Voting and Bribing in Single-exponential Time

Dušan Knop, **Martin Koutecký**, Matthias Mnich

Department of Applied Mathematics, Charles University

STACS 2017, Feb 10 2017

Outline

- Intro to voting and bribery

Outline

- Intro to voting and bribery
  - Swap Bribery

Outline

- Intro to voting and bribery
  - Swap Bribery
  - Previous approach (double-exponential)

Outline

- Intro to voting and bribery
  - Swap Bribery
  - Previous approach (double-exponential)
- New technique: n-fold Integer Programming

Outline

- Intro to voting and bribery
  - Swap Bribery
  - Previous approach (double-exponential)
- New technique: n-fold Integer Programming
- Swap Bribery in single-exponential time via n-fold IP

Outline

- Intro to voting and bribery
  - Swap Bribery
  - Previous approach (double-exponential)
- New technique: n-fold Integer Programming
- Swap Bribery in single-exponential time via n-fold IP
- Extensions

Definitions: Elections

Definition (Elections)

An *election* $E = (C, V)$ consists of a set of *candidates* C ($|C| = m$) and a set of *voters* V ($|V| = n$) who indicate their preferences over the candidates C as total orders.

- $c \succ_v c'$ if v prefers c before c'
- $\mathrm{rank}(c, v) = i$ if c ranks as the i^{th} in v's order

Definitions: Who wins?

Definition (Voting rule)

A voting rule $\mathcal{R}$ is a function that maps an election (C, V) to a subset $W \subseteq C$ – the *winners*.

Definitions: Who wins?

Definition (Voting rule)

A voting rule $\mathcal{R}$ is a function that maps an election (C, V) to a subset $W \subseteq C$ – the *winners*.

One common example:

Definition (Scoring protocol)

- Given a vector of integers $\mathbf{s} = (s_1, \ldots, s_{|C|})$ with $s_1 \geq \cdots \geq s_{|C|} \geq 0$.

Definitions: Who wins?

Definition (Voting rule)

A voting rule $\mathcal{R}$ is a function that maps an election (C, V) to a subset $W \subseteq C$ – the *winners*.

One common example:

Definition (Scoring protocol)

- Given a vector of integers $\mathbf{s} = (s_1, \ldots, s_{|C|})$ with $s_1 \geq \cdots \geq s_{|C|} \geq 0$.
- For each $p \in \{1, \ldots, |C|\}$: $s_p =$ number of points that each candidate c receives from each voter that ranks c as p^{th} best.

Definitions: Who wins?

Definition (Voting rule)

A voting rule $\mathcal{R}$ is a function that maps an election (C, V) to a subset $W \subseteq C$ – the *winners*.

One common example:

Definition (Scoring protocol)

- Given a vector of integers $\mathbf{s} = (s_1, \ldots, s_{|C|})$ with $s_1 \geq \cdots \geq s_{|C|} \geq 0$.
- For each $p \in \{1, \ldots, |C|\}$: $s_p =$ number of points that each candidate c receives from each voter that ranks c as p^{th} best.
- Any candidate with the maximum number of points is the winner.

Definitions: Who wins?

Definition (Voting rule)

A voting rule $\mathcal{R}$ is a function that maps an election (C, V) to a subset $W \subseteq C$ – the *winners*.

One common example:

Definition (Scoring protocol)

- Given a vector of integers $\mathbf{s} = (s_1, \ldots, s_{|C|})$ with $s_1 \geq \cdots \geq s_{|C|} \geq 0$.
- For each $p \in \{1, \ldots, |C|\}$: $s_p =$ number of points that each candidate c receives from each voter that ranks c as p^{th} best.
- Any candidate with the maximum number of points is the winner.
- **Examples:** Plurality rule ($\mathbf{s} = (1, 0, \ldots, 0)$), d-Approval rule ($\mathbf{s} = (1, \ldots, 1, 0, \ldots, 0)$ with d ones), Borda rule ($\mathbf{s} = (|C| - 1, |C| - 2, \ldots, 1, 0)$).

Definitions: Swaps

Definition (Swaps)

Let (C, V) be an election and $\succ_v \in V$ be a voter. For candidates $c, c' \in C$, a *swap* $s = (c, c')_v$ means to exchange the positions of c and c' in $\succ_v$; the manipulated order is $\succ_v^s$.

Definitions: Swaps

Definition (Swaps)

Let (C, V) be an election and $\succ_v \in V$ be a voter. For candidates $c, c' \in C$, a *swap* $s = (c, c')_v$ means to exchange the positions of c and c' in $\succ_v$; the manipulated order is $\succ_v^s$.

- Swap $(c, c)_v$ is *admissible in* $\succ_v$ if $\mathrm{rank}(c, v) = \mathrm{rank}(c', v) - 1$.

Definitions: Swaps

Definition (Swaps)

Let (C, V) be an election and $\succ_v \in V$ be a voter. For candidates $c, c' \in C$, a *swap* $s = (c, c')_v$ means to exchange the positions of c and c' in $\succ_v$; the manipulated order is $\succ_v^s$.

- Swap $(c, c)_v$ is *admissible in* $\succ_v$ if $\mathrm{rank}(c, v) = \mathrm{rank}(c', v) - 1$.
- Set S of swaps is *admissible in* $\succ_v$ if they can be applied sequentially in $\succ_v$ in some order, s.t. each is admissible.

Definitions: Swaps

Definition (Swaps)

Let (C, V) be an election and $\succ_v \in V$ be a voter. For candidates $c, c' \in C$, a *swap* $s = (c, c')_v$ means to exchange the positions of c and c' in $\succ_v$; the manipulated order is $\succ_v^s$.

- Swap $(c, c)_v$ is *admissible in* $\succ_v$ if $\mathrm{rank}(c, v) = \mathrm{rank}(c', v) - 1$.
- Set S of swaps is *admissible in* $\succ_v$ if they can be applied sequentially in $\succ_v$ in some order, s.t. each is admissible.
- **Fact:** order of applying S does not matter

Definitions: Swaps

Definition (Swaps)

Let (C, V) be an election and $\succ_v \in V$ be a voter. For candidates $c, c' \in C$, a *swap* $s = (c, c')_v$ means to exchange the positions of c and c' in $\succ_v$; the manipulated order is $\succ_v^s$.

- Swap $(c, c)_v$ is *admissible in* $\succ_v$ if $\mathrm{rank}(c, v) = \mathrm{rank}(c', v) - 1$.
- Set S of swaps is *admissible in* $\succ_v$ if they can be applied sequentially in $\succ_v$ in some order, s.t. each is admissible.
- **Fact:** order of applying S does not matter
- **Crucial fact:** Given two orders $\succ$ and $\succ'$, $S = \{(i, j) \mid i \succ j \Leftrightarrow j \succ' i\}$ is the unique set s.t. $\succ^S = \succ'$.

Definitions: Swaps

Definition (Swaps)

Let (C, V) be an election and $\succ_v \in V$ be a voter. For candidates $c, c' \in C$, a *swap* $s = (c, c')_v$ means to exchange the positions of c and c' in $\succ_v$; the manipulated order is $\succ_v^s$.

- Swap $(c, c)_v$ is *admissible in* $\succ_v$ if $\mathrm{rank}(c, v) = \mathrm{rank}(c', v) - 1$.
- Set S of swaps is *admissible in* $\succ_v$ if they can be applied sequentially in $\succ_v$ in some order, s.t. each is admissible.
- **Fact:** order of applying S does not matter
- **Crucial fact:** Given two orders $\succ$ and $\succ'$, $S = \{(i, j) \mid i \succ j \Leftrightarrow j \succ' i\}$ is the unique set s.t. $\succ^S = \succ'$.
- Applying swaps in several votes: V^S

Definitions: Swaps

Definition (Swaps)

Let (C, V) be an election and $\succ_v \in V$ be a voter. For candidates $c, c' \in C$, a *swap* $s = (c, c')_v$ means to exchange the positions of c and c' in $\succ_v$; the manipulated order is $\succ_v^s$.

- Swap $(c, c)_v$ is *admissible in* $\succ_v$ if $\mathrm{rank}(c, v) = \mathrm{rank}(c', v) - 1$.
- Set S of swaps is *admissible in* $\succ_v$ if they can be applied sequentially in $\succ_v$ in some order, s.t. each is admissible.
- **Fact:** order of applying S does not matter
- **Crucial fact:** Given two orders $\succ$ and $\succ'$, $S = \{(i, j) \mid i \succ j \Leftrightarrow j \succ' i\}$ is the unique set s.t. $\succ^S = \succ'$.
- Applying swaps in several votes: V^S
- v's cost of swaps given by a function $\sigma^v : C \times C \to \mathbb{Z}$.

$\mathcal{R}$-Swap Bribery: Definition and example

$\mathcal{R}$-Swap Bribery	Parameter: $\lvert C \rvert$
Input:	An election $E = (C, V)$, voting rule $\mathcal{R}$, bribery cost functions σ_v for each $v \in V$, a designated candidate $c^\star$
Task:	Find a set S of admissible swaps of minimum cost s.t. $c^\star$ wins the election $E^S = (C, V^S)$ under rule $\mathcal{R}$.

$\mathcal{R}$-Swap Bribery: Definition and example

$\mathcal{R}$-Swap Bribery	Parameter: $\lvert C \rvert$
Input:	An election $E = (C, V)$, voting rule $\mathcal{R}$, bribery cost functions σ_v for each $v \in V$, a designated candidate $c^\star$
Task:	Find a set S of admissible swaps of minimum cost s.t. $c^\star$ wins the election $E^S = (C, V^S)$ under rule $\mathcal{R}$.

- $C = \{a, b, c\}$

$\mathcal{R}$-Swap Bribery: Definition and example

$\mathcal{R}$-Swap Bribery	Parameter: $\lvert C \rvert$
Input:	An election $E = (C, V)$, voting rule $\mathcal{R}$, bribery cost functions σ_v for each $v \in V$, a designated candidate $c^\star$
Task:	Find a set S of admissible swaps of minimum cost s.t. $c^\star$ wins the election $E^S = (C, V^S)$ under rule $\mathcal{R}$.

- $C = \{a, b, c\}$
- $V = \{\succ_1, \succ_2, \succ_3, \succ_4\}$

$\mathcal{R}$-Swap Bribery: Definition and example

$\mathcal{R}$-Swap Bribery	Parameter: $\lvert C \rvert$
Input:	An election $E = (C, V)$, voting rule $\mathcal{R}$, bribery cost functions σ_v for each $v \in V$, a designated candidate $c^\star$
Task:	Find a set S of admissible swaps of minimum cost s.t. $c^\star$ wins the election $E^S = (C, V^S)$ under rule $\mathcal{R}$.

- $C = \{a, b, c\}$
- $V = \{\succ_1, \succ_2, \succ_3, \succ_4\}$
  - $a \succ_1 b \succ_1 c$

$\mathcal{R}$-Swap Bribery: Definition and example

$\mathcal{R}$-Swap Bribery	Parameter: $\lvert C \rvert$
Input: An election $E = (C, V)$, voting rule $\mathcal{R}$, bribery cost functions σ_v for each $v \in V$, a designated candidate $c^\star$	
Task: Find a set S of admissible swaps of minimum cost s.t. $c^\star$ wins the election $E^S = (C, V^S)$ under rule $\mathcal{R}$.	

- $C = \{a, b, c\}$
- $V = \{\succ_1, \succ_2, \succ_3, \succ_4\}$
  - $a \succ_1 b \succ_1 c$
  - $a \succ_2 c \succ_2 b$

$\mathcal{R}$-Swap Bribery: Definition and example

$\mathcal{R}$-Swap Bribery	Parameter: $\lvert C \rvert$
Input: An election $E = (C, V)$, voting rule $\mathcal{R}$, bribery cost functions σ_v for each $v \in V$, a designated candidate $c^\star$	
Task: Find a set S of admissible swaps of minimum cost s.t. $c^\star$ wins the election $E^S = (C, V^S)$ under rule $\mathcal{R}$.	

- $C = \{a, b, c\}$
- $V = \{\succ_1, \succ_2, \succ_3, \succ_4\}$
  - $a \succ_1 b \succ_1 c$
  - $a \succ_2 c \succ_2 b$
  - $b \succ_3 c \succ_3 a$

$\mathcal{R}$-Swap Bribery: Definition and example

$\mathcal{R}$-Swap Bribery	Parameter: $\lvert C \rvert$
Input: An election $E = (C, V)$, voting rule $\mathcal{R}$, bribery cost functions σ_v for each $v \in V$, a designated candidate $c^\star$	
Task: Find a set S of admissible swaps of minimum cost s.t. $c^\star$ wins the election $E^S = (C, V^S)$ under rule $\mathcal{R}$.	

- $C = \{a, b, c\}$
- $V = \{\succ_1, \succ_2, \succ_3, \succ_4\}$
  - $a \succ_1 b \succ_1 c$
  - $a \succ_2 c \succ_2 b$
  - $b \succ_3 c \succ_3 a$
  - $b \succ_4 a \succ_4 c$

$\mathcal{R}$-Swap Bribery: Definition and example

$\mathcal{R}$-Swap Bribery	Parameter: $\lvert C\rvert$
Input: An election $E = (C, V)$, voting rule $\mathcal{R}$, bribery cost functions σ_v for each $v \in V$, a designated candidate $c^\star$	
Task: Find a set S of admissible swaps of minimum cost s.t. $c^\star$ wins the election $E^S = (C, V^S)$ under rule $\mathcal{R}$.	

- $C = \{a, b, c\}$
- $V = \{\succ_1, \succ_2, \succ_3, \succ_4\}$
  - $a \succ_1 b \succ_1 c$
  - $a \succ_2 c \succ_2 b$
  - $b \succ_3 c \succ_3 a$
  - $b \succ_4 a \succ_4 c$
- $\mathcal{R}$ = plurality (only first candidate in $\succ_v$ gets a point)

$\mathcal{R}$-Swap Bribery: Definition and example

$\mathcal{R}$-Swap Bribery	Parameter: $\lvert C \rvert$
Input:	An election $E = (C, V)$, voting rule $\mathcal{R}$, bribery cost functions σ_v for each $v \in V$, a designated candidate $c^\star$
Task:	Find a set S of admissible swaps of minimum cost s.t. $c^\star$ wins the election $E^S = (C, V^S)$ under rule $\mathcal{R}$.

- $C = \{a, b, c\}$
- $V = \{\succ_1, \succ_2, \succ_3, \succ_4\}$
  - $a \succ_1 b \succ_1 c$
  - $a \succ_2 c \succ_2 b$
  - $b \succ_3 c \succ_3 a$
  - $b \succ_4 a \succ_4 c$
- $\mathcal{R}$ = plurality (only first candidate in $\succ_v$ gets a point)
- Unit costs: $\sigma^v(c, c') = 1$ for all $v \in V$, $c \neq c' \in C$

$\mathcal{R}$-Swap Bribery: Definition and example

$\mathcal{R}$-Swap Bribery	Parameter: $\lvert C \rvert$
Input: An election $E = (C, V)$, voting rule $\mathcal{R}$, bribery cost functions σ_v for each $v \in V$, a designated candidate $c^\star$	
Task: Find a set S of admissible swaps of minimum cost s.t. $c^\star$ wins the election $E^S = (C, V^S)$ under rule $\mathcal{R}$.	

- $C = \{a, b, c\}$
- $V = \{\succ_1, \succ_2, \succ_3, \succ_4\}$
  - $a \succ_1 b \succ_1 c$
  - $a \succ_2 c \succ_2 b$
  - $b \succ_3 c \succ_3 a$
  - $b \succ_4 a \succ_4 c$
- $\mathcal{R}$ = plurality (only first candidate in $\succ_v$ gets a point)
- Unit costs: $\sigma^v(c, c') = 1$ for all $v \in V$, $c \neq c' \in C$
- $S = \{(a, c)_2, (b, c)_3\}$

$\mathcal{R}$-Swap Bribery: Definition and example

$\mathcal{R}$-Swap Bribery	Parameter: $\lvert C\rvert$
Input:	An election $E = (C, V)$, voting rule $\mathcal{R}$, bribery cost functions σ_v for each $v \in V$, a designated candidate $c^\star$
Task:	Find a set S of admissible swaps of minimum cost s.t. $c^\star$ wins the election $E^S = (C, V^S)$ under rule $\mathcal{R}$.

- $C = \{a, b, c\}$
- $V = \{\succ_1, \succ_2, \succ_3, \succ_4\}$
  - $a \succ_1 b \succ_1 c$
  - $a \succ_2 c \succ_2 b$
  - $b \succ_3 c \succ_3 a$
  - $b \succ_4 a \succ_4 c$
- $\mathcal{R}$ = plurality (only first candidate in $\succ_v$ gets a point)
- Unit costs: $\sigma^v(c, c') = 1$ for all $v \in V$, $c \neq c' \in C$
- $S = \{(a, c)_2, (b, c)_3\}$

$\mathcal{R}$-Swap Bribery: Definition and example

$\mathcal{R}$-Swap Bribery	Parameter: $\lvert C \rvert$
Input:	An election $E = (C, V)$, voting rule $\mathcal{R}$, bribery cost functions σ_v for each $v \in V$, a designated candidate $c^\star$
Task:	Find a set S of admissible swaps of minimum cost s.t. $c^\star$ wins the election $E^S = (C, V^S)$ under rule $\mathcal{R}$.

- $C = \{a, b, c\}$
- $V = \{\succ_1, \succ_2, \succ_3, \succ_4\}$
  - $a \succ_1 b \succ_1 c$
  - $a \succ_2 c \succ_2 b$
  - $b \succ_3 c \succ_3 a$
  - $b \succ_4 a \succ_4 c$
- $\mathcal{R}$ = plurality (only first candidate in $\succ_v$ gets a point)
- Unit costs: $\sigma^v(c, c') = 1$ for all $v \in V$, $c \neq c' \in C$
- $S = \{(a, c)_2, (b, c)_3\}$ (cost of S is 2)

$\mathcal{R}$-Swap Bribery: Definition and example

$\mathcal{R}$-Swap Bribery	Parameter: $\lvert C\rvert$
Input:	An election $E = (C, V)$, voting rule $\mathcal{R}$, bribery cost functions σ_v for each $v \in V$, a designated candidate $c^\star$
Task:	Find a set S of admissible swaps of minimum cost s.t. $c^\star$ wins the election $E^S = (C, V^S)$ under rule $\mathcal{R}$.

- $C = \{a, b, c\}$
- $V = \{\succ_1, \succ_2, \succ_3, \succ_4\}$
  - $a \succ_1 b \succ_1 c$
  - $a \succ_2 c \succ_2 b$
  - $b \succ_3 c \succ_3 a$
  - $b \succ_4 a \succ_4 c$
- $\mathcal{R}$ = plurality (only first candidate in $\succ_v$ gets a point)
- Unit costs: $\sigma^v(c, c') = 1$ for all $v \in V$, $c \neq c' \in C$
- $S = \{(a, c)_2, (b, c)_3\}$ (cost of S is 2)
  (also $S = \{(a, c)_2, (a, c)_4, (b, c)_4\}$)
- $\Rightarrow$ c wins (C, V^S)

$\mathcal{R}$-Swap Bribery: The complexity story

Our lens: Parameterized Complexity

- Input size n, parameter k; expect n to be large but k small

$\mathcal{R}$-Swap Bribery: The complexity story

Our lens: Parameterized Complexity

- Input size n, parameter k; expect n to be large but k small
- Question: does $f(k)n^{O(1)}$ algorithm exist (for some f)?

$\mathcal{R}$-Swap Bribery: The complexity story

Our lens: Parameterized Complexity

- Input size n, parameter k; expect n to be large but k small
- Question: does $f(k)n^{O(1)}$ algorithm exist (for some f)?
- If YES: get $f(k)$ as small as possible

$\mathcal{R}$-Swap Bribery: The complexity story

Our lens: Parameterized Complexity

- Input size n, parameter k; expect n to be large but k small
- Question: does $f(k)n^{O(1)}$ algorithm exist (for some f)?
- If YES: get $f(k)$ as small as possible

$\mathcal{R}$-Swap Bribery: The complexity story

Our lens: Parameterized Complexity

- Input size n, parameter k; expect n to be large but k small
- Question: does $f(k)n^{O(1)}$ algorithm exist (for some f)?
- If YES: get $f(k)$ as small as possible

The story of $\mathcal{R}$-Swap Bribery

- Previous approach: $m^{O(m^m)} \log n$ with unit costs [Dorn, Schlotter 2010]

$\mathcal{R}$-Swap Bribery: The complexity story

Our lens: Parameterized Complexity

- Input size n, parameter k; expect n to be large but k small
- Question: does $f(k)n^{O(1)}$ algorithm exist (for some f)?
- If YES: get $f(k)$ as small as possible

The story of $\mathcal{R}$-Swap Bribery

- Previous approach: $m^{O(m^m)} \log n$ with unit costs [Dorn, Schlotter 2010]
- Similar complexity (and approach) to many related problems with parameter m

$\mathcal{R}$-Swap Bribery: The complexity story

Our lens: Parameterized Complexity

- Input size n, parameter k; expect n to be large but k small
- Question: does $f(k)n^{O(1)}$ algorithm exist (for some f)?
- If YES: get $f(k)$ as small as possible

The story of $\mathcal{R}$-Swap Bribery

- Previous approach: $m^{O(m^m)} \log n$ with unit costs [Dorn, Schlotter 2010]
- Similar complexity (and approach) to many related problems with parameter m
- [Bredereck et al. 2014: Nine research challenges]:

$\mathcal{R}$-Swap Bribery: The complexity story

Our lens: Parameterized Complexity

- Input size n, parameter k; expect n to be large but k small
- Question: does $f(k)n^{O(1)}$ algorithm exist (for some f)?
- If YES: get $f(k)$ as small as possible

The story of $\mathcal{R}$-Swap Bribery

- Previous approach: $m^{O(m^m)} \log n$ with unit costs [Dorn, Schlotter 2010]
- Similar complexity (and approach) to many related problems with parameter m
- [Bredereck et al. 2014: Nine research challenges]:
  - Challenge #1: Get $f(m)$ from double-exp to single-exp

$\mathcal{R}$-Swap Bribery: The complexity story

Our lens: Parameterized Complexity

- Input size n, parameter k; expect n to be large but k small
- Question: does $f(k)n^{O(1)}$ algorithm exist (for some f)?
- If YES: get $f(k)$ as small as possible

The story of $\mathcal{R}$-Swap Bribery

- Previous approach: $m^{O(m^m)} \log n$ with unit costs [Dorn, Schlotter 2010]
- Similar complexity (and approach) to many related problems with parameter m
- [Bredereck et al. 2014: Nine research challenges]:
  - Challenge #1: Get $f(m)$ from double-exp to single-exp
  - Challenge #2: Go from unit costs to general costs

$\mathcal{R}$-Swap Bribery: The complexity story

Our lens: Parameterized Complexity

- Input size n, parameter k; expect n to be large but k small
- Question: does $f(k)n^{O(1)}$ algorithm exist (for some f)?
- If YES: get $f(k)$ as small as possible

The story of $\mathcal{R}$-Swap Bribery

- Previous approach: $m^{O(m^m)} \log n$ with unit costs [Dorn, Schlotter 2010]
- Similar complexity (and approach) to many related problems with parameter m
- [Bredereck et al. 2014: Nine research challenges]:
  - Challenge #1: Get $f(m)$ from double-exp to single-exp
  - Challenge #2: Go from unit costs to general costs
- **Our result:** $m^{O(m^6)}n^3$ with arbitrary costs.

Previous approach: ILP in fixed dimension

All previous results use Integer Linear Programming:

$$\min\{\mathbf{c}^\top \mathbf{x} | A\mathbf{x} \leq \mathbf{b}, \mathbf{x} \in \mathbb{Z}^k\} \ .$$

Theorem ([Lenstra 1983])

ILP can be solved in time $k^{O(k)}L$, where $L = \langle A, \mathbf{c}, \mathbf{b}\rangle$.

Previous approach: ILP in fixed dimension

All previous results use Integer Linear Programming:

$$\min\{\mathbf{c}^\top \mathbf{x} | A\mathbf{x} \leq \mathbf{b}, \mathbf{x} \in \mathbb{Z}^k\} \ .$$

Theorem ([Lenstra 1983])

ILP can be solved in time $k^{O(k)}L$, *where* $L = \langle A, \mathbf{c}, \mathbf{b} \rangle$.

Example: Swap Bribery with score $\mathbf{s} = (s_1, \ldots, s_m)$.

Previous approach: ILP in fixed dimension

All previous results use Integer Linear Programming:

$$\min\{\mathbf{c}^\top \mathbf{x} | A\mathbf{x} \leq \mathbf{b}, \mathbf{x} \in \mathbb{Z}^k\} \ .$$

Theorem ([Lenstra 1983])

ILP can be solved in time $k^{O(k)}L$, *where* $L = \langle A, \mathbf{c}, \mathbf{b} \rangle$.

Example: Swap Bribery with score $\mathbf{s} = (s_1, \ldots, s_m)$.
Observation: m candidates $\Rightarrow$ $m!$ voter types

Previous approach: ILP in fixed dimension

All previous results use Integer Linear Programming:

$$\min\{\mathbf{c}^\top \mathbf{x} | A\mathbf{x} \leq \mathbf{b}, \mathbf{x} \in \mathbb{Z}^k\} \ .$$

Theorem ([Lenstra 1983])

ILP can be solved in time $k^{O(k)}L$*, where* $L = \langle A, \mathbf{c}, \mathbf{b} \rangle$*.*

Example: Swap Bribery with score $\mathbf{s} = (s_1, \ldots, s_m)$.
Observation: m candidates $\Rightarrow$ $m!$ voter types
Variables, coefficients, etc:

Previous approach: ILP in fixed dimension

All previous results use Integer Linear Programming:

$$\min\{\mathbf{c}^\top \mathbf{x} | A\mathbf{x} \leq \mathbf{b}, \mathbf{x} \in \mathbb{Z}^k\} \ .$$

Theorem ([Lenstra 1983])

ILP can be solved in time $k^{O(k)}L$*, where* $L = \langle A, \mathbf{c}, \mathbf{b} \rangle$*.*

Example: Swap Bribery with score $\mathbf{s} = (s_1, \ldots, s_m)$.
Observation: m candidates $\Rightarrow$ $m!$ voter types
Variables, coefficients, etc:

- z_{ij} = number of voters of type i swapped to type j

Previous approach: ILP in fixed dimension

All previous results use Integer Linear Programming:

$$\min\{\mathbf{c}^\top \mathbf{x} | A\mathbf{x} \leq \mathbf{b}, \mathbf{x} \in \mathbb{Z}^k\} \ .$$

Theorem ([Lenstra 1983])

ILP can be solved in time $k^{O(k)}L$*, where* $L = \langle A, \mathbf{c}, \mathbf{b} \rangle$*.*

Example: Swap Bribery with score $\mathbf{s} = (s_1, \ldots, s_m)$.
Observation: m candidates $\Rightarrow$ $m!$ voter types
Variables, coefficients, etc:

- z_{ij} = number of voters of type i swapped to type j
- c_{ij} = number of swaps from type i to type j ($i = j \Rightarrow c_{ij} = 0$)

Previous approach: ILP in fixed dimension

All previous results use Integer Linear Programming:

$$\min\{\mathbf{c}^\top \mathbf{x} | A\mathbf{x} \leq \mathbf{b}, \mathbf{x} \in \mathbb{Z}^k\} \ .$$

Theorem ([Lenstra 1983])

ILP can be solved in time $k^{O(k)}L$, where $L = \langle A, \mathbf{c}, \mathbf{b}\rangle$.

Example: Swap Bribery with score $\mathbf{s} = (s_1, \ldots, s_m)$.
Observation: m candidates $\Rightarrow$ $m!$ voter types
Variables, coefficients, etc:

- z_{ij} = number of voters of type i swapped to type j
- c_{ij} = number of swaps from type i to type j ($i = j \Rightarrow c_{ij} = 0$)
- n_i = number of voters of type i on input

Previous approach: ILP in fixed dimension

All previous results use Integer Linear Programming:

$$\min\{\mathbf{c}^\top \mathbf{x} | A\mathbf{x} \leq \mathbf{b}, \mathbf{x} \in \mathbb{Z}^k\} \ .$$

Theorem ([Lenstra 1983])

ILP can be solved in time $k^{O(k)}L$, where $L = \langle A, \mathbf{c}, \mathbf{b}\rangle$.

Example: Swap Bribery with score $\mathbf{s} = (s_1, \ldots, s_m)$.
Observation: m candidates $\Rightarrow$ $m!$ voter types
Variables, coefficients, etc:

- z_{ij} = number of voters of type i swapped to type j
- c_{ij} = number of swaps from type i to type j ($i = j \Rightarrow c_{ij} = 0$)
- n_i = number of voters of type i on input
- x_j = number of voters of type j after bribery

Previous approach: ILP in fixed dimension

All previous results use Integer Linear Programming:

$$\min\{\mathbf{c}^\top \mathbf{x} | A\mathbf{x} \leq \mathbf{b}, \mathbf{x} \in \mathbb{Z}^k\} \ .$$

Theorem ([Lenstra 1983])

ILP can be solved in time $k^{O(k)}L$, where $L = \langle A, \mathbf{c}, \mathbf{b}\rangle$.

Example: Swap Bribery with score $\mathbf{s} = (s_1, \ldots, s_m)$.
Observation: m candidates $\Rightarrow$ $m!$ voter types
Variables, coefficients, etc:

- z_{ij} = number of voters of type i swapped to type j
- c_{ij} = number of swaps from type i to type j ($i = j \Rightarrow c_{ij} = 0$)
- n_i = number of voters of type i on input
- x_j = number of voters of type j after bribery
- S_c = score obtained by candidate c

Previous approach: ILP in fixed dimension

$$\min \quad \sum_{i,j=1}^{m!} c_{ij} z_{ij}$$

$$\text{s.t.} \quad \sum_{j=1}^{m!} z_{ij} = n_i \qquad \forall i = 1, \ldots, m!$$

$$\sum_{i=1}^{m!} z_{ij} = x_j \qquad \forall j = 1, \ldots, m!$$

$$\sum_{p=1}^{m} \sum_{\substack{j=1,\ldots,m! \\ c \text{ is } p^{\text{th}} \text{ in } j}} s_p x_j = S_c \qquad \forall c \in C$$

$$S_{c^\star} \geq S_{c'} \qquad \forall c^\star \neq c' \in C$$

Number of variables $O(m!) \Rightarrow$ runtime $m!^{m!} \log(n)$ by Lenstra's algorithm.

New technique: n-Fold Integer Programming

n-Fold Integer Programming:

$$\min\left\{\mathbf{wx} \;:\; E^{(n)}\mathbf{x} = \mathbf{b}\,,\; \mathbf{l} \le \mathbf{x} \le \mathbf{u}\,,\; \mathbf{x} \in \mathbb{Z}^{nt}\right\}\,,$$

$$E^{(n)} \quad := \quad \begin{pmatrix} D & D & \cdots & D \\ A & 0 & \cdots & 0 \\ 0 & A & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & A \end{pmatrix}$$

Large theory: Graver bases, primal optimization

Theorem ([Hemmecke, Onn, Romanchuk 2011])

n-fold IP can be solved in time $a^{O(rst+st^2)}n^3L$, where $a = \max\{\|A\|_\infty, \|D\|_\infty\}$, $D \in \mathbb{Z}^{r\times t}$, $A \in \mathbb{Z}^{s\times t}$ and $L = \langle \mathbf{l}, \mathbf{u}, \mathbf{b}, \mathbf{w}\rangle$.

Making sense of n-Fold IP: bricks, constraints, etc.

$$E^{(n)} := \begin{pmatrix} D & D & \cdots & D \\ A & 0 & \cdots & 0 \\ 0 & A & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & A \end{pmatrix}$$

- $\mathbf{x} \in \mathbb{Z}^{nt}$ partitions into n *bricks* of size t: $\mathbf{x} = (\mathbf{x}^1, \ldots, \mathbf{x}^n)$ (upper index always denotes a brick)

Making sense of n-Fold IP: bricks, constraints, etc.

$$E^{(n)} := \begin{pmatrix} D & D & \cdots & D \\ A & 0 & \cdots & 0 \\ 0 & A & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & A \end{pmatrix}$$

- $\mathbf{x} \in \mathbb{Z}^{nt}$ partitions into n *bricks* of size t: $\mathbf{x} = (\mathbf{x}^1, \ldots, \mathbf{x}^n)$ (upper index always denotes a brick)
- Two kinds of constraints:

Making sense of n-Fold IP: bricks, constraints, etc.

$$E^{(n)} := \begin{pmatrix} D & D & \cdots & D \\ A & 0 & \cdots & 0 \\ 0 & A & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & A \end{pmatrix}$$

- $\mathbf{x} \in \mathbb{Z}^{nt}$ partitions into n *bricks* of size t: $\mathbf{x} = (\mathbf{x}^1, \ldots, \mathbf{x}^n)$ (upper index always denotes a brick)
- Two kinds of constraints:
  - *Globally uniform constraints*: upper row of D's

Making sense of n-Fold IP: bricks, constraints, etc.

$$E^{(n)} := \begin{pmatrix} D & D & \cdots & D \\ A & 0 & \cdots & 0 \\ 0 & A & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & A \end{pmatrix}$$

- $\mathbf{x} \in \mathbb{Z}^{nt}$ partitions into n *bricks* of size t: $\mathbf{x} = (\mathbf{x}^1, \ldots, \mathbf{x}^n)$ (upper index always denotes a brick)
- Two kinds of constraints:
  - *Globally uniform constraints*: upper row of D's
  - *Locally uniform constraints:* lower blocks A (note: same matrix for each brick, but possibly different right hand side)

Making sense of n-Fold IP: bricks, constraints, etc.

$$E^{(n)} := \begin{pmatrix} D & D & \cdots & D \\ A & 0 & \cdots & 0 \\ 0 & A & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & A \end{pmatrix}$$

- $\mathbf{x} \in \mathbb{Z}^{nt}$ partitions into n *bricks* of size t: $\mathbf{x} = (\mathbf{x}^1, \ldots, \mathbf{x}^n)$ (upper index always denotes a brick)
- Two kinds of constraints:
  - *Globally uniform constraints*: upper row of D's
  - *Locally uniform constraints:* lower blocks A (note: same matrix for each brick, but possibly different right hand side)
- **"Theorem"**: possible to use standard IP tricks

Making sense of n-Fold IP: bricks, constraints, etc.

$$E^{(n)} := \begin{pmatrix} D & D & \cdots & D \\ A & 0 & \cdots & 0 \\ 0 & A & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & A \end{pmatrix}$$

- $\mathbf{x} \in \mathbb{Z}^{nt}$ partitions into n *bricks* of size t: $\mathbf{x} = (\mathbf{x}^1, \ldots, \mathbf{x}^n)$ (upper index always denotes a brick)
- Two kinds of constraints:
  - *Globally uniform constraints*: upper row of D's
  - *Locally uniform constraints:* lower blocks A (note: same matrix for each brick, but possibly different right hand side)
- **"Theorem"**: possible to use standard IP tricks
  - inequalities ($x \leq y$),

Making sense of n-Fold IP: bricks, constraints, etc.

$$E^{(n)} := \begin{pmatrix} D & D & \cdots & D \\ A & 0 & \cdots & 0 \\ 0 & A & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & A \end{pmatrix}$$

- $\mathbf{x} \in \mathbb{Z}^{nt}$ partitions into n *bricks* of size t: $\mathbf{x} = (\mathbf{x}^1, \ldots, \mathbf{x}^n)$ (upper index always denotes a brick)
- Two kinds of constraints:
  - *Globally uniform constraints*: upper row of D's
  - *Locally uniform constraints:* lower blocks A (note: same matrix for each brick, but possibly different right hand side)
- **"Theorem"**: possible to use standard IP tricks
  - inequalities ($x \leq y$),
  - logical connectives ($x \wedge y$),

Making sense of n-Fold IP: bricks, constraints, etc.

$$E^{(n)} := \begin{pmatrix} D & D & \cdots & D \\ A & 0 & \cdots & 0 \\ 0 & A & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & A \end{pmatrix}$$

- $\mathbf{x} \in \mathbb{Z}^{nt}$ partitions into n *bricks* of size t: $\mathbf{x} = (\mathbf{x}^1, \ldots, \mathbf{x}^n)$ (upper index always denotes a brick)
- Two kinds of constraints:
  - *Globally uniform constraints*: upper row of D's
  - *Locally uniform constraints:* lower blocks A (note: same matrix for each brick, but possibly different right hand side)
- **"Theorem"**: possible to use standard IP tricks
  - inequalities ($x \leq y$),
  - logical connectives ($x \wedge y$),
  - $\mathrm{bool}(x)$ and $\mathrm{sign}(x)$ if x is not too large.

$\mathcal{R}$-Swap Bribery via n-fold IP

We will construct an n-fold IP with:

- a brick per voter $\Rightarrow$ n bricks,

$\mathcal{R}$-Swap Bribery via n-fold IP

We will construct an n-fold IP with:

- a brick per voter $\Rightarrow$ n bricks,
- variable $x_c^v = \mathrm{rank}^S(c, v)$ (bribed order),

$\mathcal{R}$-Swap Bribery via n-fold IP

We will construct an n-fold IP with:

- a brick per voter $\Rightarrow$ n bricks,
- variable $x_c^v = \mathrm{rank}^S(c, v)$ (bribed order),
- binary variable $s_{cc'}^v = 1$ if $(c, c')_v \in S$,

$\mathcal{R}$-Swap Bribery via n-fold IP

We will construct an n-fold IP with:

- a brick per voter $\Rightarrow$ n bricks,
- variable $x_c^v = \mathrm{rank}^S(c, v)$ (bribed order),
- binary variable $s_{cc'}^v = 1$ if $(c, c')_v \in S$,
- variable $\tau_c^v =$ score contribution of v to c,

$\mathcal{R}$-Swap Bribery via n-fold IP

We will construct an n-fold IP with:

- a brick per voter $\Rightarrow$ n bricks,
- variable $x_c^v = \mathrm{rank}^S(c, v)$ (bribed order),
- binary variable $s_{cc'}^v = 1$ if $(c, c')_v \in S$,
- variable $\tau_c^v =$ score contribution of v to c,
- constants $o_c^v = \mathrm{rank}(c, v)$.

$\mathcal{R}$-Swap Bribery via n-fold IP

We will construct an n-fold IP with:

- a brick per voter $\Rightarrow$ n bricks,
- variable $x_c^v = \mathrm{rank}^S(c, v)$ (bribed order),
- binary variable $s_{cc'}^v = 1$ if $(c, c')_v \in S$,
- variable $\tau_c^v =$ score contribution of v to c,
- constants $o_c^v = \mathrm{rank}(c, v)$.

$\mathcal{R}$-Swap Bribery via n-fold IP

We will construct an n-fold IP with:

- a brick per voter $\Rightarrow$ n bricks,
- variable $x_c^v = \mathrm{rank}^S(c, v)$ (bribed order),
- binary variable $s_{cc'}^v = 1$ if $(c, c')_v \in S$,
- variable $\tau_c^v =$ score contribution of v to c,
- constants $o_c^v = \mathrm{rank}(c, v)$.

Objective: minimize $\sum_v \sum_{(c,c') \in C^2} \sigma^v(c, c') s_{cc'}^v$

Constraints:

- x_c^v are distinct between 1 and m (a permutation).

$\mathcal{R}$-Swap Bribery via n-fold IP

We will construct an n-fold IP with:

- a brick per voter $\Rightarrow$ n bricks,
- variable $x_c^v = \mathrm{rank}^S(c, v)$ (bribed order),
- binary variable $s_{cc'}^v = 1$ if $(c, c')_v \in S$,
- variable $\tau_c^v =$ score contribution of v to c,
- constants $o_c^v = \mathrm{rank}(c, v)$.

Objective: minimize $\sum_v \sum_{(c,c') \in C^2} \sigma^v(c, c') s_{cc'}^v$

Constraints:

- x_c^v are distinct between 1 and m (a permutation).
- $s_{cc'}^v = 1$ iff $x_c^v > x_{c'}^v \Leftrightarrow o_c^v < o_{c'}^v$
  ("c and c' are swapped in $\succ_v^S$")

$\mathcal{R}$-Swap Bribery via n-fold IP

We will construct an n-fold IP with:

- a brick per voter $\Rightarrow$ n bricks,
- variable $x_c^v = \mathrm{rank}^S(c, v)$ (bribed order),
- binary variable $s_{cc'}^v = 1$ if $(c, c')_v \in S$,
- variable $\tau_c^v =$ score contribution of v to c,
- constants $o_c^v = \mathrm{rank}(c, v)$.

Objective: minimize $\sum_v \sum_{(c,c') \in C^2} \sigma^v(c, c') s_{cc'}^v$
Constraints:

- x_c^v are distinct between 1 and m (a permutation).
- $s_{cc'}^v = 1$ iff $x_c^v > x_{c'}^v \Leftrightarrow o_c^v < o_{c'}^v$
  ("c and c' are swapped in $\succ_v^S$")
- $c^\star$ wins, i.e. $\sum_v \tau_{c^\star}^v \geq \sum_v \tau_{c'}^v$ for all other candidates c'.

n-fold IP formulation

Objective:

$$\min \sum_{v} \sum_{(c,c') \in C^2} \sigma^v(c, c') s^v_{cc'}$$

Locally uniform constraints (for every voter v):

$$\text{s.t.} \quad \sum_{c} x^v_c = \binom{m+1}{2} \bigwedge x^v_c \neq_m x^v_c \qquad \forall c \neq c'$$

$$s^v_{cc'} = \text{bool}_2 \left(\text{sign}_m(x^v_c - x^v_{c'}) = \text{sign}_m(o^v_c - o^v_{c'}) \right) \qquad \forall c \neq c'$$

$$\tau^v_c = \sum_{k=1}^{m} s^v_c \, \text{bool}_m(x^v_c = k) \qquad \forall c$$

Globally uniform constraints:

$$\sum_{v \in V} \tau^{va}_c < \sum_{v \in V} \tau^{va}_{c^\star} \qquad \forall c \in C \setminus \{c^\star\}$$

Extensions

In fact, we solve many other problems by introducing a "meta-problem" $\mathcal{R}$-Multi Bribery, which adds

- **Push actions:** some rules consider an "approval count" of a voter – how many first candidates does a voter support? Push actions perturb the approval count.

Extensions

In fact, we solve many other problems by introducing a "meta-problem" $\mathcal{R}$-Multi Bribery, which adds

- **Push actions:** some rules consider an "approval count" of a voter – how many first candidates does a voter support? Push actions perturb the approval count.
- **Control changes:** adding "latent" voters and deleting "active" voters.

Extensions

In fact, we solve many other problems by introducing a "meta-problem" $\mathcal{R}$-Multi Bribery, which adds

- **Push actions:** some rules consider an "approval count" of a voter – how many first candidates does a voter support? Push actions perturb the approval count.
- **Control changes:** adding "latent" voters and deleting "active" voters.

Extensions

In fact, we solve many other problems by introducing a "meta-problem" $\mathcal{R}$-Multi Bribery, which adds

- **Push actions:** some rules consider an "approval count" of a voter – how many first candidates does a voter support? Push actions perturb the approval count.
- **Control changes:** adding "latent" voters and deleting "active" voters.

Also can solve for other rules $\mathcal{R}$, such as: Copeland, SP-AV, Maximin, Bucklin, Fallback etc. (Kemeny in double-exp time.)

Extensions

In fact, we solve many other problems by introducing a "meta-problem" $\mathcal{R}$-Multi Bribery, which adds

- **Push actions:** some rules consider an "approval count" of a voter – how many first candidates does a voter support? Push actions perturb the approval count.
- **Control changes:** adding "latent" voters and deleting "active" voters.

Also can solve for other rules $\mathcal{R}$, such as: Copeland, SP-AV, Maximin, Bucklin, Fallback etc. (Kemeny in double-exp time.)

Thus, we generalize the following problems (with $\mathcal{R}$ one of the above): \$Bribery, Manipulation, CCAV/CCDV, Shift Bribery, Support Bribery, Extension Bribery, Possible Winner, Dodgson Score, Young Score.

Extensions

In fact, we solve many other problems by introducing a "meta-problem" $\mathcal{R}$-Multi Bribery, which adds

- **Push actions:** some rules consider an "approval count" of a voter – how many first candidates does a voter support? Push actions perturb the approval count.
- **Control changes:** adding "latent" voters and deleting "active" voters.

Also can solve for other rules $\mathcal{R}$, such as: Copeland, SP-AV, Maximin, Bucklin, Fallback etc. (Kemeny in double-exp time.)

Thus, we generalize the following problems (with $\mathcal{R}$ one of the above): \$Bribery, Manipulation, CCAV/CCDV, Shift Bribery, Support Bribery, Extension Bribery, Possible Winner, Dodgson Score, Young Score.

How: obvious attempts works with one more IP trick (disjunctions).

Open problems

- **Double $\rightarrow$ single-exp:** try speeding up your favorite parameterized problem with double-exp complexity!

Open problems

- **Double $\rightarrow$ single-exp:** try speeding up your favorite parameterized problem with double-exp complexity!
- **Lower bounds:** with some new technology, we can go from k^{k^6} to k^{k^2} $\rightarrow$ lower bounds? (Also: Closest String, Makespan minimization.)

Open problems

- **Double $\rightarrow$ single-exp:** try speeding up your favorite parameterized problem with double-exp complexity!
- **Lower bounds:** with some new technology, we can go from k^{k^6} to k^{k^2} $\rightarrow$ lower bounds? (Also: Closest String, Makespan minimization.)

Open problems

- **Double $\rightarrow$ single-exp:** try speeding up your favorite parameterized problem with double-exp complexity!
- **Lower bounds:** with some new technology, we can go from k^{k^6} to k^{k^2} $\rightarrow$ lower bounds? (Also: Closest String, Makespan minimization.)

Thank you!