the \( n \) women over the \( n \) men

- Find a *stable marriage*
  - Assignment of men to women (or equivalently of women to men)
    - Idealization: everyone marries at the same time
    - No pair (man, woman) not married to each other would prefer to run off together
      - Idealization: assumes no barrier to divorce!

Stable marriage

- Unstable solution
  - Greg: Amy > Bertha > Clare
  - Harry: Bertha > Amy > Clare
  - Ian: Amy > Bertha > Clare
  - Amy: Harry > Greg > Ian
  - Bertha: Greg > Harry > Ian
  - Clare: Greg > Harry > Ian

*Bertha & Greg would prefer to elope*

Stable marriage

- One solution
  - Greg: Amy > Bertha > Clare
  - Harry: Bertha > Amy > Clare
  - Ian: Amy > Bertha > Clare
  - Amy: Harry > Bertha
  - Bertha: Greg > Harry > Ian
  - Clare: Greg > Harry > Ian

*Men do ok, women less well*
Stable marriage

- Another solution
  - Greg: Amy>Bertha>Clare
  - Harry: Bertha>Amy>Clare
  - Ian: Amy>Bertha>Clare

  - Amy: Harry>Greg>Ian
  - Bertha: Greg>Harry>Ian
  - Clare: Greg>Harry>Ian

Women do ok, men less well

Gale Shapley algorithm

- Initialize every person to be free
- While exists a free man
  - Find best woman he hasn’t proposed to yet
  - If this woman is free, declare them engaged
  - Else if this woman prefers this proposal to her current fiancee then declare them engaged (and “free” her current fiancee)
  - Else this woman prefers her current fiancee and she rejects the proposal

Gale Shapley algorithm

- Initialize every person to be free
- While exists a free man
  - Find best woman he hasn’t proposed to yet
  - If this woman is free, declare them engaged
  - Else if this woman prefers this proposal to her current fiancee then declare them engaged (and “free” her current fiancee)
  - Else this woman prefers her current fiancee and she rejects the proposal

- Terminates with everyone matched
  - Suppose some man is unmatched at the end
  - Then some woman is also unmatched
  - But once a woman is matched, she only “trades” up
  - Hence this woman was never proposed to
    - But if a man is unmatched, he has proposed to and been rejected by every woman
  - This is a contradiction as he has never proposed to the unmatched woman!
Gale Shapley algorithm

- Terminates with perfect matching
  - Suppose there is an unstable pair in the final matching
  - Case 1. This man never proposed to this woman
    - As men propose to women in preference order, man must prefer his current fiancee
    - Hence current pairing is stable!
  - Case 2. This man had proposed to this woman
    - But the woman rejected him (immediately or later)
    - However, women only ever trade up
    - Hence the woman prefers her current partner
    - So the current pairing is stable!

Gale Shapley algorithm

- Terminates with perfect matching
  - Suppose some man is engaged to someone who is not the best possible woman
  - Then they have proposed and been rejected by this woman
  - Consider first such man A, who is rejected by X in favour ultimately of marrying B
    - There exists (some other) stable marriage with A married to X and B to Y
  - By assumption, B has not yet been rejected by his best possible woman
  - Hence B must prefer X at least as much as his best possible woman
  - So (A,X) (B,Y) is not a stable marriage as B and X would prefer to elope!
**Gale Shapley algorithm**

- GS finds woman pessimal solution
  - Suppose some woman is engaged to someone who is not the worst possible man
    - Let (A,X) be married but A is not worst possible man for X
    - There exists a stable marriage with (B,X) (A,Y) and B worse than A for X
    - By male optimality, A prefers X to Y
    - Then (A,Y) is unstable!

**Gale Shapley algorithm**

- Initialize every person to be free
- While exists a free man
  - Find best woman he hasn’t proposed to yet
  - If this woman is free, declare them engaged
    - Else if this woman prefers this proposal to her current fiancee then declare them engaged (and “free” her current fiancee)
    - Else this woman prefers her current fiancee and she rejects the proposal

**Extensions: ties**

- Cannot always make up our minds
- Preference ordering: total order with ties
- Stability
  - (weak) no couple strictly prefers each other
  - (strong) no couple such that one strictly prefers the other, and the other likes them as much or more

```
Greg: Amy>Bertha>Clare
Harry: Bertha>Amy>Clare
Ian: Amy>Bertha>Clare

Amy: Greg>Harry>Ian
Bertha: Greg>Bertha>Harry
Clare: Greg>Bertha>Harry
```

```
**Extensions: ties**

- **Stability**
  - (weak) no couple strictly prefers each other
  - (strong) no couple such that one strictly prefers the other, and the other likes them as much or more

- **Existence**
  - Strongly stable marriage may not exist
  - \(O(n^4)\) algorithm for deciding existence
  - Weakly stable marriage always exists
  - Just break ties arbitrarily
  - Run GS, resulting marriage is weakly stable!

**Extensions: incomplete preferences**

- There are some people we may be unwilling to marry
  - I’d prefer to remain single than marry Margaret

- \((m,w)\) unstable iff
  - \(m\) and \(w\) do not find each other unacceptable
  - \(m\) is unmatched or prefers \(w\) to current fiancee
  - \(w\) is unmatched or prefers \(w\) to current fiancee

**Extensions: incomplete prefs**

- GS algorithm
  - Extends easily

- Men and woman partition into two sets
  - Those who have partners in all stable marriages
  - Those who do not have partners in any stable marriage

**Extensions: ties & incomplete prefs**

- Weakly stable marriages may be different sizes
  - Unlike with just ties where they are all complete

- Finding weakly stable marriage of max. cardinality is NP-hard
  - Even if only women declare ties
Strategy proofness

- GS is strategy proof for men
  - Assuming male optimal algorithm
  - No man can do better than the male optimal solution
- However, women can profit from lying
  - Assuming male optimal algorithm is run
  - And they know complete preference lists

Impossibility of strategy proofness

- [Roth 82]
  - No matching procedure for which stating the truth is a dominant strategy for all agents when preference lists can be incomplete
- Consider
  - Greg: Amy>Bertha
  - Harry: Bertha>Amy
  - Ian: Amy>Bertha
  - Amy: Harry>Greg
  - Bertha: Greg>Harry
  - Clare: Greg>Harry
- Two stable marriages:
  - (Greg,Amy)(Harry,Bertha) or (Greg,Bertha)(Harry,Amy)

Strategy proofness

- Greg: Amy>Bertha>Clare
- Harry: Bertha>Amy>Clare
- Ian: Amy>Bertha>Clare
- Amy lies
- Amy: Harry>Greg>Ian
- Bertha: Greg>Harry>Ian
- Clare: Greg>Harry>Ian

Impossibility of strategy proofness

- Consider
  - Greg: Amy>Bertha
  - Harry: Bertha>Amy
  - Ian: Amy>Bertha
  - Amy: Harry>Greg
  - Bertha: Greg>Harry
  - Clare: Greg>Harry
- Two stable marriages:
  - (Greg,Amy)(Harry,Bertha) or (Greg,Bertha)(Harry,Amy)
- Suppose we get male optimal solution
  - (Greg,Amy)(Harry,Bertha)
  - If Amy lies and says Harry is only acceptable partner
  - Then we must get (Harry,Amy)(Greg,Bertha) as this is the only stable marriage
  - Other cases can be manipulated in a similar way
Making manipulation hard

- Can we make the manipulation hard to find?
  - As with voting, this may be a barrier to mis-reporting of preferences
  - Complexity can again be our friend!

Solution: gender swapping

- Basic idea
  - Men have no incentive to manipulate GS
  - But women do

- Construct SM procedure that may swap men with women

Solution: gender swapping

- Toss a coin
  - Heads: men stay men
  - Tails: men become women and vice versa

- No incentive to mis-report preferences
  - 50% chance that it will hurt

Solution: gender swapping

- Toss a coin
  - Heads: men stay men
  - Tails: men become women and vice versa

- Not everyone likes
  - Randomized procedures
  - Probabilistic guarantees
Solution: deterministic choice

- Pick a set of stable matchings
  - Male and female-optimal
  - All stable matchings
  - f(M,W) union f(W,M)
    - f is any procedure computing one or more stable matchings
  - ...

Solution: deterministic choice

- Pick a set of stable matchings
- Choose between them based on agents’ preferences
  - Make this choice difficult to manipulate!

Solution: deterministic choice

- Simple (but un-natural) SM procedure to prove this can be computationally hard
  - Manipulator’s preferences = witness to NP-complete problem
  - Other agents’ preferences = instance of NP-complete problem
  - Swap men with women iff witness is a solution
  - Then run GS algorithm

Solution: deterministic choice

- Pick a set of stable matchings
- Choose between them based on agents’ preferences
  - Make this choice difficult to manipulate!
  - More natural procedure that is based on voting
    - Complexity of manipulating voting rule => complexity of manipulating SM procedure
Solution: deterministic choice

- Pick a set of stable matchings
- Choose between them based on agents’ preferences
  - Run a STV election to order men by women’s preferences (and women by men)
  - Compute a regret vector using this order
    - $i^{th}$ component of vector is rank of $i^{th}$’s spouse in their preference ordering
    - Pick SM with lex smallest regret

Solution: deterministic choice

- Pick a set of stable matchings
- Choose between them based on agents’ preferences
  - Run a STV election to order men by women’s preferences (and women by men)
  - Compute a regret vector using this order
  - Pick SM with lex smallest regret

Conclusions

- Compact preference modelling
- Comparison of their expressive power and computational properties
- Ability to reason with more than one formalism in the same problem
- Handling uncertainty and vagueness
  - look for good solutions robust to uncertainty
  - resort to elicitation

Conclusions

- Computational complexity is an important issue in
  - Manipulation
  - Preference elicitation
- Complexity can be a friend or foe
  - Ideally want it to be hard to find manipulation but easy to decide when to stop eliciting preferences!
- But NP-hardness is only worst case
  - See [Walsh, Where are the really hard manipulation problems?, IJCAI-09]
References: preferences, soft constraints and CP-nets

- Extended semantics and optimization algorithms for CP-nets, R. Brafman and Y. Dimopoulos, Computational Intelligence, 20(2), 2004
- Constraint processing, R. Dechter, Morgan Kaufmann, 2003
- Constraint-based Preference Optimization, S. Prestwich, F. Rossi, K. B. Venable, T. Walsh, AAAI 2005
- Constraint-based Preferential Optimization, S. Prestwich, F. Rossi, K. B. Venable, T. Walsh, AAAI 2005
- Handbook of constraint programming, Rossi, Van Beek, Walsh eds., Elsevier, 2006
- Possibilistic Constraint Satisfaction Problems or “How to Handle Soft Constraints?”. T. Schiex, UAI 1992

References

- Preferences and uncertainty
- Gelain, Pini, Rossi, Venable, Wilson, Imprecise Soft Constraint Problems, AAAI08 workshop on advances in preferences handling, 2008

References

- Voting theory

References

- Manipulation
  - Bartholdi, Tovey, Trick. The computational difficulty of manipulating an election. Social Choice and Welfare, 6(3):227-241, 1989
  - Walsh. Where are the really hard manipulation problems? The phase transition in manipulating the veto rule. UCIAI-2009.
References

- **Elicitation, uncertainty**

- **Stable marriage**