Complexity of Voting Systems

Piotr Faliszewski
AGH University
Kraków, Poland

Agenda

- A First Course in Complexity Theory
  - Complexity classes P and NP.
  - NP-completeness
  - Dealing with NP-completeness
- Complexity is Bad
  - Winner determination problems
    - Dodgson, Kemeny, Young…
    - Monroe, Chamberlin-Courant
    - Way around!
- Complexity is Good
  - The complexity barrier approach
  - Fighting Gibbard-Satterhwaite
  - Fighting other deamons…
  - … and not winning

Agenda

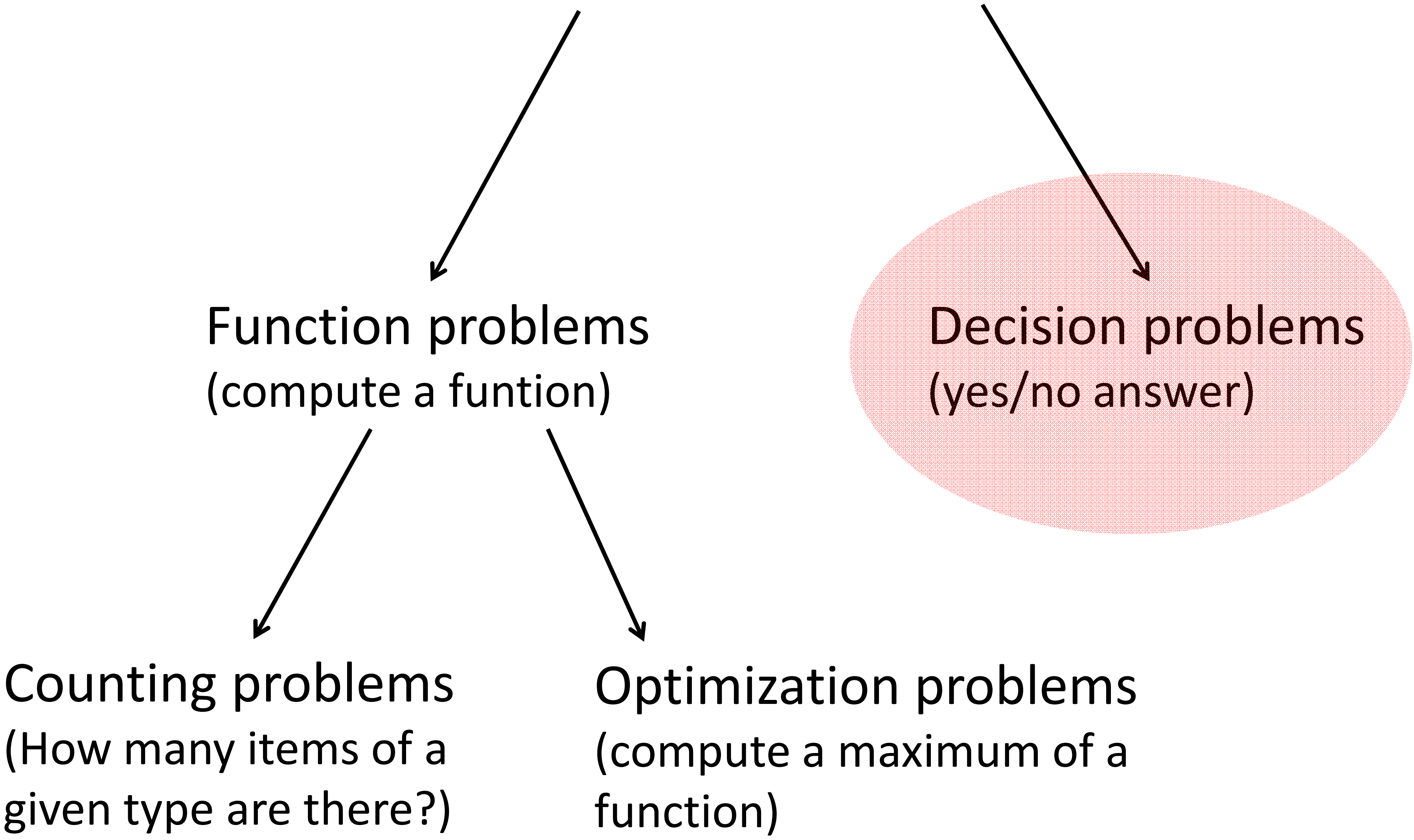
- **A First Course in Complexity Theory**
  - **Complexity classes P and NP.**
  - **NP-completeness**
  - **Dealing with NP-completeness**
- Complexity is Bad
  - Winner determination problems
    - Dodgson, Kemeny, Young…
    - Monroe, Chamberlin-Courant
    - Way around!
- Complexity is Good
  - The complexity barrier approach
  - Fighting Gibbard-Satterhwaite
  - Fighting other deamons…
  - … and not winning

What is complexity theory?

Computational complexity theory – a formal theory that identifies and explains which tasks can be efficiently carried out on a computer.

- **Sort of…**
- **What do you mean?**
- **It will take 10’000 years**
- **It’s not going to be very useful then, will it?**
- **Not particularly…**
- **Why shouldn’t I fire you?**
- **Because there is noone better…**

Computational Problems



Why Decision Problems?

Because they suffice…

Primes
Input: n – an integer
Task: compute the smallest prime factor of n

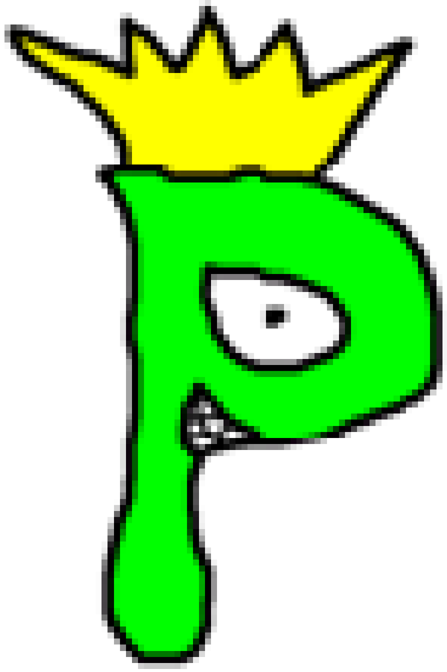
If we can solve Primes, then we can solve PrimesDecision.

If we can solve PricesDecision, we can also solve Primes!

PrimesDecision
Input: n, k – integers
Question: Is n's smallest prime factor smaller or equal to k?

If we can solve Primes…

… but what does it even mean? Obviously we can solve Primes – just divide n by all number from 2 to n-1

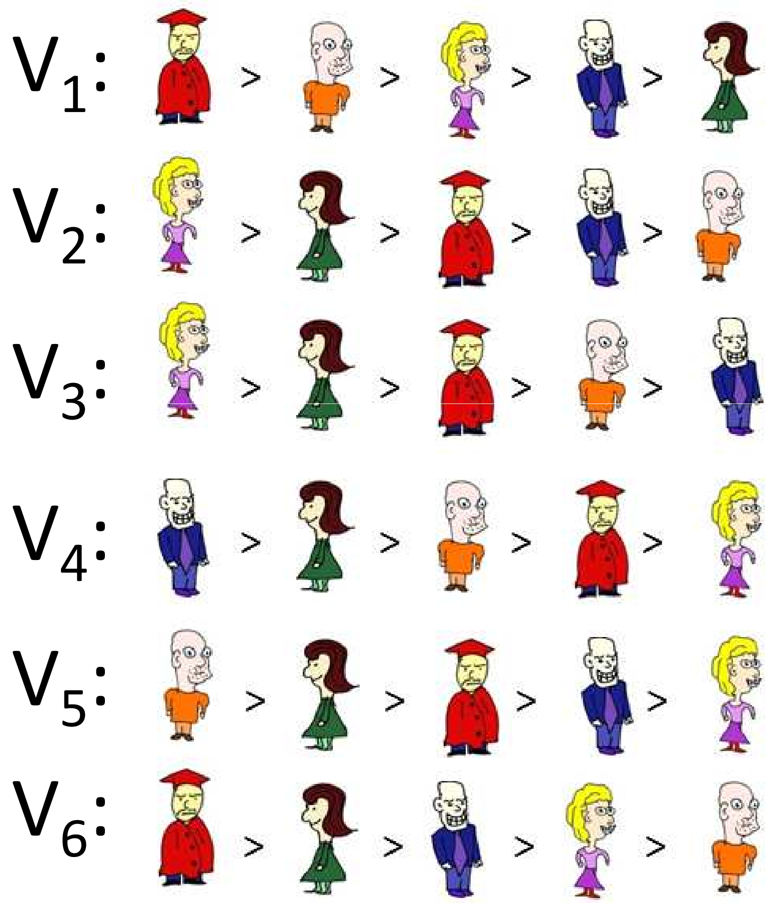


Complexity class P (polynomial-time): The class of decision problems for which there are polynomial-time , deterministic algorithms.

The notion of effective computation!

Borda-Winner is in P

Borda-Winner
Input: P=(P_1, …, P_n) is a profile of preference orders, c – a candidate from P
Question: Is c a Borda winner under profile P?



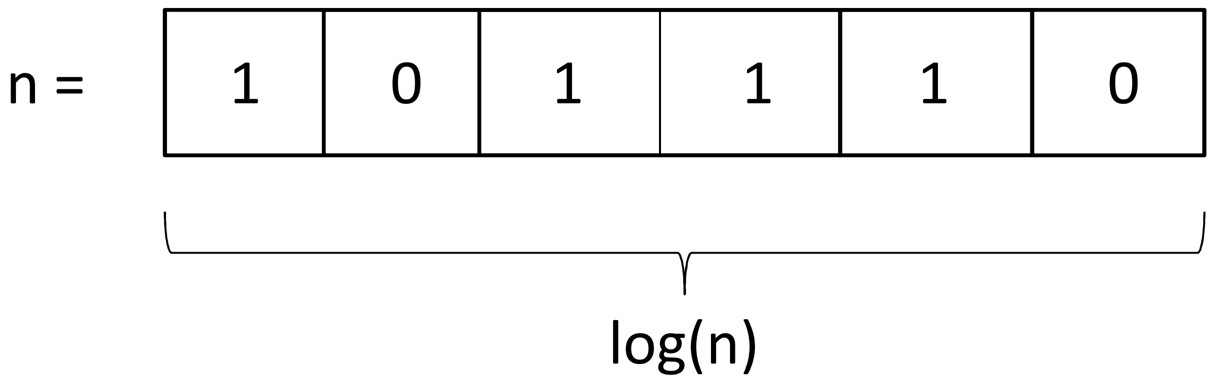
Input size: n voters x m candidates

Algorithm:
For each candidate compute his/her Borda score Check if c has highest Borda score.

Running time: O(nm) ← polynomial!

Primes is in P… but not as we thought it!

Simply dividing n by 2, 3, … , n-1 is an exponential time algorithm!



Doing O(n) divisions means, in fact, doing $O(2^{\log(n)})$ divisions—exponential within the length of the encoding.

There is a more complexy proof that Primes is in P though…

Class P

A decision problem D is in class P if there exists an algorithm that given input I for D, solves I in time polynomial with respect to the length of the encoding of I.

Examples of P-time running times (n – size of the input):

- n^2
- n log n
- n^{1000}

Examples of running times not in P:

- 2^n
- 1.00000000000001^n

Sort of silly, but the best we have...

Computationally Hard Problems

What does it mean that a problem is computationally hard?

- No polynomial time algorithm!
- Can we prove that such problems exist?
  - Yes…
  - … but it's useless in most cases

A different computational complexity class…

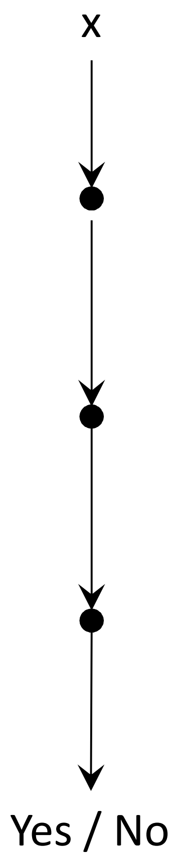
Compexity Class NP

Complexity class NP (nondeterministic polynomial-time): The class of decision problems for which there are polynomial-time , nondeterministic algorithms.

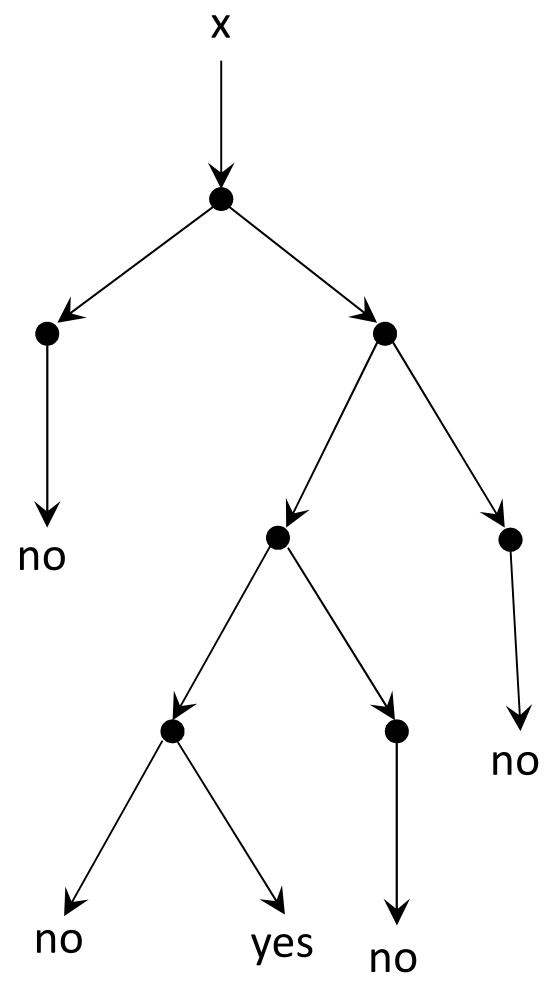
What is a nondeterministic computation?

(Non)deterministic Computation

Deterministic computation

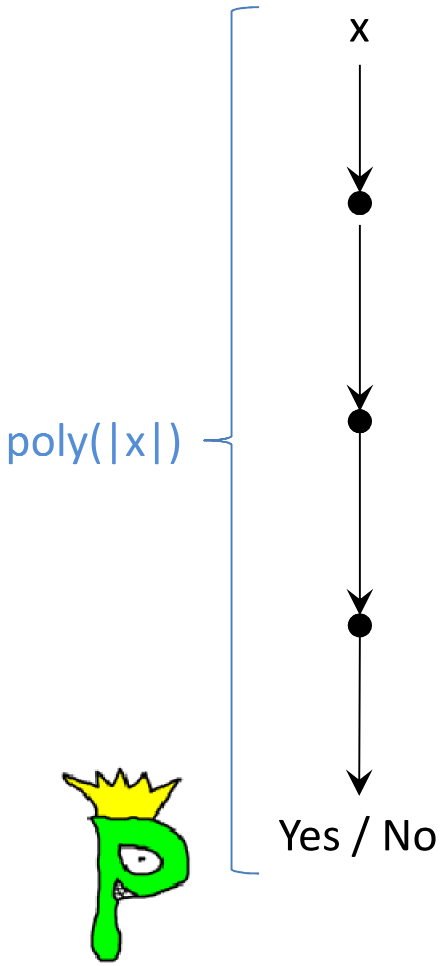


Nondeterministic computation

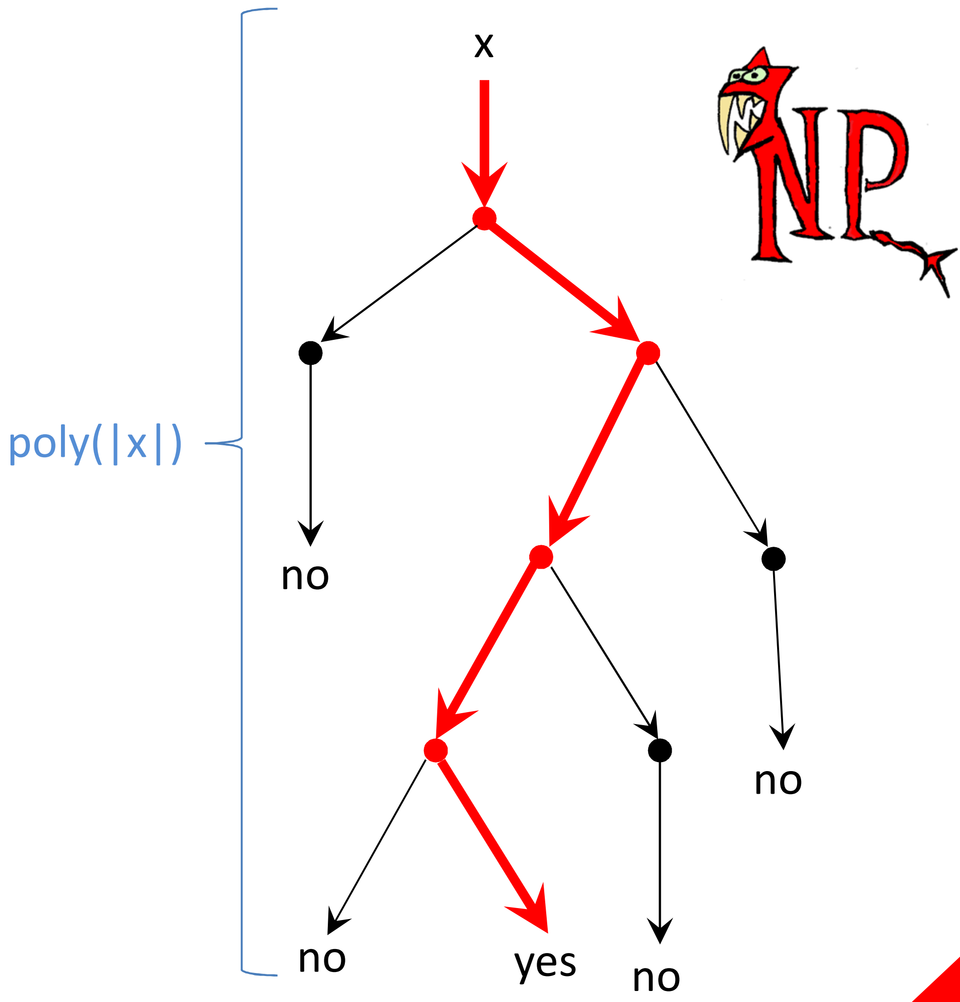


(Non)deterministic Computation

Deterministic computation



Nondeterministic computation



What does it all mean?

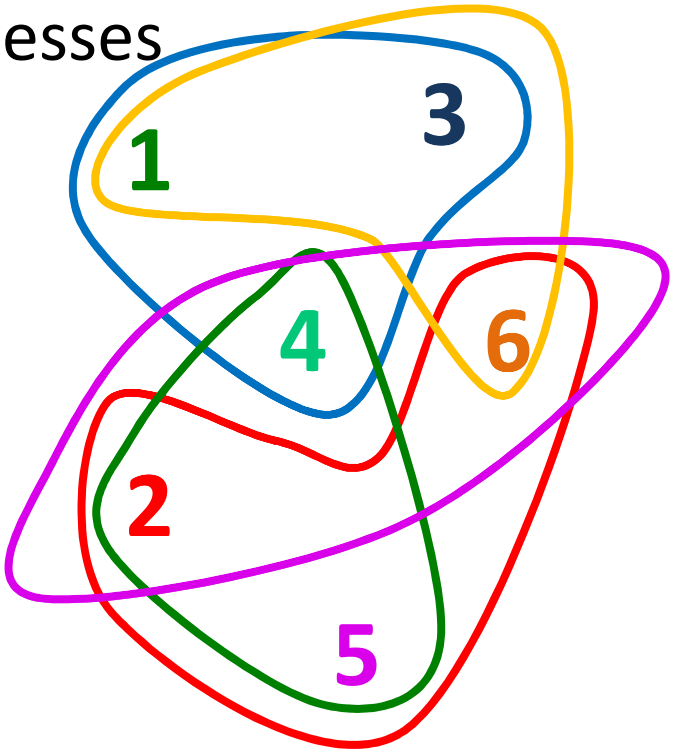
Nondeterministic computation

- Just like normal computation …
- … but the algorithm can make guesses

SetCover

Input: $S = \{S_1, \ldots S_m\}$ – family of sets
k – an integer

Question: Is there a family of k sets from S whose union is equal to union of all sets from S?

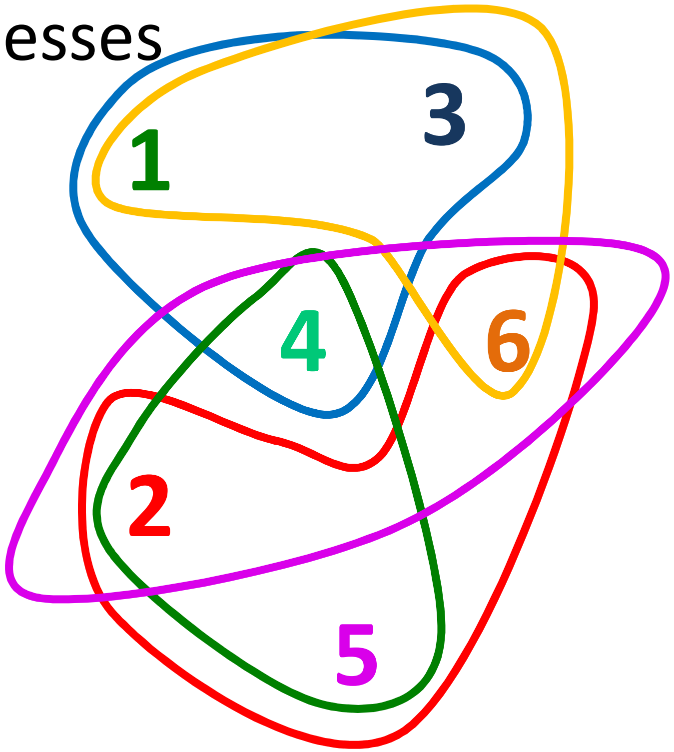


What does it all mean?

Nondeterministic computation

- Just like normal computation …
- … but the algorithm can make guesses

SetCover
Input: S = {S_1, … S_m} – family of sets
k – an integer
Question: Is there a family of k sets from S whose union is equal to union of all sets from S?



Compexity Class NP

Complexity class NP (nondeterministic polynomial-time): The class of decision problems for which there are polynomial-time , nondeterministic algorithms.

Class NP: Class of problems whose solutions can be verified in polynomial time.

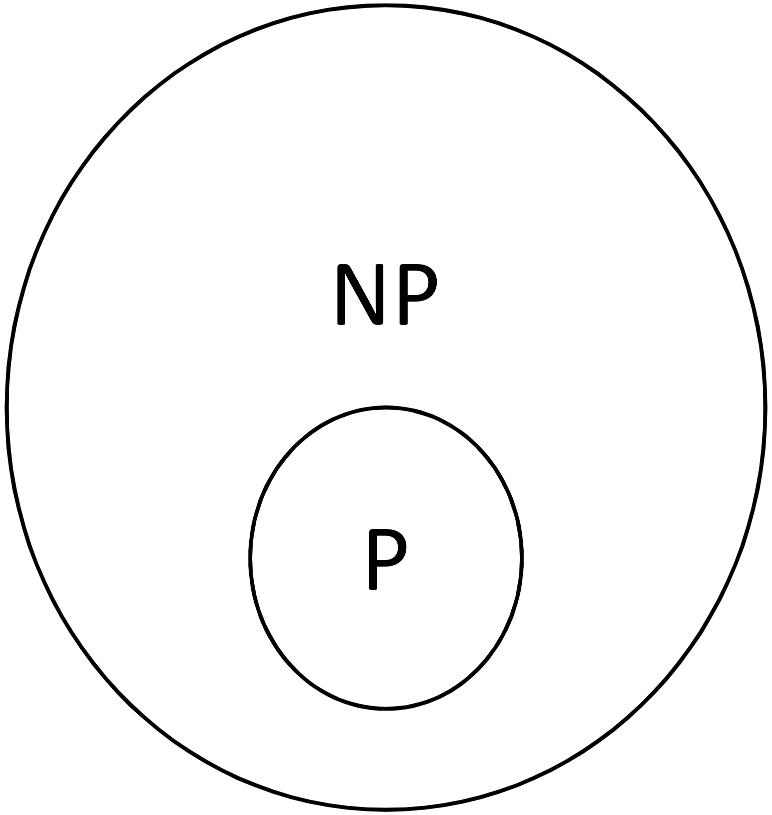
Compexity Class NP

Complexity class NP (nondeterministic polynomial-time): The class of decision problems for which there are polynomial-time , nondeterministic algorithms.

Class NP: Class of problems whose solutions can be verified in polynomial time.

In effect, NP is exactle the class that captures most of voting related problems. Is there a successful manipulation? If there is one, we can show it and verify that, indeed, it is successful.

Is NP bigger than P? – that is the question!



Clearly, all problems from P also belong to NP. What aboue the other way round?

One of the biggest questions in … well... all of science ☺

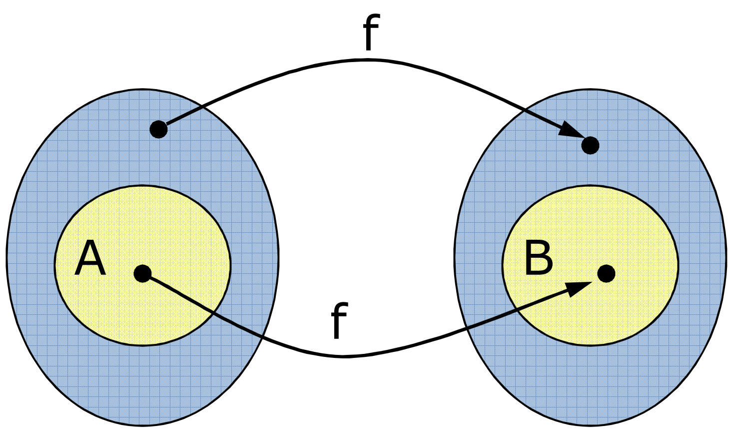
If we do not know if NP is bigger, how can it help us? There is an order on the hardness of problems!

Partial Order of Hardness

Reduction between problems

- A, B – two decision problems
- A reduces to B if there is a polynomial-time computable function f such that

x in A ⇔ f(x) in B



If A reduces to B, then A is no harder than B → If we could solve B, we could solve A as well.

Example of a Reduction

SAT-3CNF
Input: Logical formuka F in 3CNF form
Question: Is F satisfiable?

reduces to

SetCover
Input: $S = \{S_1, \dots S_m\}$ – family of sets
k – an integer
Question: Is there a family of k sets from S whose union is equal to union of all sets from S?

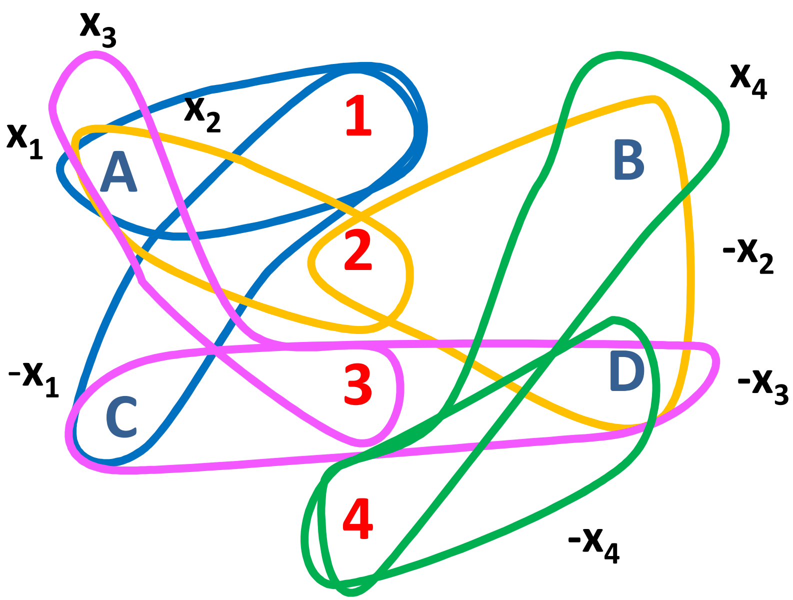
Example of a Reduction

SetCover instance:

$$(x_1 \vee x_2 \vee x_3)\ (-x_2 \vee x_4)\ (-x_1 \vee -x_3)\ (-x_2 \vee -x_3 \vee -x_4)$$

A B C D

SetCover instance:



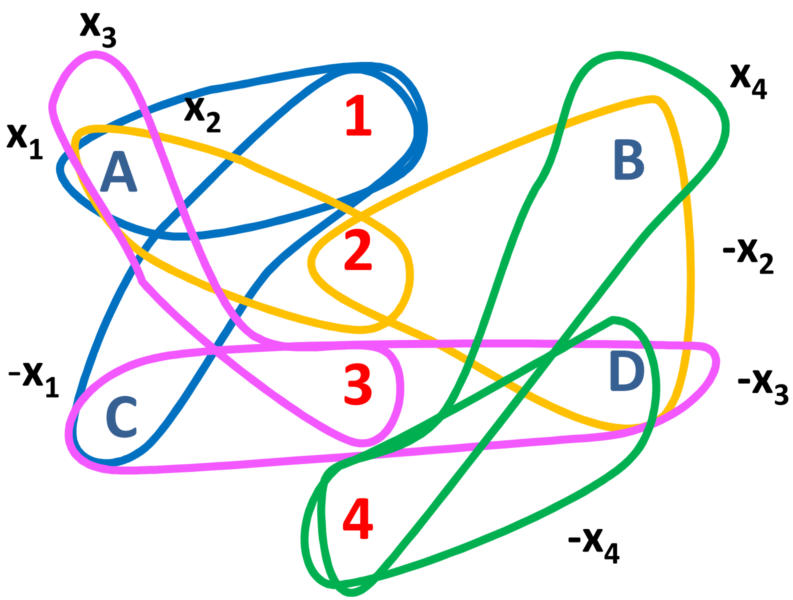
Example of a Reduction

SetCover instance:

$$(x_1 \vee x_2 \vee x_3)\ (-x_2 \vee x_4)\ (-x_1 \vee -x_3)\ (-x_2 \vee -x_3 \vee -x_4)$$

A B C D

SetCover instance:



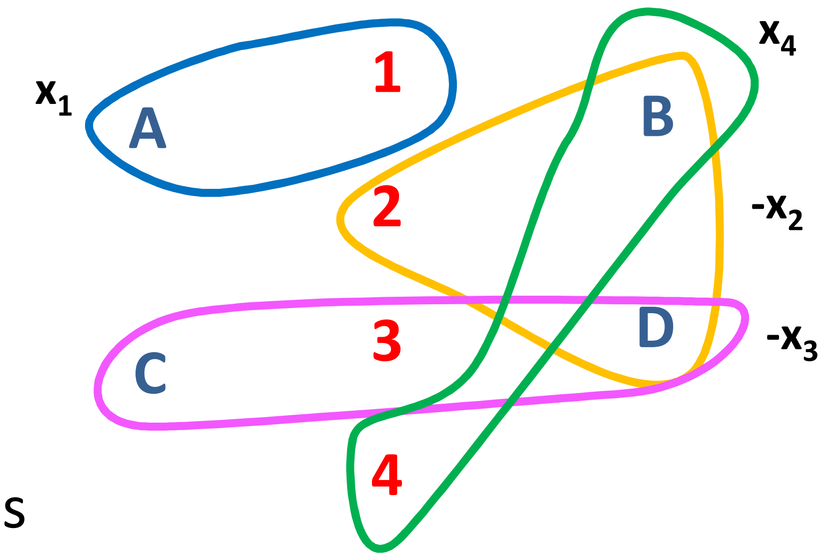
We can take 4 sets

Example of a Reduction

<u>SetCover instance:</u>

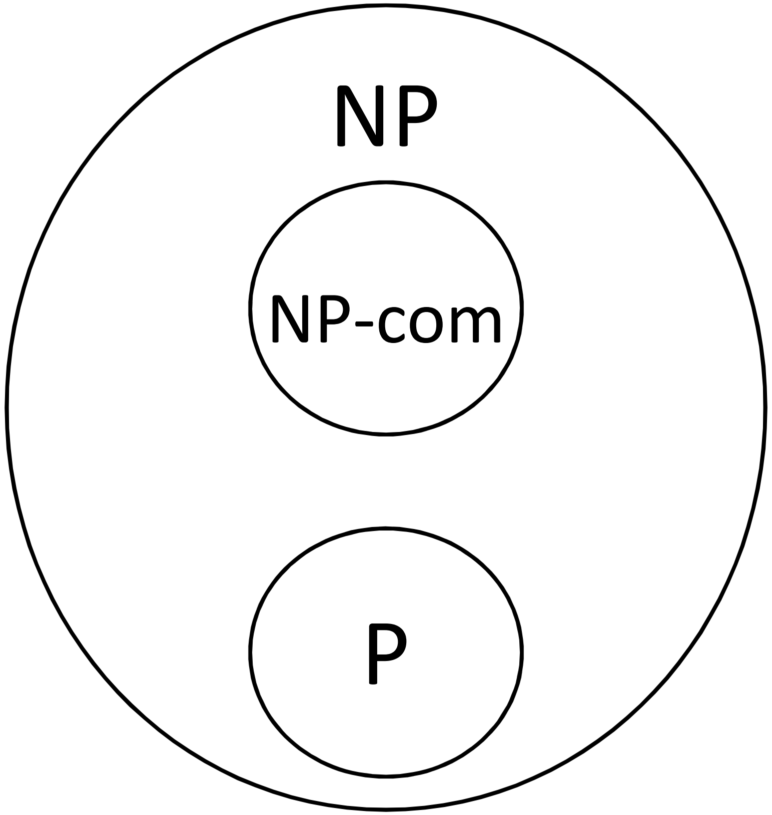
$$\underbrace{(x_1 \vee x_2 \vee x_3)}_{A}\underbrace{(-x_2 \vee x_4)}_{B}\underbrace{(-x_1 \vee -x_3)}_{C}\underbrace{(-x_2 \vee -x_3 \vee -x_4)}_{D}$$

<u>SetCover instance:</u>



We can take 4 sets

Is NP bigger than P? – that is the question!



NP-completeness: A problem is NP- complete if it is in NP and every problem from NP reduces to it ➔ The hardest problems in NP!

SAT-3CNF is NP-complete…

… so SetCover is too!

NP-completeness

Definition: A problem is NP-complete if it belongs to NP and every problem in NP reduces to it

Proving NP-completeness: Tak an NP-complete problem and reduce it to your problem of interest (reductions are transitive!)

NP-complete problems are hard: No polynomial time algorithm known for them, in spite of decades of search! A natural notion of hardness!

NP-complete Problems: Examples

SetCover

Inp

Qu

X3C

In

Q

VertexCover

Inp

Qu

Partition

Input: $s_1, \ldots, s_n$ – sequence of integers

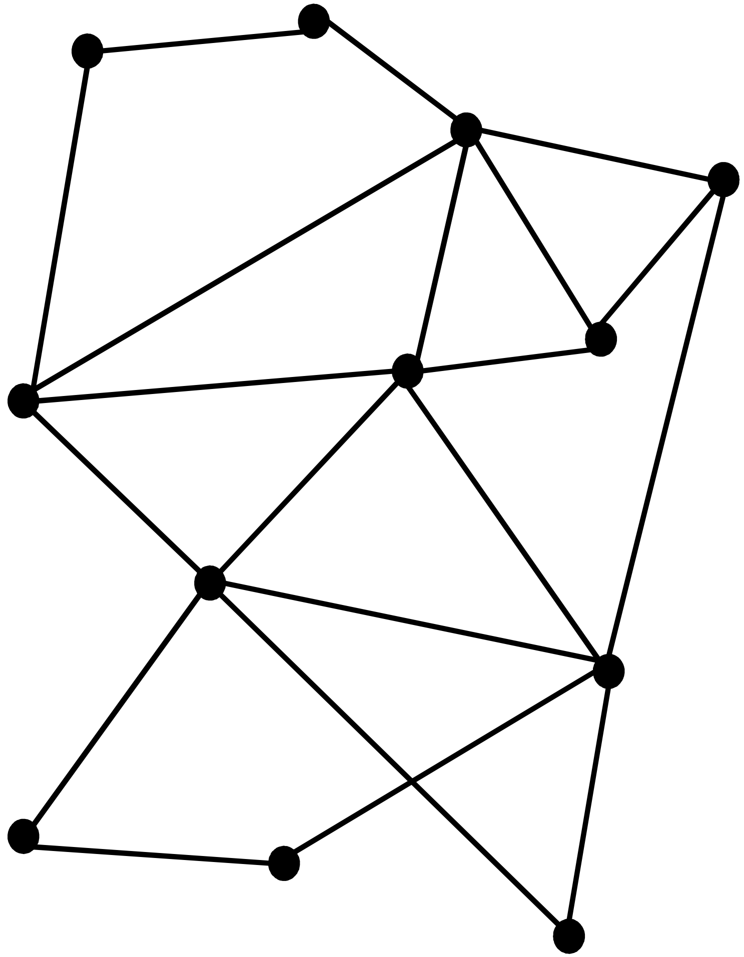
Question: Can we a subset of these integers that sums up to exactly half the sum of all of them?

NP-completeness: Not always beyond reach

VertexCover

Input: G = (V, E) – undirected graph k – an integer

Question: Can we pick k vertices so that all edges are touched by at least one chosen vertex?



Algorithm

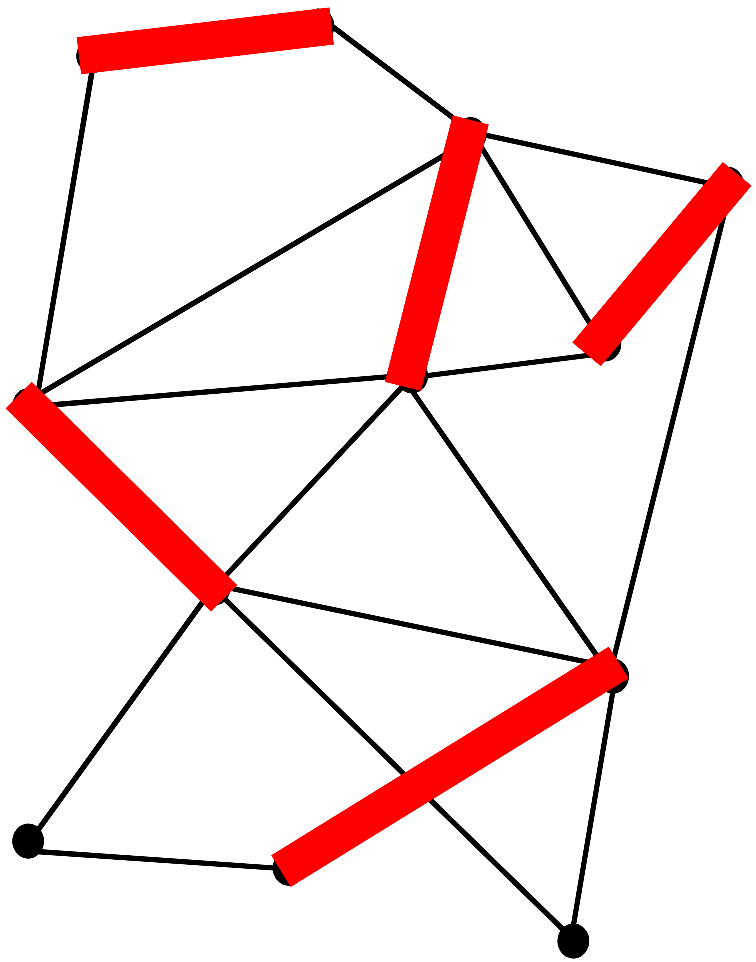
Pick an edge that does not touch any vertices yet chosen. Pick both its endpoints

NP-completeness: Not always beyond reach

VertexCover

Input: G = (V, E) – undirected graph k – an integer

Question: Can we pick k vertices so that all edges are touched by at least one chosen vertex?



Algorithm

Pick an edge that does not touch any vertices yet chosen. Pick both its endpoints

Solution at worst twice as big as the optimal one!

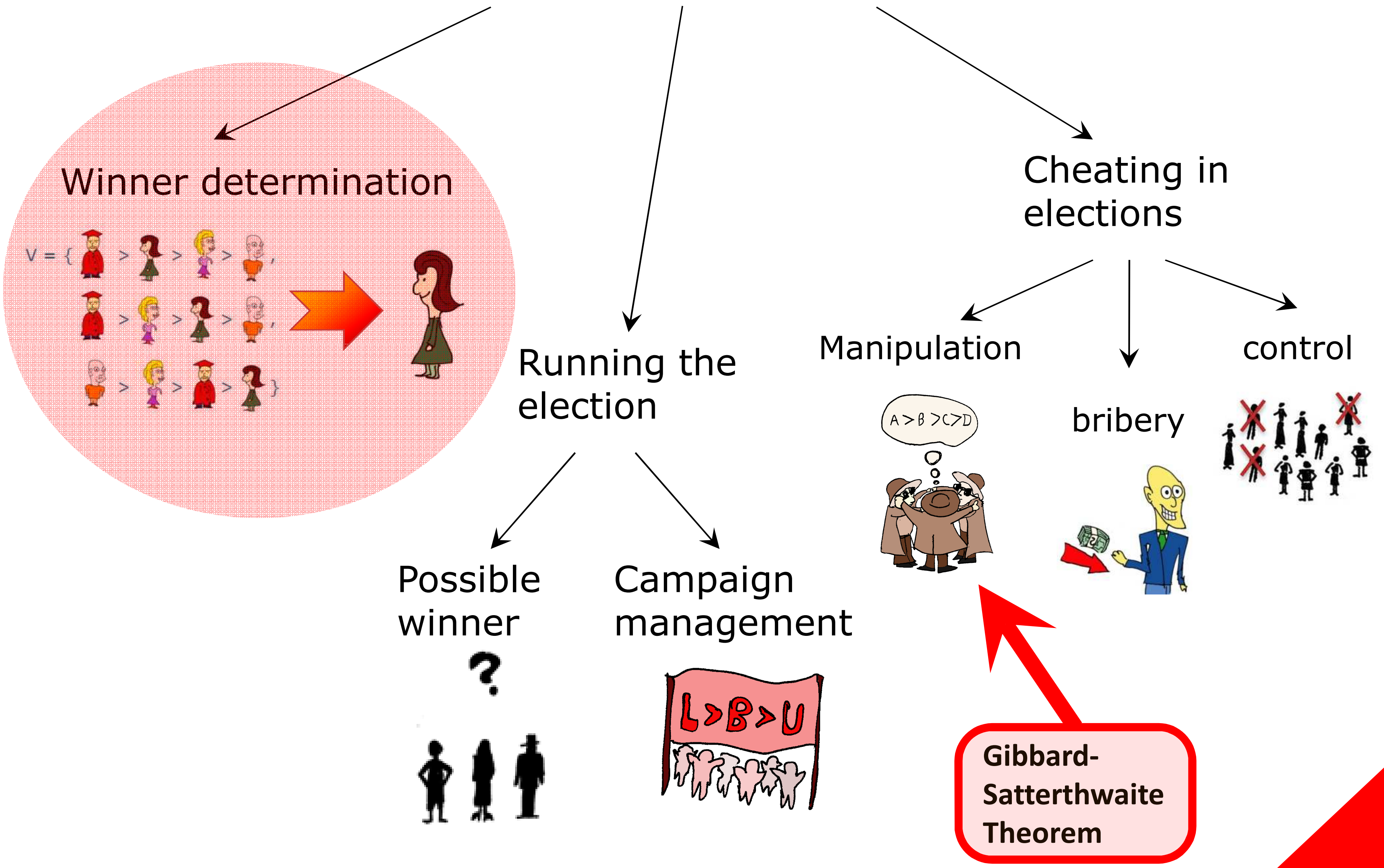
Complexity Theory: Conclusions

- P and NP – the most important complexity classes
  - P – efficient computation
  - NP – efficient verification
- NP-completeness
  - The hardest problems in NP.
  - Solving large instances seems to require millenia…
- Dealing wiht NP-completeness
  - Approximations…
  - .. and many many others

Agenda

- A First Course in Complexity Theory
  - Complexity classes P and NP.
  - NP-completeness
  - Dealing with NP-completeness
- **Complexity is Bad**
  - **Winner determination problems**
    - **Dodgson, Kemeny, Young…**
    - **Monroe, Chamberlin-Courant**
    - **Way around!**
- Complexity is Good
  - The complexity barrier approach
  - Fighting Gibbard-Satterhwaite
  - Fighting other deamons…
  - … and not winning

Computational issues in elections

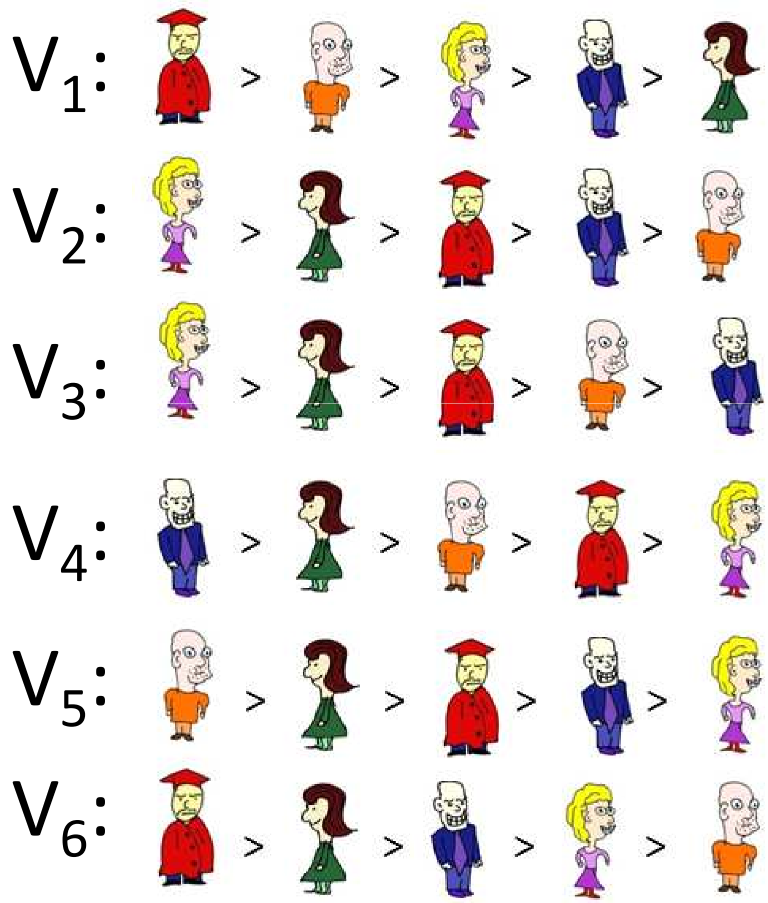


Winner Determination Problem

R-Winner

Input: P=(P_1, …, P_n) – preference profile, c – a candidate from P

Question: Is c an R winner under profile P?



Input size: n voters x m candidates

Typically easy…

- Scoring rules (Plurality, Borda, etc.)
- STV
- Copeland, Maximin, Schuze
- Bucklin
- Approval, and many others …

Winner Determination Can Be Hard!

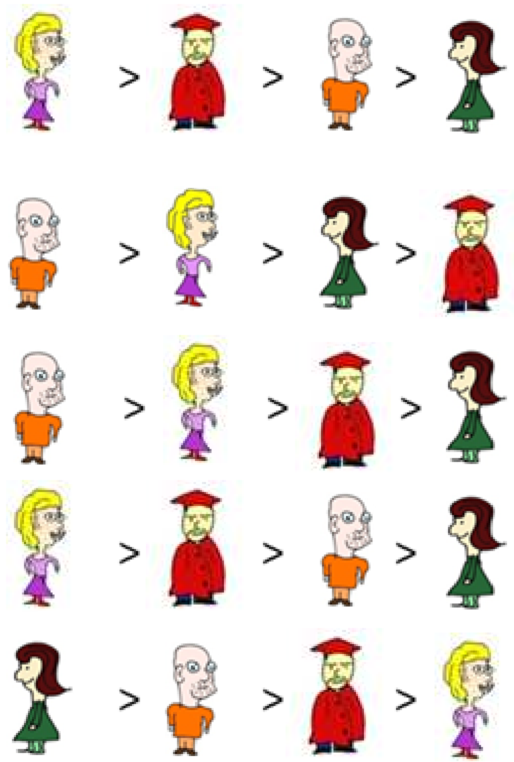
Three interesting voting rules:

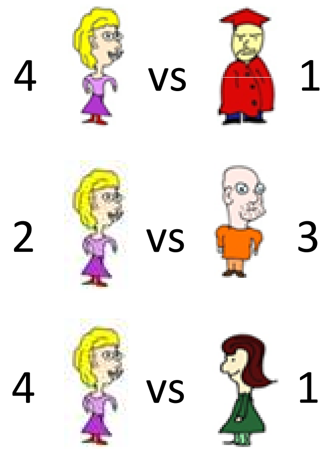
- Dodgson’s
- Kemeny’s
- Young’s

Under each system, we wish to elect someone closest to being a Condorcet winner. Each system defines „closest” in a different way

Dodgson's Rule

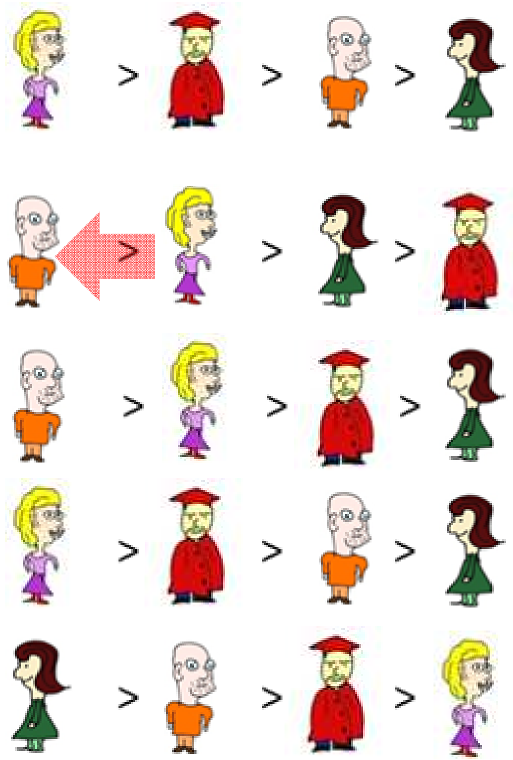
Dodgson's score: Number of swaps of adjacent candidates necessary to ensure that a candidate is a winner

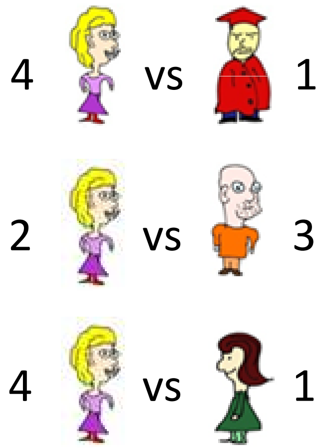




Dodgson's Rule

Dodgson's score: Number of swaps of adjacent candidates necessary to ensure that a candidate is a winner

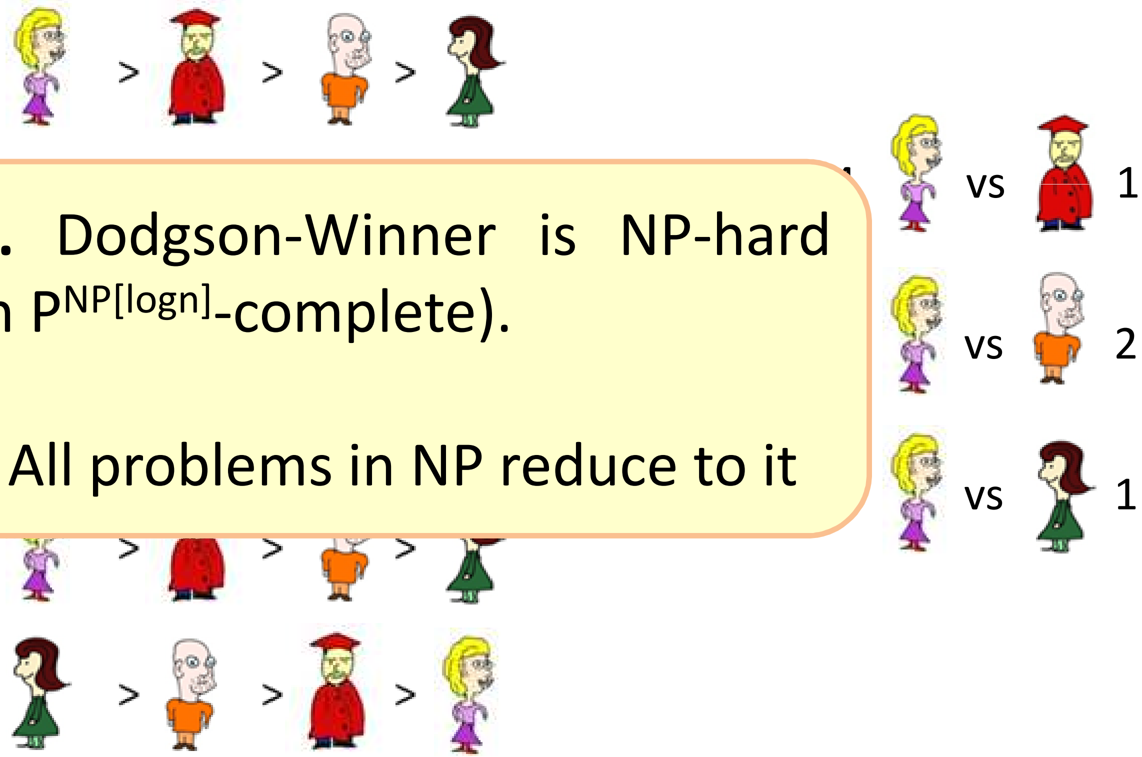




Green lady becomes Condorcet winner after one swap

Dodgson's Rule

Dodgson's score: Number of swaps of adjacent candidates necessary to ensure that a candidate is a winner



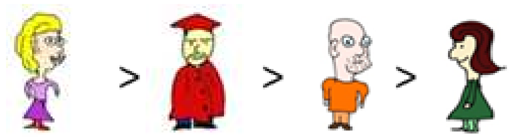
Theorem. Dodgson-Winner is NP-hard (and even $P^{NP[\log n]}$-complete).

NP-hard: All problems in NP reduce to it

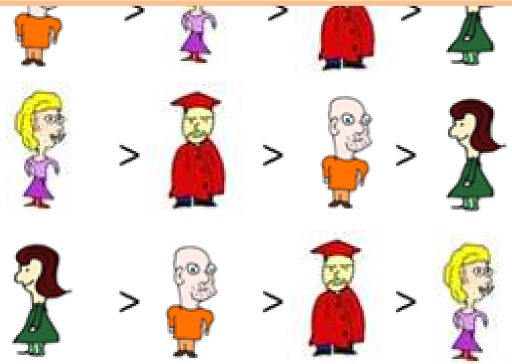
Green lady becomes Condorcet winner after one swap

Kemeny’s Rule

Kemeny’s score of a ranking: The number of inversions between the votes and the ranking.



Theorem. Kemeny-Winner is NP-hard (and even $P^{NP[\log n]}$-complete).



Kemeny-Winner is NP-hard

Other Hard-To-Compute Rules

We will now consider the issue of electing a parliament

Given:

P – preference profile

k – an integer, the size of the parliament

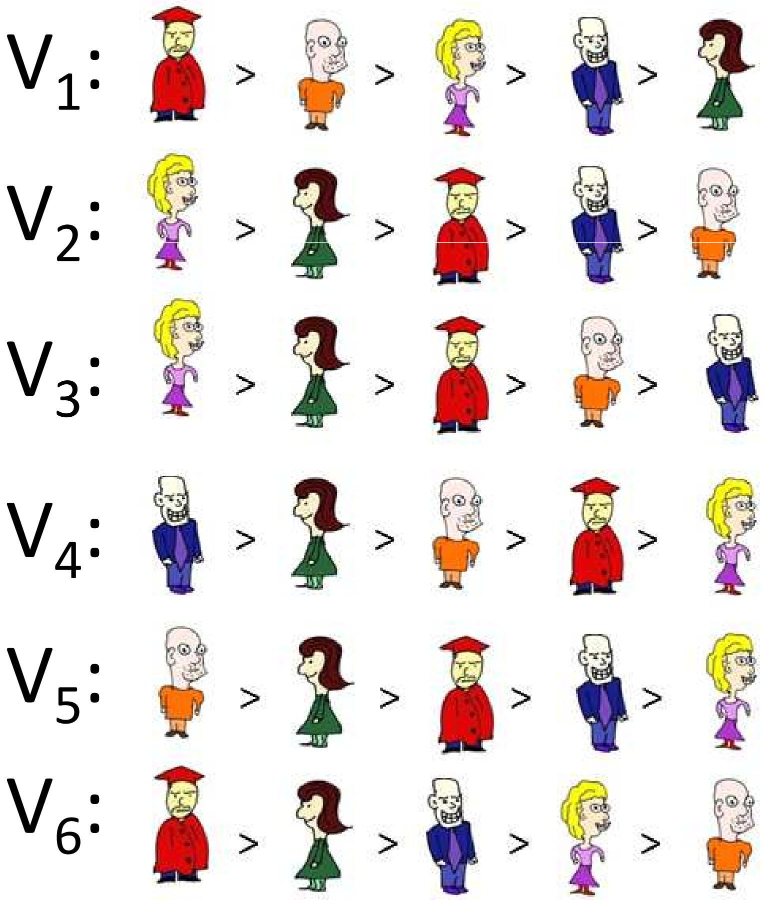
Task:

Pick k candidates that will represent the voters

Many ways of solving the problem…

Monroe and Chambelrin—Courant

Interesting rules to choose parliaments

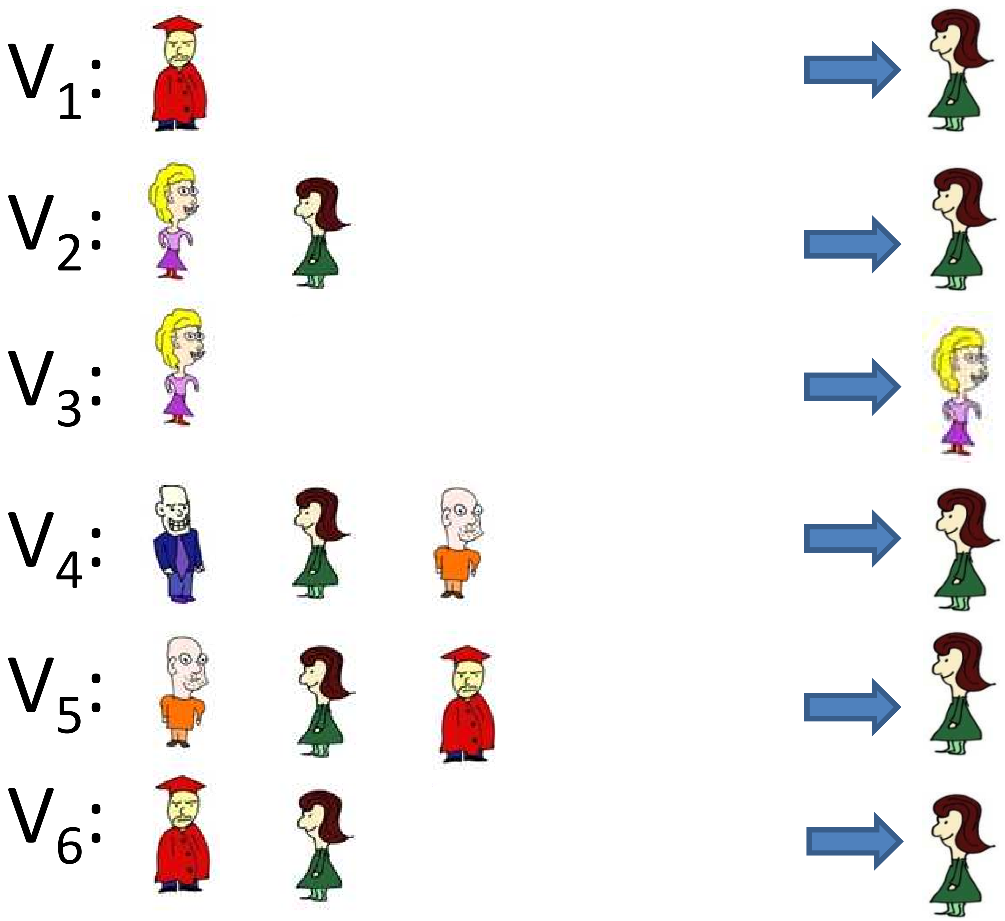


Candidates = Resources

Election system that matches candidates to voters

Monroe oraz Chambelrin—Courant

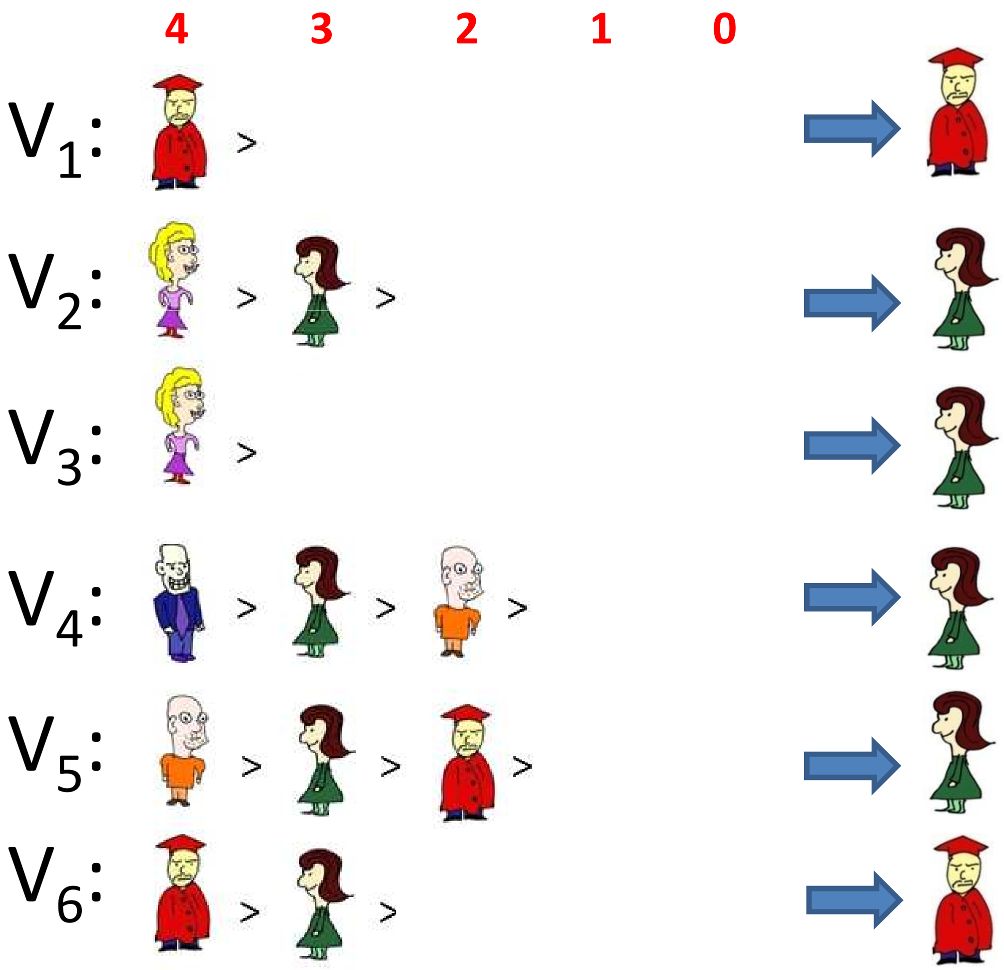
Interesting rules to choose parliaments



Chamberlin-Courant
Pick k candidates and assign them to voters to maximize voter satisfaction

Monroe oraz Chambelrin—Courant

Interesting rules to choose parliaments



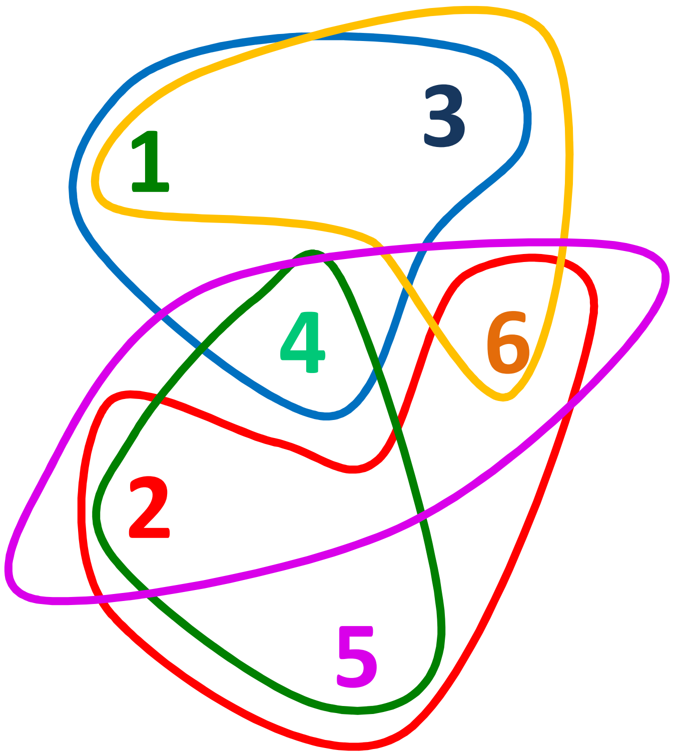
Chamberlin-Courant
Pick k candidates and assign them to voters to maximize voter satisfaction

Monroe and Chamberlin-Courant are NP-Complete

P – polynomial time computation

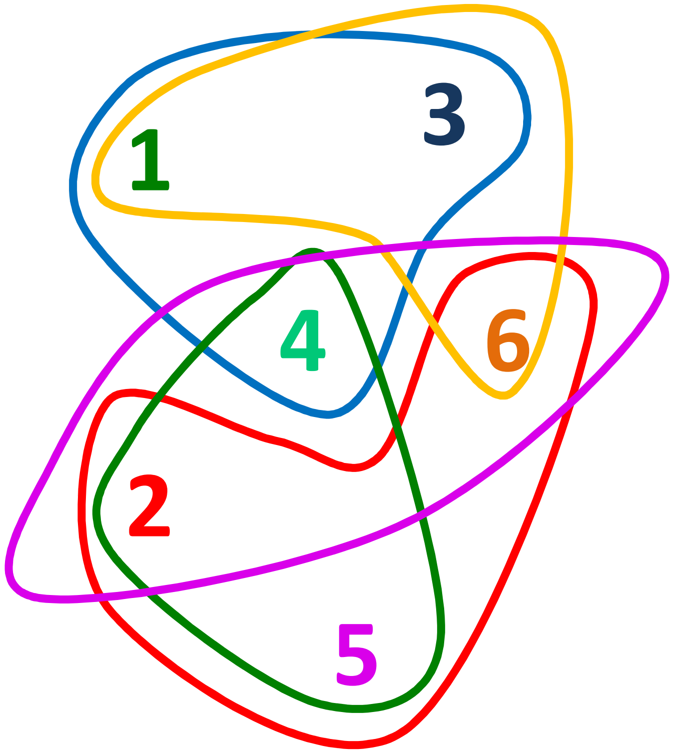
NP – polynomial time verification of solutions

eXact 3-set Cover (X3C)

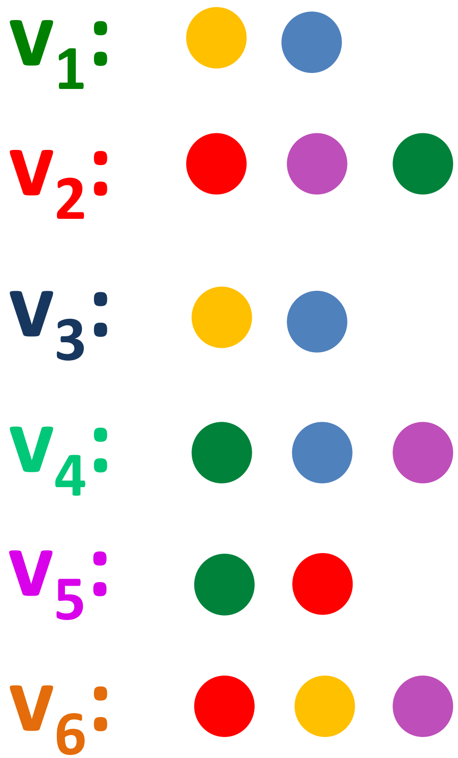


Monroe and Chamberlin-Courant are NP-Complete

eXact 3-set Cover (X3C)



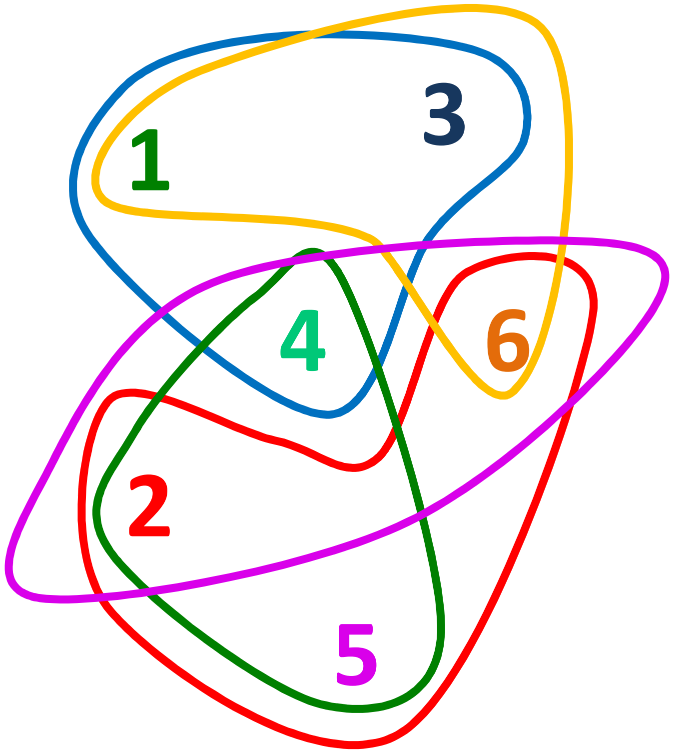
Monroe Winner (Approval)



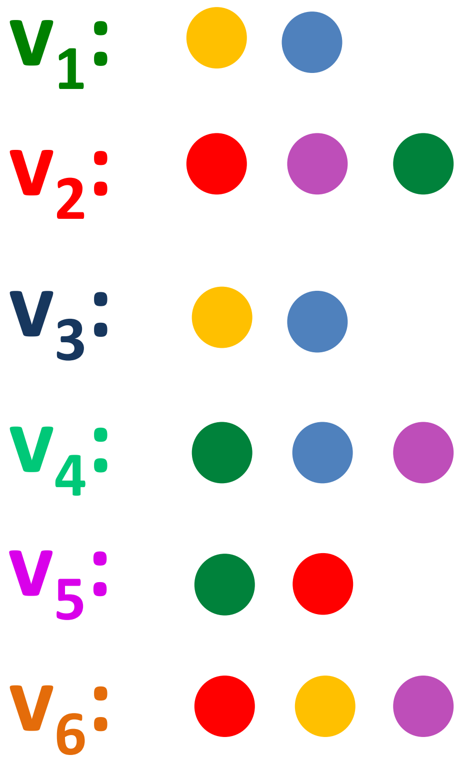
k = 2 (#elements / 3)

Monroe and Chamberlin-Courant are NP-Complete

eXact 3-set Cover (X3C)



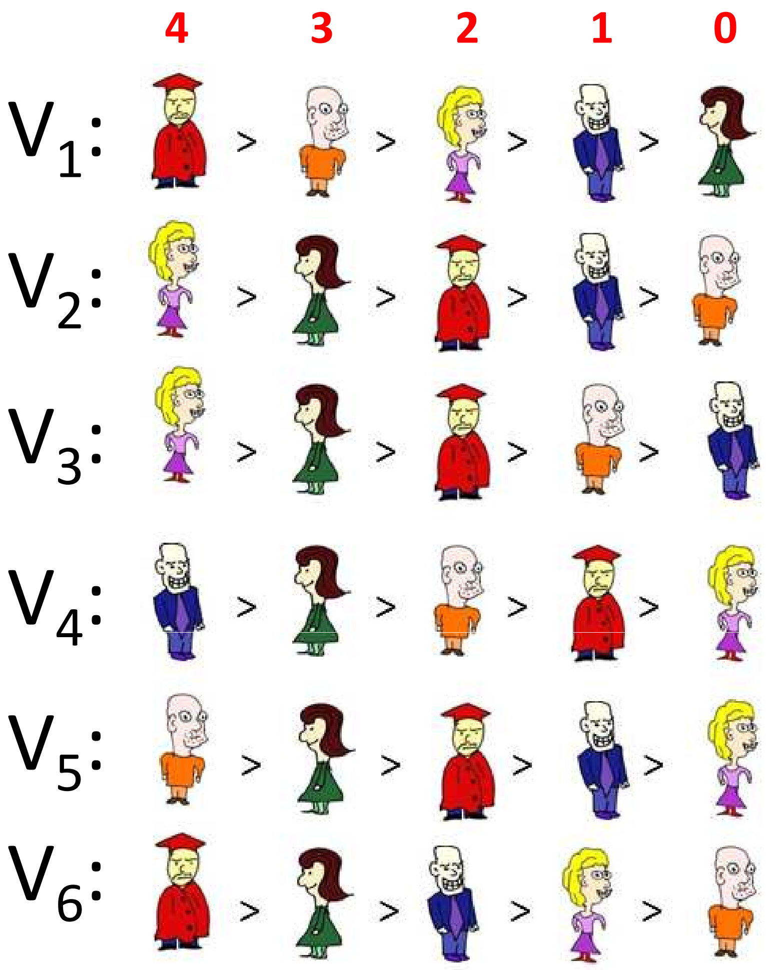
Monroe Winner (Approval)



k = 2 (#elements/ 3)

Approximation!

Goal: Match candidates to voters to maximize satisfaction



Greedy Monroe

Input:

E = (C,V) — election

k — parliament size

Algorithm:

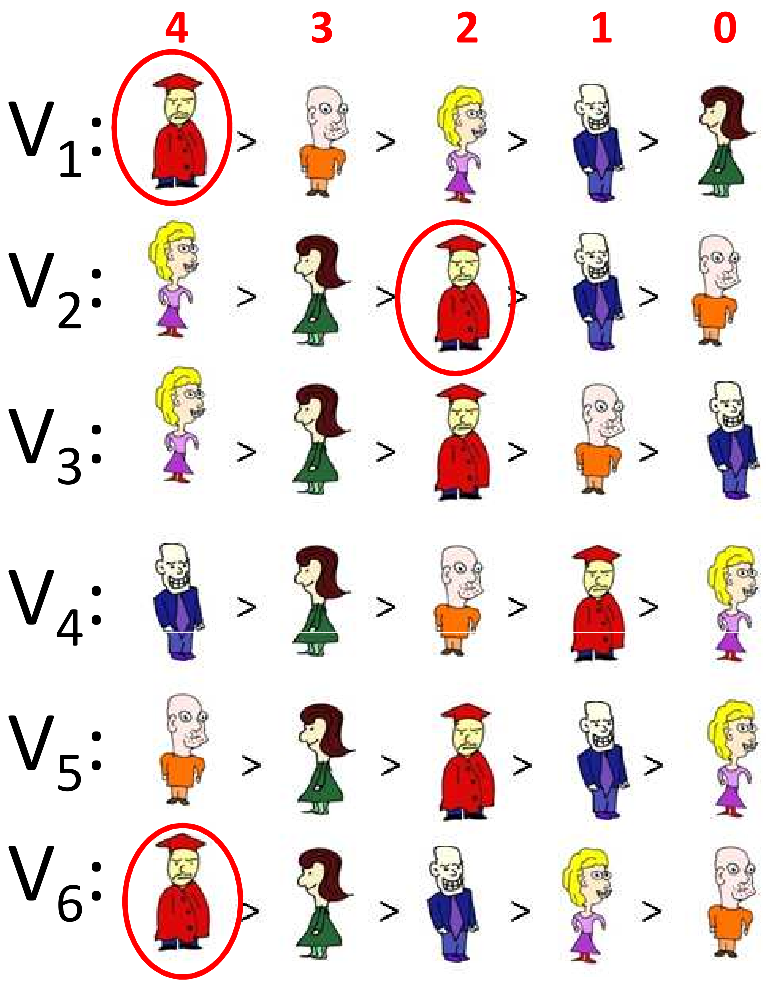
S ← ∅

for *i* = 1 **to** *k* **do**:

for each c **in** C – S:

V(c) ← n/k voters ranking c highest

score(c) ← points of c in V(c)



Greedy Monroe

Input:

E = (C,V) — election

k — parliament size

Algorithm:

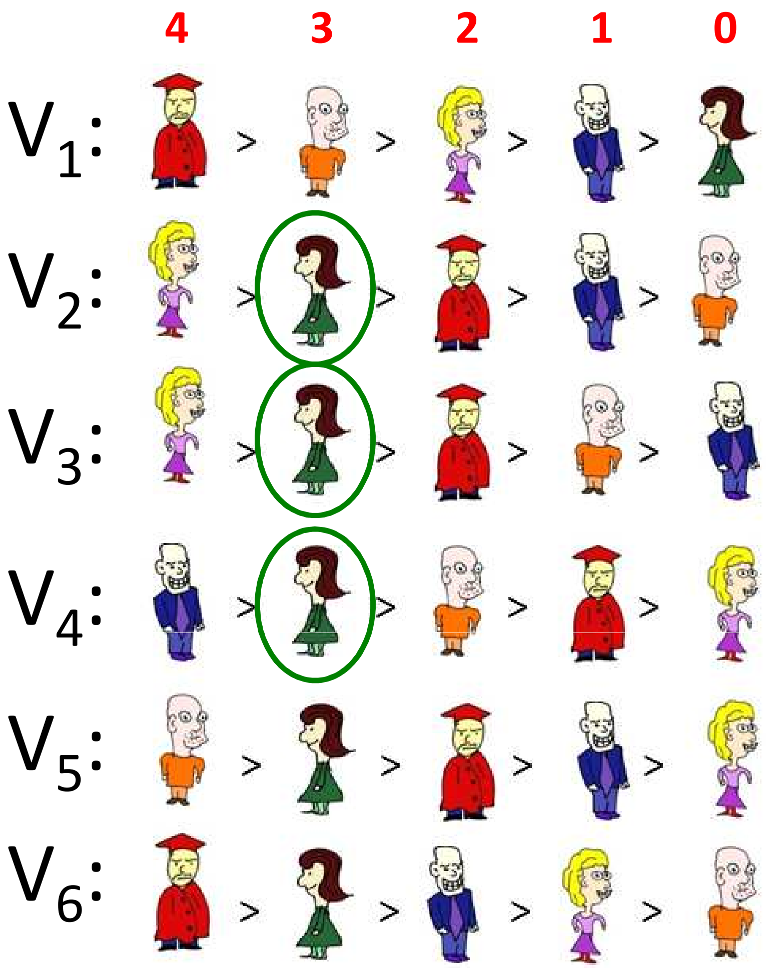
S ← ∅

for *i* = 1 **to** *k* **do**:

  for each c **in** C – S:

    V(c) ← n/k voters ranking c highest

    score(c) ← points of c in V(c)



Greedy Monroe

Input:

E = (C,V) — election

k — parliament size

Algorithm:

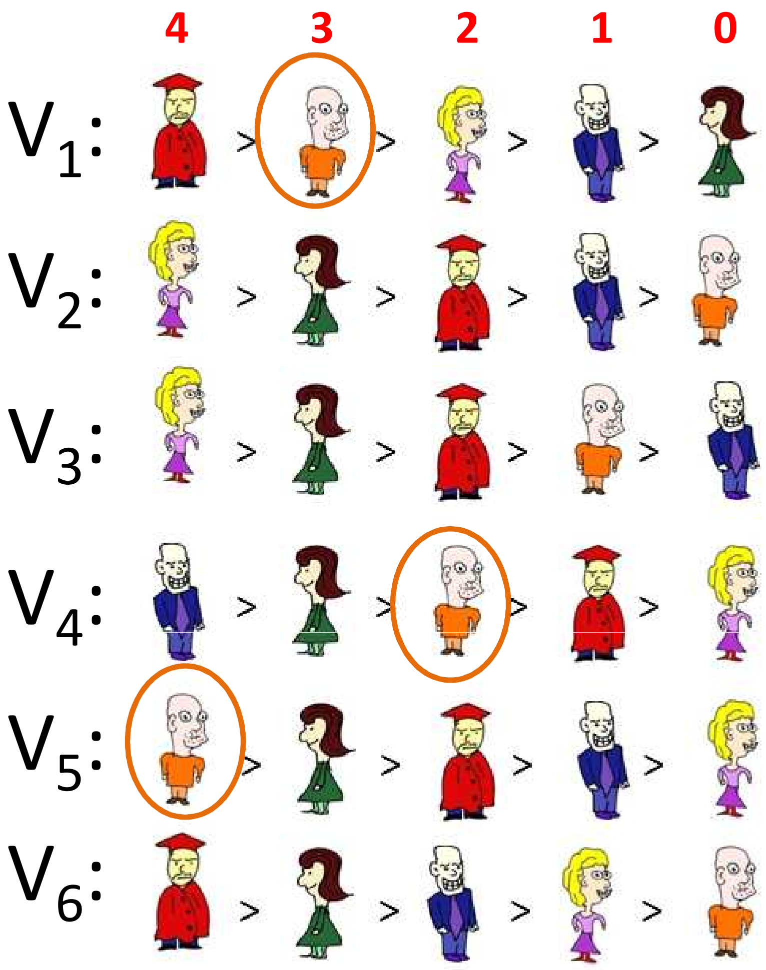
S ← ∅

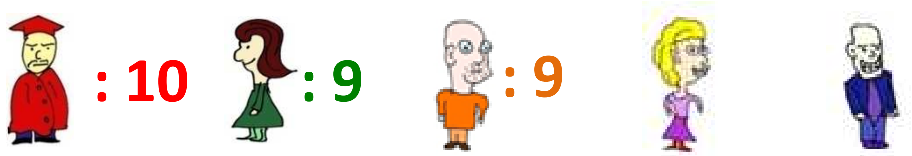
for *i* = 1 **to** *k* **do**:

  for each c **in** C – S:

    V(c) ← n/k voters ranking c highest

    score(c) ← points of c in V(c)





Greedy Monroe

Input:

E = (C,V) — election

k — parliament size

Algorithm:

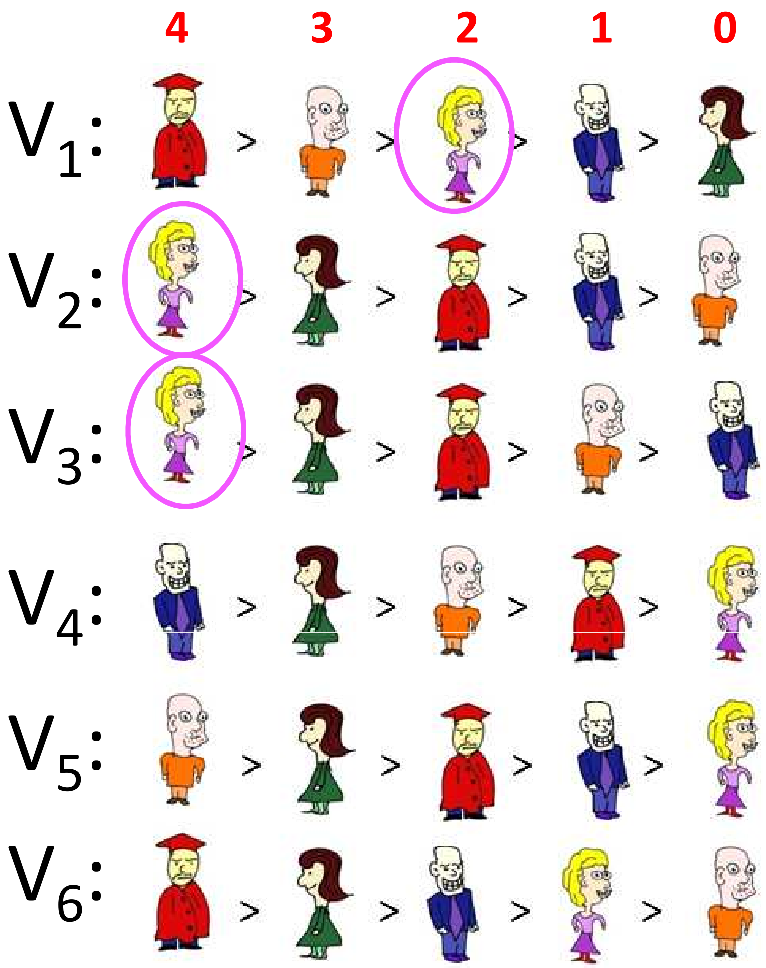
S ← ∅

for *i* = 1 **to** *k* **do**:

for each c **in** C – S:

V(c) ← n/k voters ranking c highest

score(c) ← points of c in V(c)



Greedy Monroe

Input:

E = (C,V) — election

k — parliament size

Algorithm:

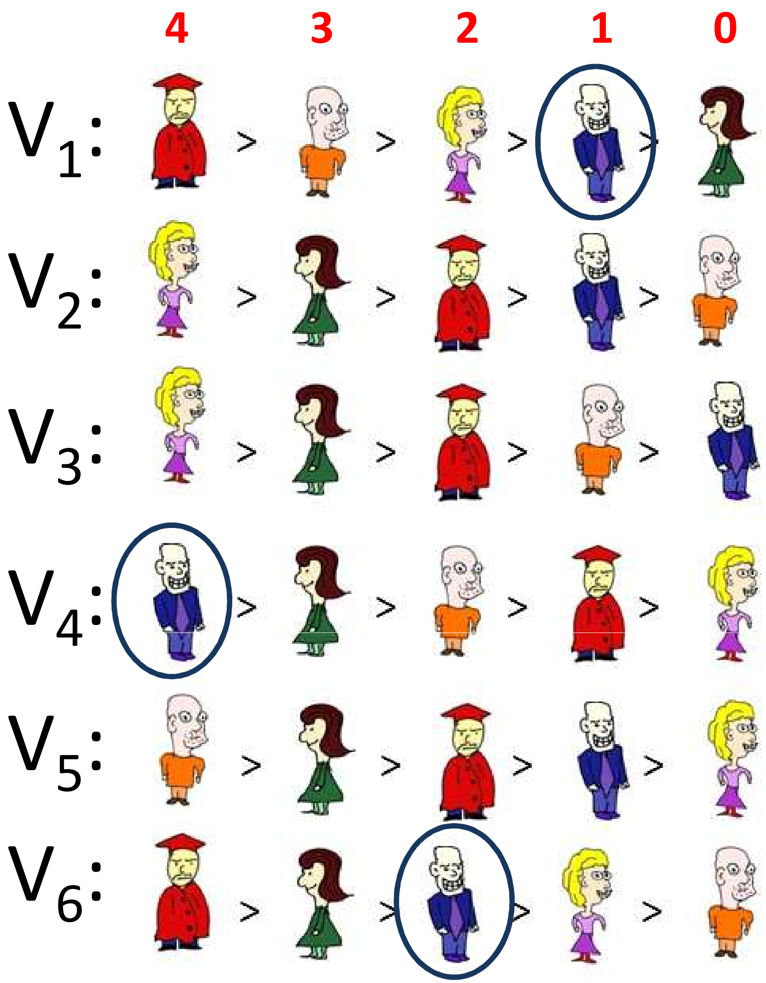
S ← ∅

for *i* = 1 **to** *k* **do**:

  for each c **in** C – S:

    V(c) ← n/k voters ranking c highest

    score(c) ← points of c in V(c)



Greedy Monroe

Input:

E = (C,V) — election

k — parliament size

Algorithm:

S ← ∅

for *i* = 1 **to** *k* **do**:

for each c **in** C – S:

V(c) ← n/k voters ranking c highest

score(c) ← points of c in V(c)

c* ← $\text{argmax}_{c \in C}$ (score(c))

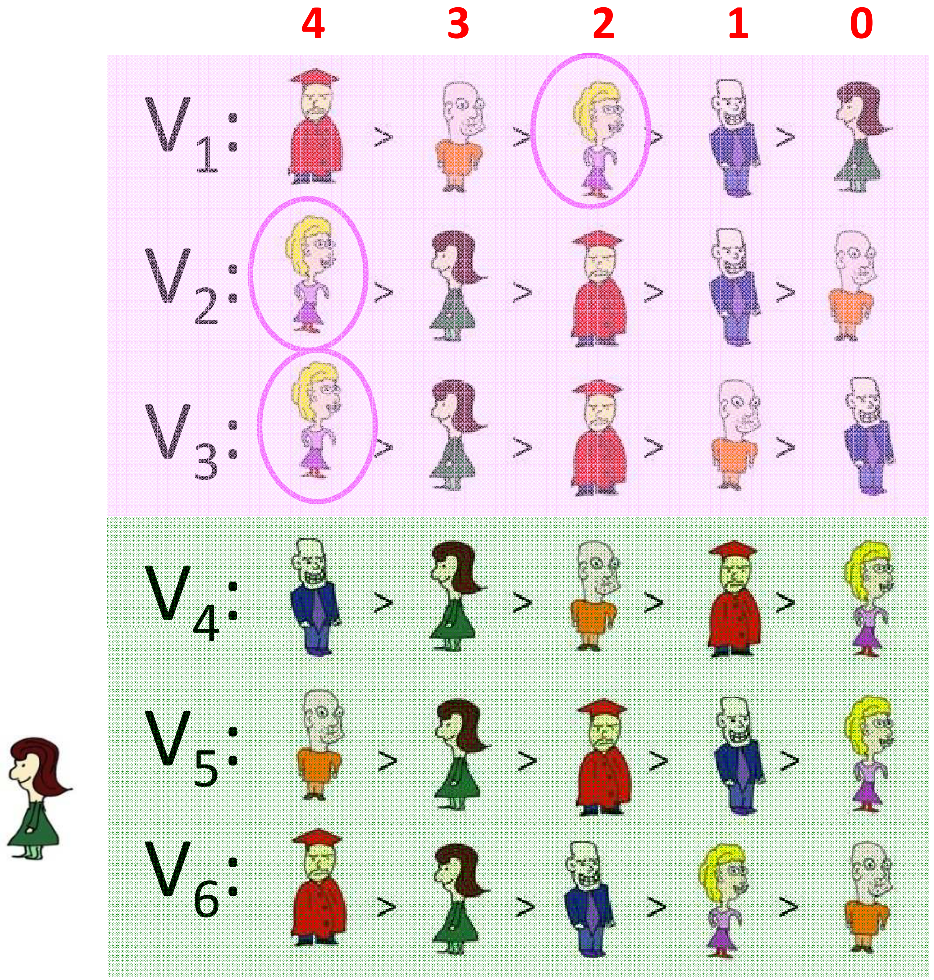
S ← S ∪ {c*}

V ← V – V(c*)

C ← C – {c*}

assign c* to voters from V(c*)

return computed assignment



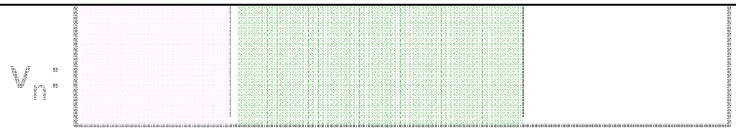
How Good is Greedy Monroe?

Consider the situation after the i-th iteration

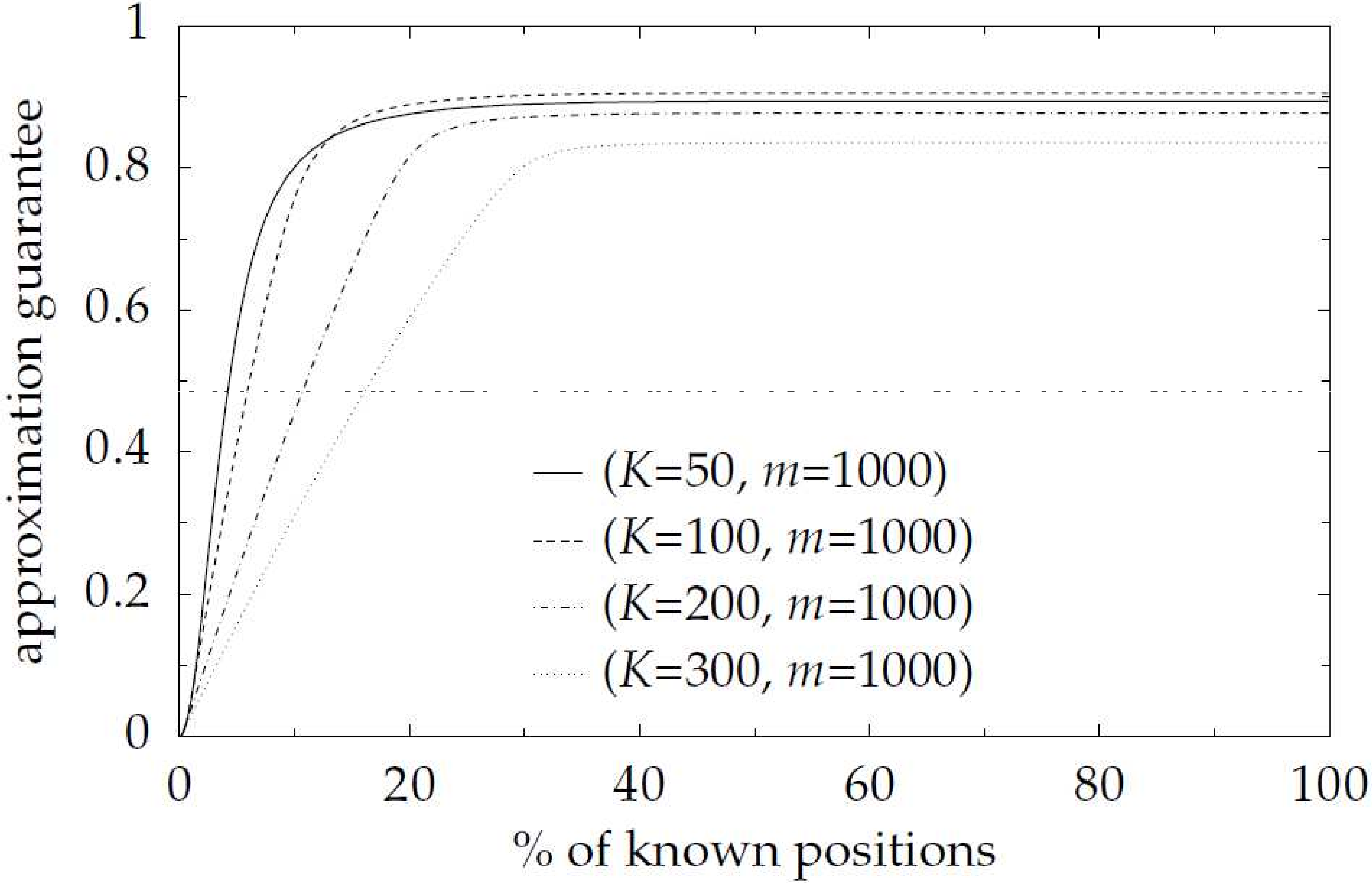
By the pigeonhole principle, there are at

i potentially

$$
\begin{aligned}
\sum_{i=0}^{K-1}\frac{n}{K}\cdot\left(m-i-\left\lceil\frac{m-i}{K-i}\right\rceil\right) &\geq \sum_{i=0}^{K-1}\frac{n}{K}\cdot\left(m-i-\frac{m-i}{K-i}-1\right)\\
&= \sum_{i=1}^{K}\frac{n}{K}\cdot\left(m-i-\frac{m-1}{K-i+1}+\frac{i-2}{K-i+1}\right)\\
&= \frac{n}{K}\left(\frac{K(2m-K-1)}{2}-(m-1)H_K+K(H_K-1)-H_K\right)\\
&= (m-1)n\left(1-\frac{K-1}{2(m-1)}-\frac{H_K}{K}+\frac{H_K-1}{m-1}-\frac{H_K}{K(m-1)}\right)\\
&> (m-1)n\left(1-\frac{K-1}{2(m-1)}-\frac{H_K}{K}\right)
\end{aligned}
$$



How Good is Greedy Monroe?



Winner Determination: Conclusions

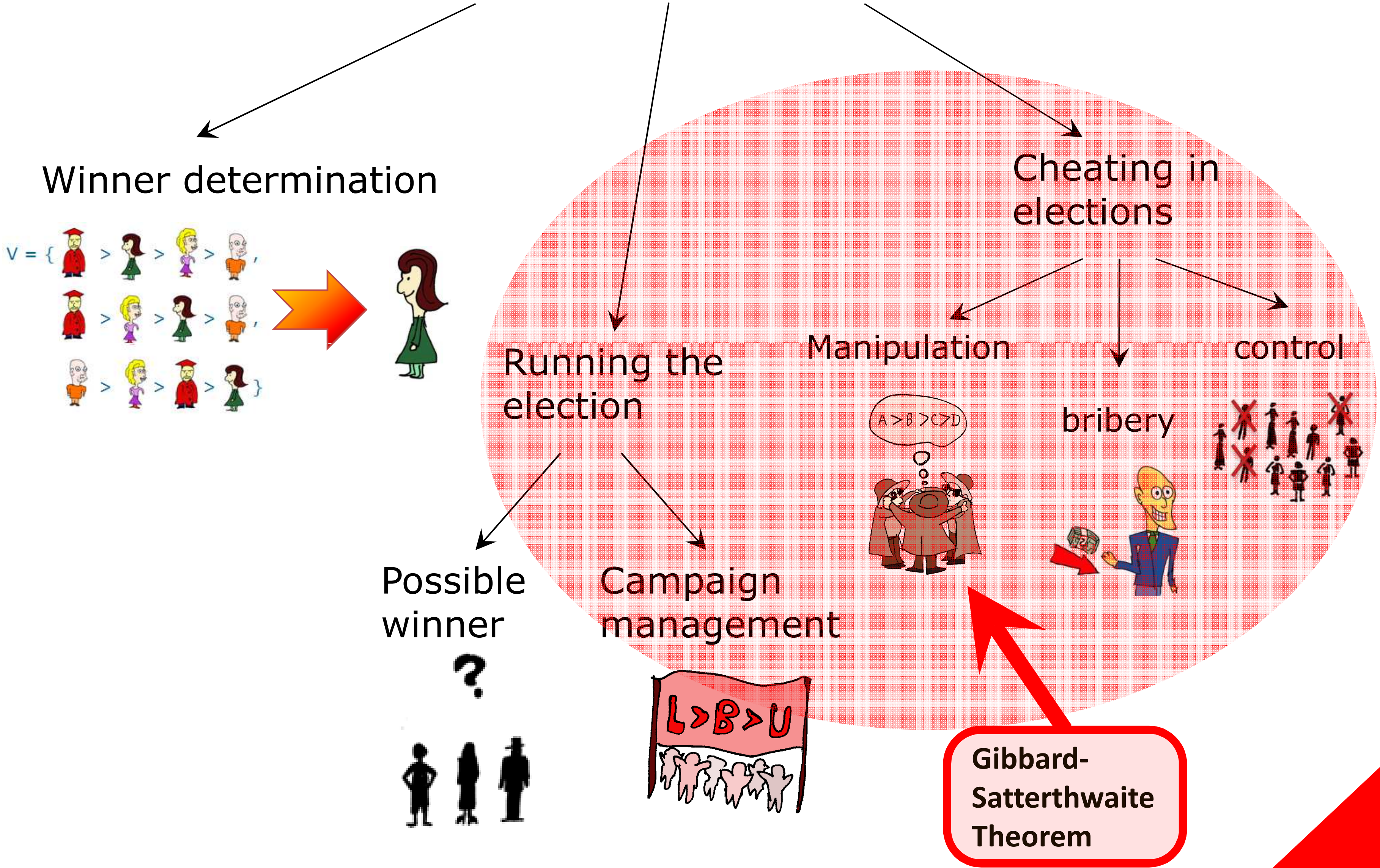
- Most voting rules have efficient winner determination procedures
  - Scoring rules, STV, Bucklin, …
  - Copeland, Maximin, Schulze
- But for some it is computationally hard
  - Dodgson, Kemeny, Young
  - Monroe, Chamberlin-Courant

  ... But almost always there is a workaround (almost)

Agenda

- A First Course in Complexity Theory
  - Complexity classes P and NP.
  - NP-completeness
  - Dealing with NP-completeness
- Complexity is Bad
  - Winner determination problems
    - Dodgson, Kemeny, Young…
    - Monroe, Chamberlin-Courant
    - Way around!
- **Complexity is Good**
  - **The complexity barrier approach**
  - **Fighting Gibbard-Satterhwaite**
  - **Fighting other deamons…**
  - **… and not winning**

Computational issues in elections



Complexity Barrier Approach

Model: Represent each cheating strategy as a computational decision problem.

Complexity barrier approach: If manipulating elections is hard, then we can ignore the fact that it is in principle possible.

Approach initiated by Bartholdi, Tovey, and Trick in the late 80s and the early 90s

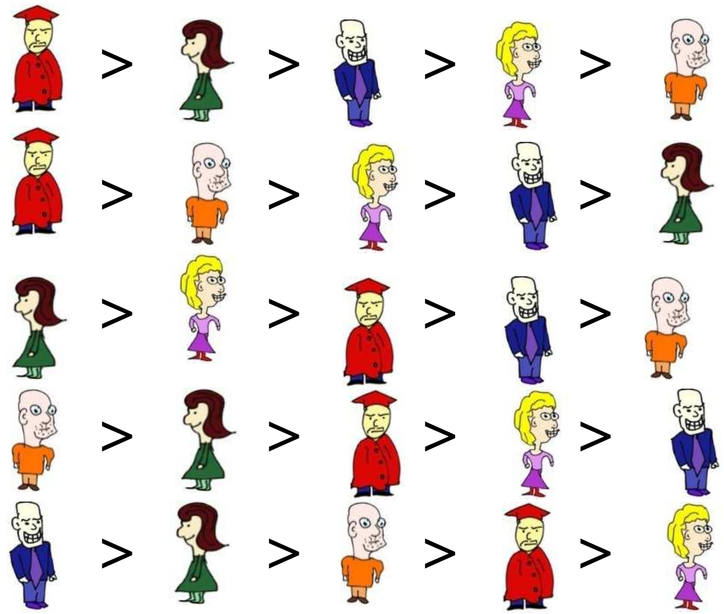
Complexity Barrier: Results

- Effects of complexity barrier research
  - Dozens of computational problems identified
  - Multiple standard election systems analyzed
  - Quite thorough understanding of **worst case** complexity of elections

- Complications…
  - We would like **some** of the problems to be efficiently computable
    - Determining winners
    - Organizing a campaign
  - Worst-case analysis seems problematic…

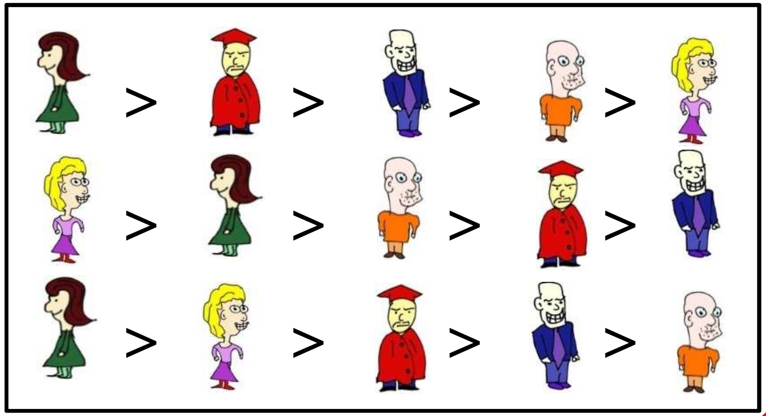
Control under Plurality

Control by adding voters
Given:
E = (C, V) – an election
W – additional voters
p in C – preferred candidate
k – budget
Question:
Is it possible to ensure p's victory by adding at most k voters



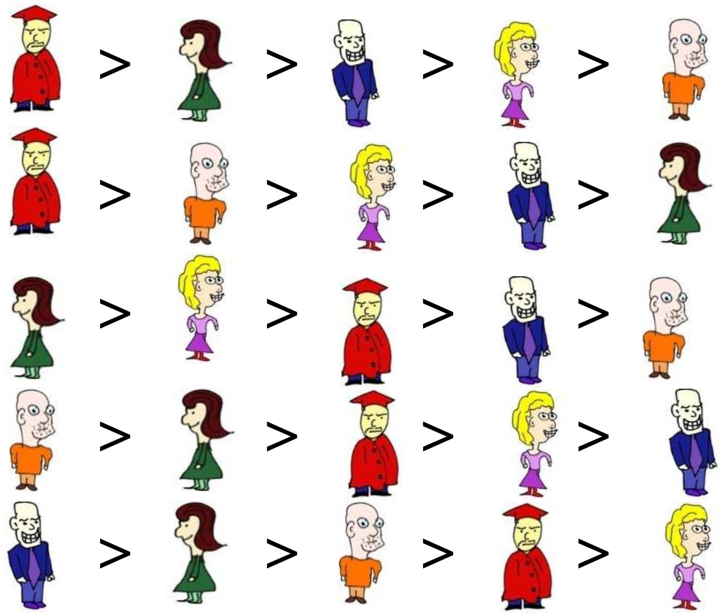
p = 

k = 2



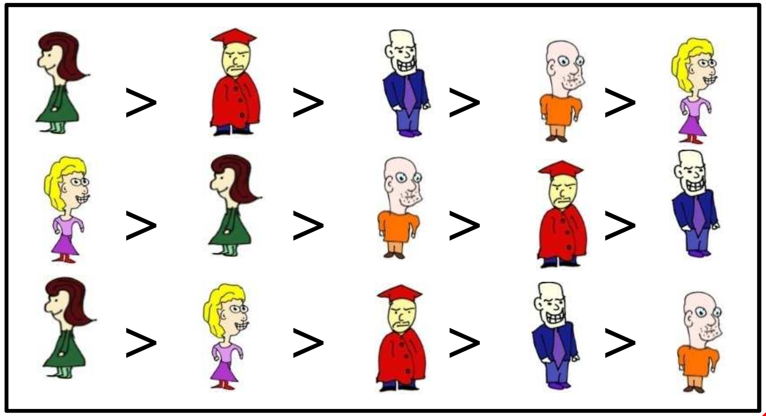
Control under Plurality

Control by adding voters
Given:
E = (C, V) – an election
W – additional voters
p in C – preferred candidate
k – budget
Question:
Is it possible to ensure p’s victory by adding at most k voters



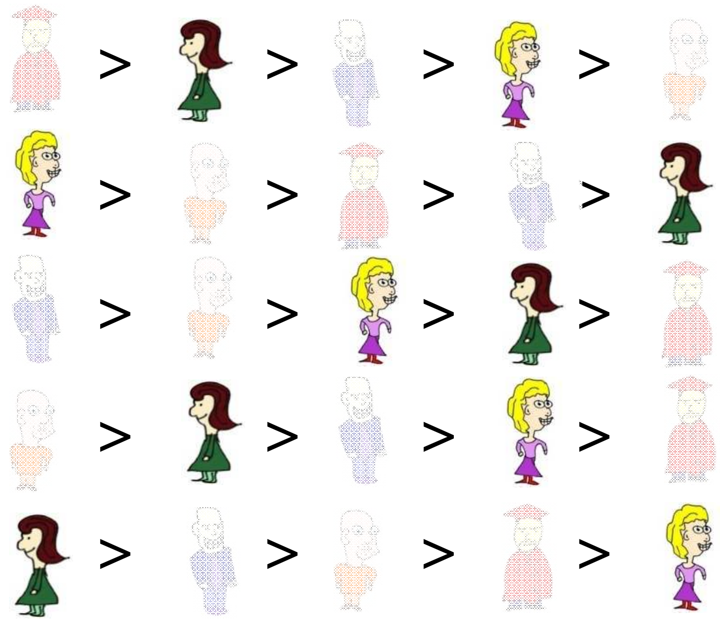
p = 

k = 2



Control under Plurality

Control by adding candidates
Given:
E = (C, V) – an election
A – additional candidates
p in C – preferred candidate
k – budget
Question:
Is it possible to ensure p's victory by adding at most k candidates

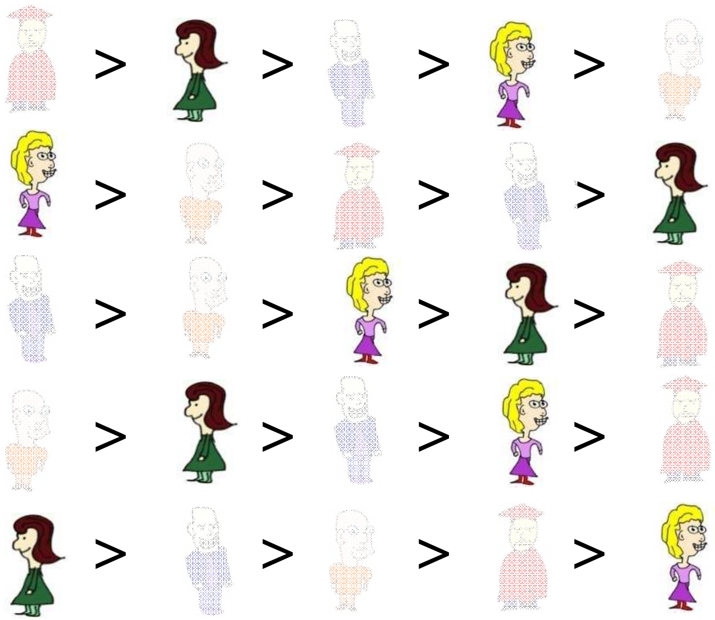


p = 

k = 2

Control under Plurality

Control by adding candidates
Given:
E = (C, V) – an election
A – additional candidates
p in C – preferred candidate
k – budget
Question:
Is it possible to ensure p's victory by adding at most k candidates



p = 

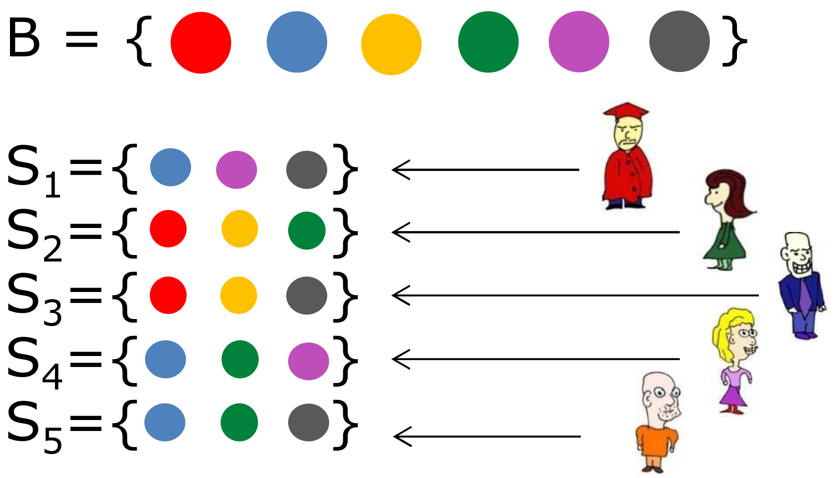
k = 2

Control by Adding Candidates ∈ NP-com

Proof: Reduction from the X3C problem

Exact Cover by 3-Sets

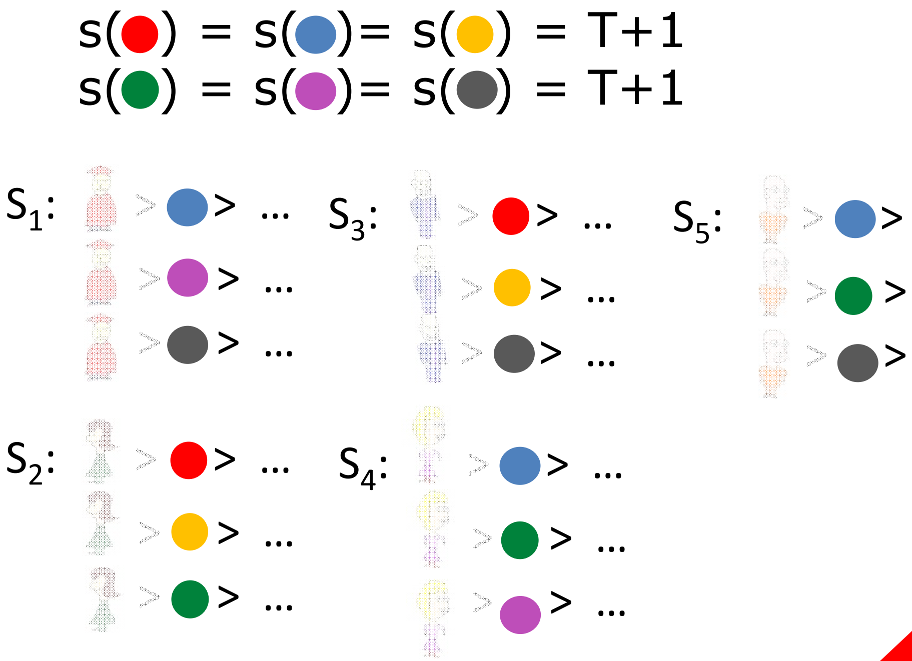
Input: $B = \{b_1, b_2, b_3, \dots, b_{3k}\}$
$S = \{S_1, \dots, S_n\}$



Question: Is it possible to pick k sets and cover all elements from B?

Control by Adding Candidates

$s(p) = T$



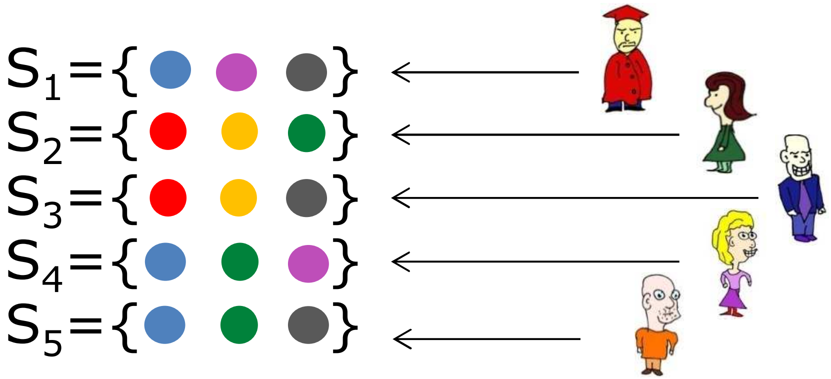
Control by Adding Candidates ∈ NP-com

Proof: Reduction from the X3C problem

Exact Cover by 3-Sets

Input: $B = \{b_1, b_2, b_3, \ldots\ , b_{3k}\}$
$S = \{S_1, \ldots, S_n\}$

B = { ● ● ● ● ● ● }

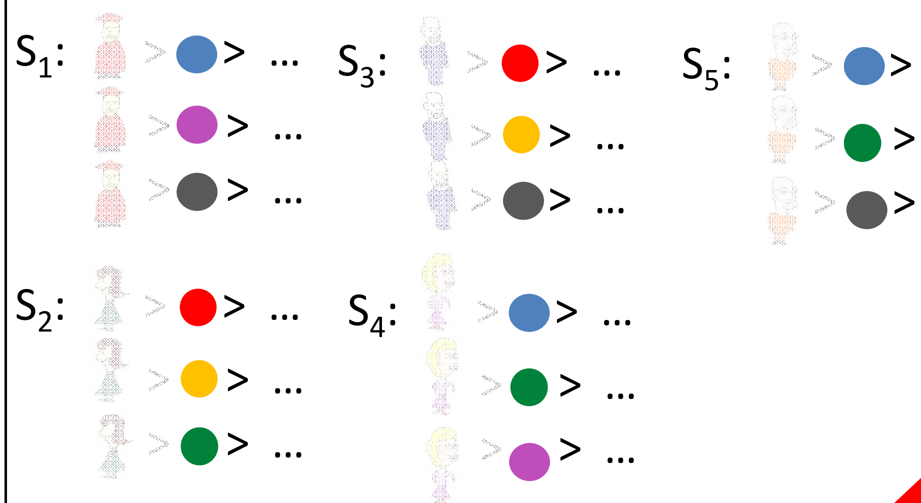


Question: Is it possible to pick k sets and cover all elements from B?

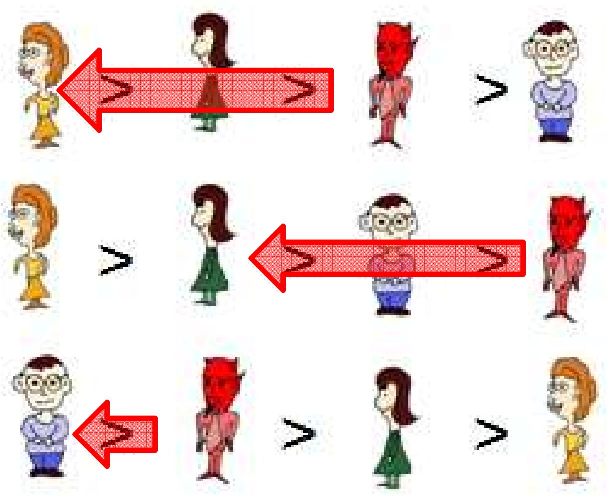
Control by Adding Candidates

$s(p) = T$

s(●) = s(●)= s(●) = T
s(●) = s(●)= s(●) = T



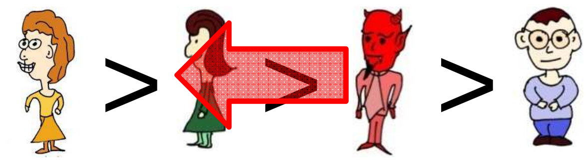
Shift Bribery



- Allowed swaps:
  - Have to involve our candidate
- Realistic?
  - As bribery: **Yes**
  - Also: as a **campaigning model!**
- Gain in complexity?

The Algorithm

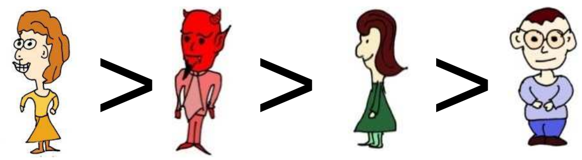
Why 2-approximation?



a_{i+1} a_i

The Algorithm

Why 2-approximation?



a_{i+1} a_i

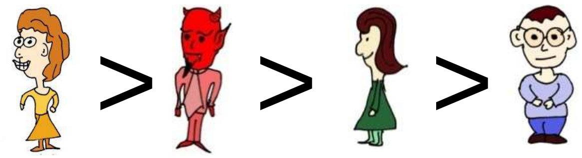
 gains $a_{i+1} - a_i$ points

 loses $a_{i+1} - a_i$ points

Getting **2x** the points for 
than the best bribery gives
is sufficient to win

The Algorithm

Why 2-approximation?



a_{i+1} a_i

 gains $a_{i+1} - a_i$ points

 loses $a_{i+1} - a_i$ points

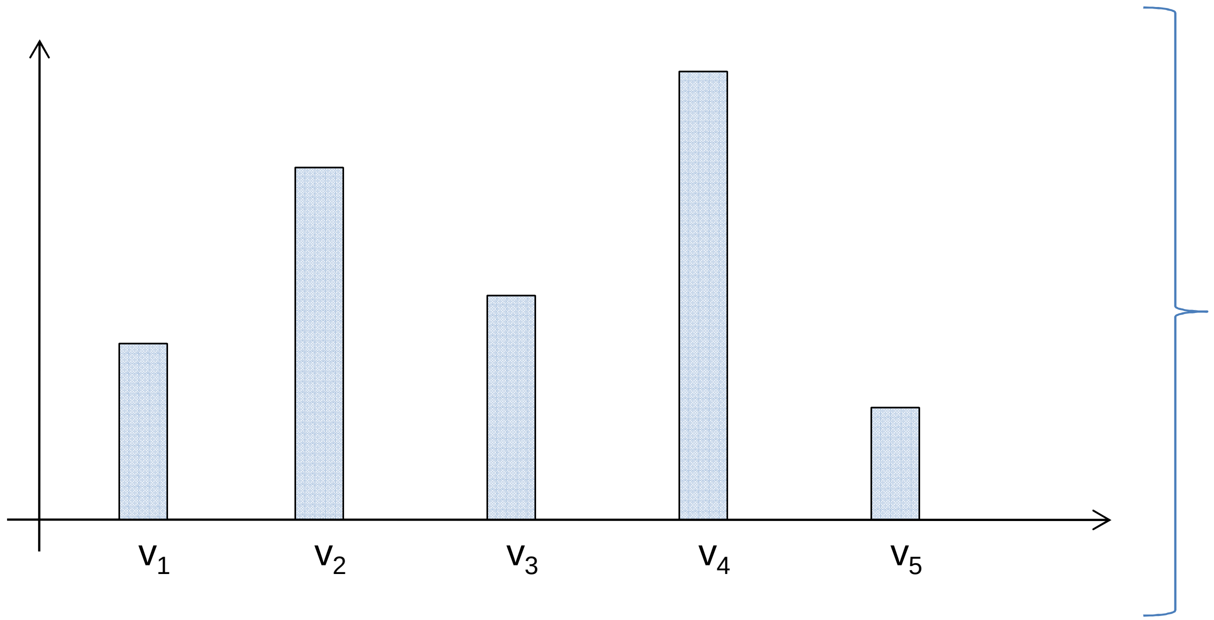
Getting **2x** the points for  than the best bribery gives is sufficient to win

Operation of the algorithm

1. Guess a cost k
2. Get most points for  t cost k
3. Guess a cost k' <= k
4. Get most points for  : cost k'

This is a 2-approximation... but works in polynomial time only if **prices are encoded in unary**

Why Does the Algorithm Work?



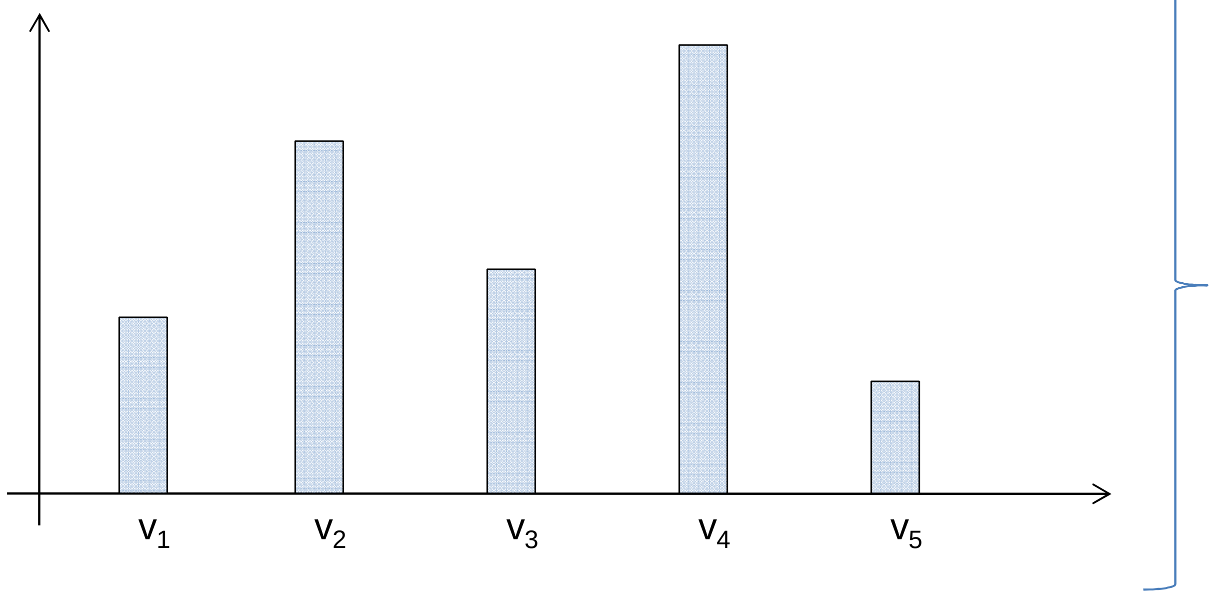
How much does optimal solution shift candidate **p** in each vote?

O – the optimal solution → gives **p** some T points

Operation of the algorithm

1. Guess a cost k
2. Get most points for **p** at cost k
3. Guess a cost k' <= k
4. Get most points for **p** at cost k'

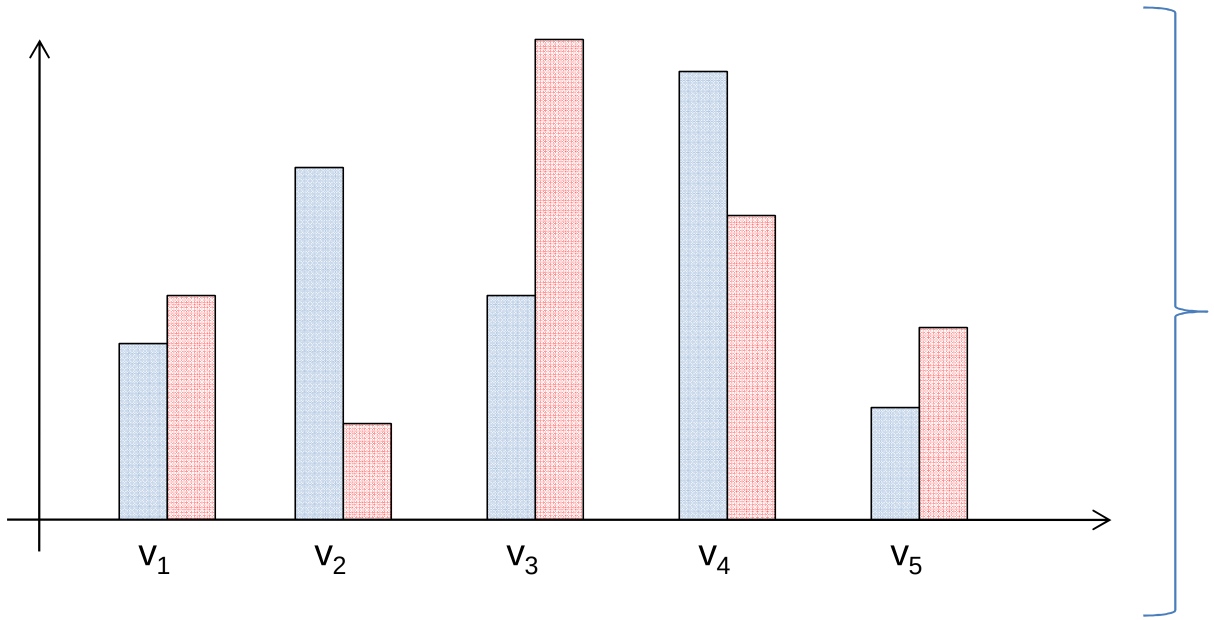
Why Does the Algorithm Work?



How much does optimal solution shift candidate **p** in each vote?

O – the optimal solution → gives **p** some T points

Why Does the Algorithm Work?

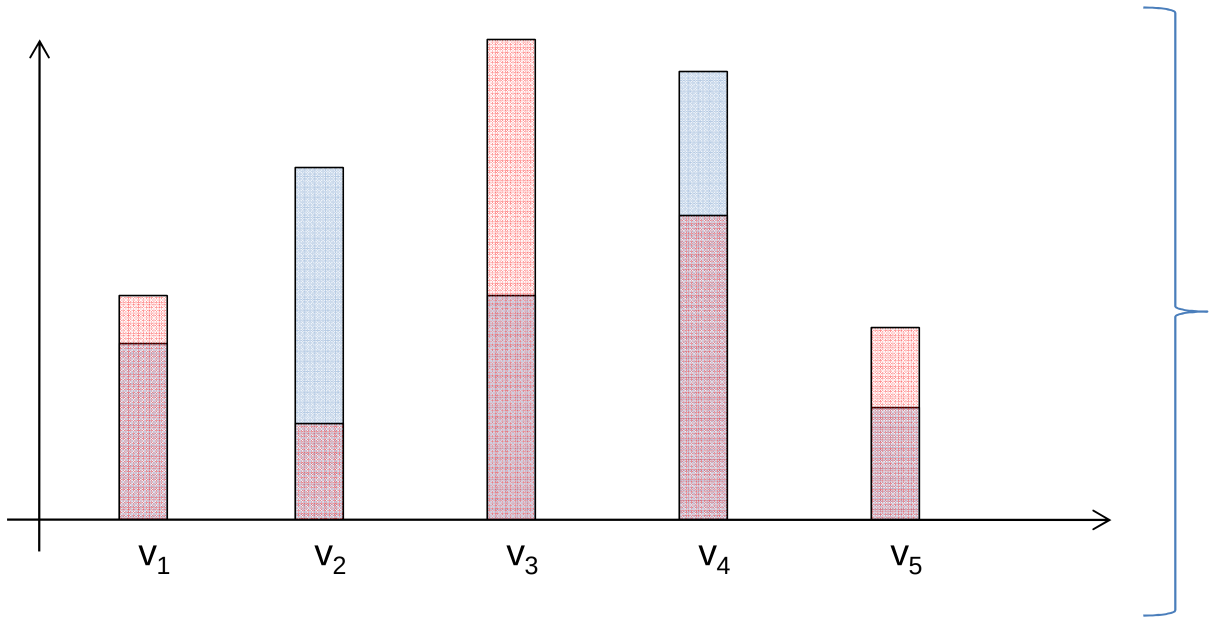


How much does optimal solution shift candidate **p** in each vote?

O – the optimal solution → gives **p** some T points

S – solution that gives most points at cost k

Why Does the Algorithm Work?



How much does optimal solution shift candidate **p** in each vote?

O – the optimal solution → gives **p** some T points

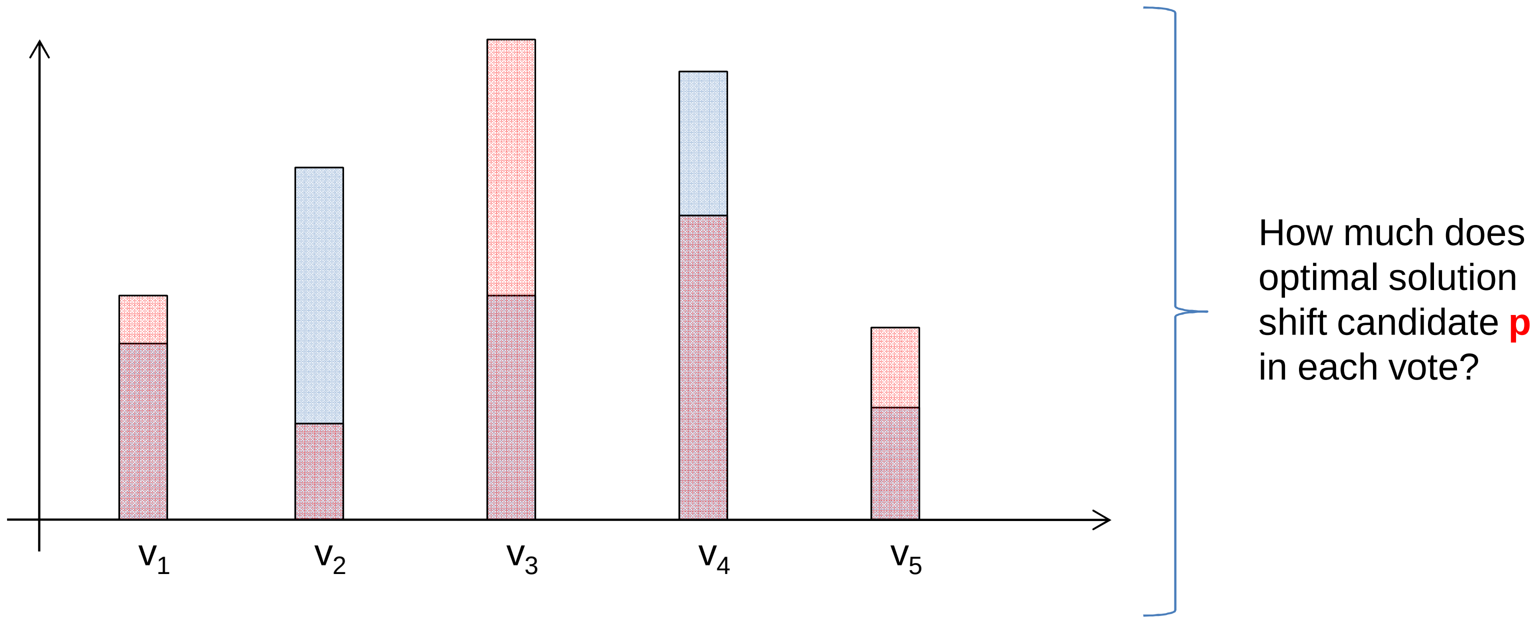
S – solution that gives most points at cost k

min(O,S) – min shift of the two in each vote gives some D points to **p**

Now it is possible to complete min(O,S) in two independent ways:

1. By continuing as S does (getting at least T-D points extra)
2. By continuing as O does (getting T-D points extra)

Why Does the Algorithm Work?



Now it is possible to complete min(O,S) in two independent ways:

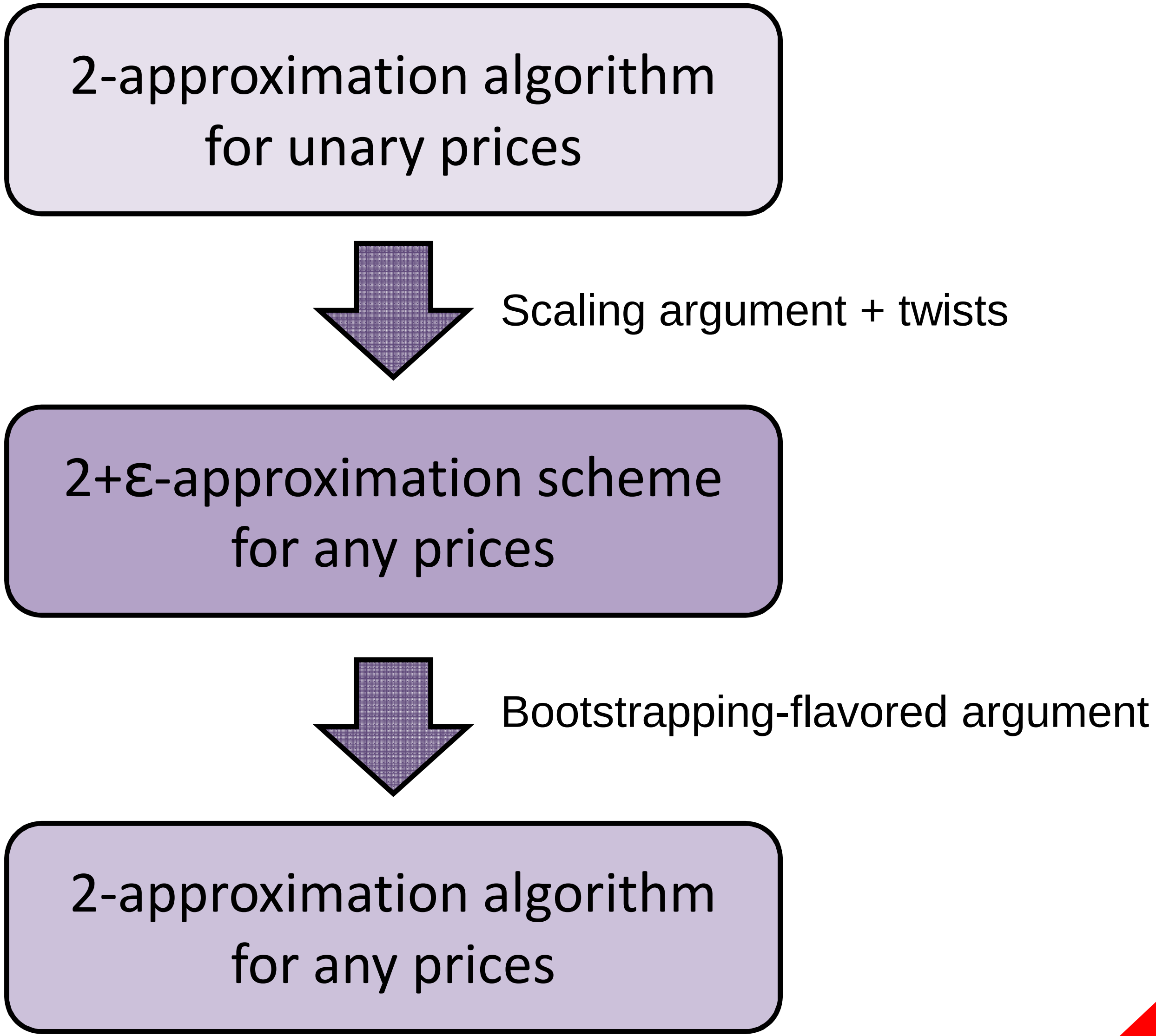
1. By continuing as S does (getting at least T-D points extra)
2. By continuing as O does (getting T-D points extra)

After we perform shifts from min(O,S), there is a way to make p win by shifts that give him T-D points

Thus, any shift that gives him 2(T-D) points, makes him a winner.

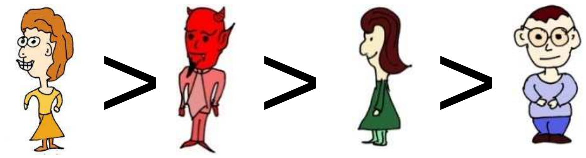
It is easy to find these 2(T-D) points. **We're done!**

The Algorithm (General Case)



The Algorithm

Why 2-approximation?



a_{i+1} a_i

 gains $a_{i+1} - a_i$ points

 loses $a_{i+1} - a_i$ points

Operation of the algorithm

1. Guess a cost k
2. Get most points for  at cost k
3. ~~Guess a cost k’ <= k~~
4. ~~Get most points for  at cost k’~~

Is this algorithm still a **2**-approximation? Unclear!

Complexity Barrier: Conclusions

- Complexity theory can mean protection from manipuation
  - Most cheating problems are NP-complete…
  - … but it is a worst-case notion
    - Approximation
    - Heuristics
    - FPT attachs (oops! Did not mention them)
- Some means of interpreting hardness/algorithmic results
  - Axiomatic view!

Thank You!