A Control Dichotomy for Pure Scoring Rules

Edith Hemaspaandra, Lane Hemaspaandra, Henning Schnoor

(The TR version is in arXiv.org and a conference version is in AAAI-14.)
Electoral Control

The “chair” of the election changes its structure/participants to affect the outcome.

Constructive Control by Adding Voters [BTT92]

**Problem:** $\mathcal{E}$-CCAV

**Input:** Registered voters $R$, unregistered voters $U$, preferred candidate $p$, $k \in \mathbb{N}$

**Question:** Can we add at most $k$ voters from $U$ such that $p$ is a winner under election system $\mathcal{E}$?

**Goal:** Determine the complexity of $\mathcal{E}$-CCAV for every $\mathcal{E}$. 
Scoring Rules

Large, important class of election systems in a common framework.

\( \alpha = (\alpha_1, \ldots, \alpha_n), \ \alpha_1 \geq \alpha_2 \geq \cdots \geq \alpha_n \) defines **scoring rule**:

- voters ranks candidates best to worst
- \( i \)th position gets \( \alpha_i \) points
- candidates with most points win
Scoring Rules

Large, important class of election systems in a common framework.

\[ \alpha = (\alpha_1, \ldots, \alpha_n), \quad \alpha_1 \geq \alpha_2 \geq \cdots \geq \alpha_n \] defines scoring rule:

- voters ranks candidates best to worst
- \( i \)th position gets \( \alpha_i \) points
- candidates with most points win

Examples

- \( k \)-approval: \( \alpha = (1, \ldots, 1, 0, \ldots, 0) \)
- \( k \)-veto: \( \alpha = (1, \ldots, 1, 0, \ldots, 0) \)
- Borda: \( (n - 1, n - 2, \ldots, 1, 0) \)
- Dowdall: \( (1, \frac{1}{2}, \frac{1}{3}, \ldots, \frac{1}{n}) \)
Scoring Rule Families

Scoring Rules

- Defined by vector $\alpha = (\alpha_1, \ldots, \alpha_n)$
- Election system for $n$ candidates

*Want:* system for arbitrary number of candidates
Scoring Rule Families

Scoring Rules

- Defined by vector $\alpha = (\alpha_1, \ldots, \alpha_n)$
- Election system for $n$ candidates

Want: system for arbitrary number of candidates

Generalized Scoring Rules

- generator: function $f$ such that $f(n)$ is scoring vector of length $n$.
- generator $f \rightsquigarrow$ election system $\mathcal{E}_f$

Identify $f$ and $\mathcal{E}_f$. Talk about $f$-CCAV.
Uniformity and other conditions

**Restrictions: Generators $f$ should be “nice”!**

**Computationally** $f$ should be p-time computable (in the number of candidates).

**Conceptually** This does not make much sense:

- $f(1) = (1)$
- $f(2) = (3, 1)$
- $f(3) = (1, 0, 0)$
- $f(4) = (1, 0, 0, 0)$
- $f(5) = (7, 5, 5, 0, 0)$
- $f(6) = (1, 0, 0, 0, 0, 0)$
- ...
Uniformity and other conditions

Restrictions: Generators $f$ should be “nice”!

Computationally $f$ should be p-time computable (in the number of candidates).

Conceptually This does not make much sense:

- $f(1) = (1)$
- $f(2) = (3, 1)$
- $f(3) = (1, 0, 0)$
- $f(4) = (1, 0, 0, 0)$
- $f(5) = (7, 5, 5, 0, 0)$
- $f(6) = (1, 0, 0, 0, 0, 0)$
- $\ldots$

Vectors $f(n)$ and $f(n + 1)$ should be related.
**Purity Constraint**

Pure generator $f$: p-time computable in $n$ and vector $f(n+1)$ obtained from $f(n)$ by adding single coefficient.
Pure Scoring Rules [BD10]

Purity Constraint

Pure generator $f$: p-time computable in $n$ and vector $f(n + 1)$ obtained from $f(n)$ by adding single coefficient.

Examples

- Borda $f(1) = (0)$
Pure Scoring Rules [BD10]

Purity Constraint

Pure generator $f$: p-time computable in $n$ and vector $f(n + 1)$ obtained from $f(n)$ by adding single coefficient.

Examples

Borda $f(2) = (1, 0)$
Pure Scoring Rules [BD10]

Purity Constraint

Pure generator $f$: p-time computable in $n$ and vector $f(n+1)$ obtained from $f(n)$ by adding single coefficient.

Examples

Borda $f(3) = (2, 1, 0)$
Pure Scoring Rules [BD10]

**Purity Constraint**

Pure generator $f$: p-time computable in $n$ and vector $f(n + 1)$ obtained from $f(n)$ by adding single coefficient.

**Examples**

Borda  $f(4) = (3, 2, 1, 0)$
Pure Scoring Rules [BD10]

### Purity Constraint

**Pure generator** $f$: p-time computable in $n$ and vector $f(n + 1)$ obtained from $f(n)$ by adding single coefficient.

### Examples

- **Borda** $f(5) = (4, 3, 2, 1, 0)$
Pure Scoring Rules [BD10]

**Purity Constraint**

Pure generator $f$: p-time computable in $n$ and vector $f(n + 1)$ obtained from $f(n)$ by adding single coefficient.

**Examples**

Borda $f(6) = (5, 4, 3, 2, 1, 0)$
Pure Scoring Rules [BD10]

Purity Constraint

Pure generator $f$: p-time computable in $n$ and vector $f(n+1)$ obtained from $f(n)$ by adding single coefficient.

Examples

Borda $f(7) = (6, 5, 4, 3, 2, 1, 0)$
Pure Scoring Rules [BD10]

Purity Constraint

Pure generator $f$: p-time computable in $n$ and vector $f(n + 1)$ obtained from $f(n)$ by adding single coefficient.

Examples

Borda $f(7) = (6, 5, 4, 3, 2, 1, 0)$

Thirds $f(1) = (1)$
Pure Scoring Rules [BD10]

Purity Constraint

Pure generator \( f \): p-time computable in \( n \) and vector \( f(n + 1) \) obtained from \( f(n) \) by adding single coefficient.

Examples

Borda \( f(7) = (6, 5, 4, 3, 2, 1, 0) \)

Thirds \( f(2) = (1, 0) \)
Pure Scoring Rules [BD10]

Purity Constraint

Pure generator $f$: p-time computable in $n$ and vector $f(n + 1)$ obtained from $f(n)$ by adding single coefficient.

Examples

- **Borda** $f(7) = (6, 5, 4, 3, 2, 1, 0)$
- **Thirds** $f(3) = (2, 1, 0)$
Pure Scoring Rules [BD10]

Purity Constraint

A pure generator \( f \): p-time computable in \( n \) and vector \( f(n + 1) \) obtained from \( f(n) \) by adding single coefficient.

Examples

- **Borda** \( f(7) = (6, 5, 4, 3, 2, 1, 0) \)
- **Thirds** \( f(4) = (2, 1, 0, 0) \)
Pure Scoring Rules [BD10]

Purity Constraint

Pure generator $f$: p-time computable in $n$ and vector $f(n + 1)$ obtained from $f(n)$ by adding single coefficient.

Examples

Borda $f(7) = (6, 5, 4, 3, 2, 1, 0)$

Thirds $f(5) = (2, 1, 1, 0, 0)$
Pure Scoring Rules [BD10]

Purity Constraint

Pure generator $f$: p-time computable in $n$ and vector $f(n + 1)$ obtained from $f(n)$ by adding single coefficient.

Examples

- **Borda** $f(7) = (6, 5, 4, 3, 2, 1, 0)$
- **Thirds** $f(6) = (2, 2, 1, 1, 0, 0)$
Purity Constraint

Pure generator $f$: p-time computable in $n$ and vector $f(n + 1)$ obtained from $f(n)$ by adding single coefficient.

Examples

Borda $f(7) = (6, 5, 4, 3, 2, 1, 0)$

Thirds $f(7) = (2, 2, 1, 1, 0, 0, 0)$
## Pure Scoring Rules [BD10]

### Purity Constraint

A pure generator $f$: p-time computable in $n$ and vector $f(n + 1)$ obtained from $f(n)$ by adding a single coefficient.

### Examples

<table>
<thead>
<tr>
<th>Scoring Rule</th>
<th>Example $f(n)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Borda</td>
<td>$f(7) = (6, 5, 4, 3, 2, 1, 0)$</td>
</tr>
<tr>
<td>Thirds</td>
<td>$f(8) = (2, 2, 1, 1, 1, 0, 0, 0)$</td>
</tr>
</tbody>
</table>
Pure Scoring Rules [BD10]

Purity Constraint

Pure generator $f$: p-time computable in $n$ and vector $f(n + 1)$ obtained from $f(n)$ by adding single coefficient.

Examples

- Borda $f(7) = (6, 5, 4, 3, 2, 1, 0)$
- Thirds $f(9) = (2, 2, 2, 1, 1, 1, 0, 0, 0)$
Pure Scoring Rules [BD10]

Purity Constraint

Pure generator $f$: p-time computable in $n$ and vector $f(n + 1)$ obtained from $f(n)$ by adding single coefficient.

Examples

<table>
<thead>
<tr>
<th>Method</th>
<th>Score Vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>Borda</td>
<td>(6, 5, 4, 3, 2, 1, 0)</td>
</tr>
<tr>
<td>Thirds</td>
<td>(2, 2, 2, 1, 1, 1, 0, 0, 0)</td>
</tr>
</tbody>
</table>

Pure Scoring Rules

Generated by pure generators.
Examples of Dichotomy Results for Scoring Rules

- Weighted manipulation: If “diversity of dislike,” i.e., $\alpha_2 > \alpha_n$, holds the problem is NP-complete and for all other cases the problem is in P [Hem-Hem-JCSS-2007].

- Weighted constructive control by adding (and by deleting) voters: NP-complete if there is an $i$ such that $\alpha_1 > \alpha_i > \alpha_n$ and P otherwise [Fal-Hem-Hem-JAIR-2015].

- Weighted manipulation in the case of single-peaked electorates: NP-completeness holds if (a) $n \geq 2$ and $\alpha_2 > \alpha_{\lfloor \frac{n-1}{2} \rfloor + 2}$ and there exist integers $i, j > 1$ such that $i + j \leq n + 1$ and $(\alpha_1 - \alpha_i)(\alpha_1 - \alpha_j) > (\alpha_i - \alpha_{i+1})(\alpha_j - \alpha_{j+1})$ or (b) $n \geq 2$ and $\alpha_2 = \alpha_{\lfloor \frac{n-1}{2} \rfloor + 2}$ and $\alpha_1 > \alpha_2 > \alpha_n$ and $(\alpha_2 > \alpha_{n-1}$ or $\alpha_1 - \alpha_n > 2(\alpha_2 - \alpha_n))$; and all other cases are in P [Bra-Bri-Hem-Hem-JAIR-2015].


Note these all are for the **weighted** case, and for a **fixed** number of candidates (not about PSRs). Can we get dichotomy results for the **unweighted** case? “Holy Grail!? We’ve already got one!... (or do we?).”
Dichotomy Theorem [BD10,BR12]

Theorem

The possible winner problem for $f$ is in P for $f$ ultimately equivalent to:

Triviality: $f(n) = (0, \ldots, 0)$

Plurality: $f(n) = (1, 0, \ldots, 0)$

Veto: $f(n) = (1, \ldots, 1, 0)$

For all pure $f$ not ultimately equivalent to one of these, the possible winner problem is NP-complete.
A Control Dichotomy for Pure Scoring Rules

Theorem

\( f \)-CCAV is in P for \( f \) ultimately equivalent to:

\[
\begin{align*}
\text{k-Approval, } k \leq 3: & \quad f(n) = (1, \ldots, 1, 0, \ldots, 0) \\
\text{k-Veto, } k \leq 2: & \quad f(n) = (1, \ldots, 1, 0, \ldots, 0) \\
\text{Gen. 2-App.} & \quad f(n) = (\alpha, \beta, 0, \ldots, 0) \text{ for fixed } \alpha, \beta \\
\text{Appr/Veto} & \quad f(n) = (2, 1, \ldots, 1, 0)
\end{align*}
\]

For all pure \( f \) not ultimately equivalent to one of these, \( f \)-CCAV is NP-complete.
Electoral Control

The “chair” of the election changes its structure/participants to affect the outcome.

Constructive Control by Adding Voters [BTT92]

**Problem:** \( \mathcal{E} \)-CCAV

**Input:** Registered voters \( R \), unregistered voters \( U \), preferred candidate \( p \), \( k \in \mathbb{N} \)

**Question:** Can we add at most \( k \) voters from \( U \) such that \( p \) is a winner under election system \( \mathcal{E} \)?

**Goal:** Determine the complexity of \( \mathcal{E} \)-CCAV for every \( \mathcal{E} \).
A Polynomial-Time Result

“Hybrid” between 1-Approval and 1-Veto

Generator $(2, 1, \ldots, 1, 0)$ or $(1, 0, \ldots, 0, -1)$
- Approve of one candidate
- Veto another

Proof

Preferred candidate $p$, registered voters $R$, unregistered voters $U$, $k$ voters can be added.

- Add no voter with $p$ in last place
- Add all voters with $p$ in first place
  - Greedy strategy if we can’t use all
- After this step: add only voters $c_1 > \cdots > c_2$ with $p \notin \{c_1, c_2\}$. 
A Polynomial-Time Result: \((1, 0, \ldots, 0, -1)\)

<table>
<thead>
<tr>
<th>Situation</th>
<th>Effect of Vote</th>
</tr>
</thead>
</table>
| - Only votes \(c_1 > \cdots > c_2\) for \(p \notin \{c_1, c_2\}\) | - Adding vote \(c_1 > \cdots > c_2\):  
  - adds point to \(c_1\)  
  - subtracts point from \(c_2\)  
  \(\leadsto\) “transfers” point from \(c_2\) to \(c_1\) |
| - Select at most \(k'\) of these to add | |

Edith Hemaspaandra, Lane Hemaspaandra, Henning Schnoor - A Control Dichotomy for Pure Scoring Rules
A Polynomial-Time Result: \((1, 0, \ldots, 0, -1)\)

<table>
<thead>
<tr>
<th>Situation</th>
<th>Effect of Vote</th>
</tr>
</thead>
<tbody>
<tr>
<td>Only votes (c_1 &gt; \cdots &gt; c_2) for (p \notin {c_1, c_2})</td>
<td>Adding vote (c_1 &gt; \cdots &gt; c_2):</td>
</tr>
<tr>
<td>Select at most (k') of these to add</td>
<td>* adds point to (c_1)</td>
</tr>
<tr>
<td></td>
<td>* subtracts point from (c_2)</td>
</tr>
<tr>
<td></td>
<td>(\Rightarrow) “transfers” point from (c_2) to (c_1)</td>
</tr>
</tbody>
</table>

**Approach**

- Vote additions “transfer” points between candidates
- Score of \(p\) is fixed (after preprocessing)
- **Goal:** For each \(c_i \neq p\), shift surplus points away
A Polynomial-Time Result: (1, 0, \ldots, 0, -1)

### Situation
- Only votes $c_1 > \cdots > c_2$ for $p \not\in \{c_1, c_2\}$
- Select at most $k'$ of these to add

### Effect of Vote
Adding vote $c_1 > \cdots > c_2$:
- adds point to $c_1$
- subtracts point from $c_2$
\[ \Rightarrow \text{“transfers” point from } c_2 \text{ to } c_1 \]

### Approach
- Vote additions “transfer” points between candidates
- Score of $p$ is fixed (after preprocessing)
- **Goal**: For each $c_i \neq p$, shift surplus points away

### Solution
Min-cost network flow!
Polynomial Time via Network Flow: \((1, 0, \ldots, 0, -1)\)

**Network Setup**

- one node for each \(c_i \neq p\)
- source, target nodes \(S\) and \(T\)
Polynomial Time via Network Flow: $(1, 0, \ldots, 0, -1)$

**Network Setup**
- one node for each $c_i \neq p$
- source, target nodes $S$ and $T$

**Flows in Network**
- each $c_i$: gets $\text{score}(c_i)$ points
Polynomial Time via Network Flow: $(1, 0, \ldots, 0, -1)$

**Network Setup**
- one node for each $c_i \neq p$
- source, target nodes $S$ and $T$

**Flows in Network**
- each $c_i$: gets $\text{score}(c_i)$ points
- each potential vote allows to “move” one point
  - This costs one “unit”
Polynomial Time via Network Flow: \((1, 0, \ldots, 0, -1)\)

**Network Setup**
- one node for each \(c_i \neq p\)
- source, target nodes \(S\) and \(T\)

**Flows in Network**
- each \(c_i\): gets \(score(c_i)\) points
- each potential vote allows to “move” one point
  - This costs one “unit”
- each \(c_i \neq p\): at most \(score(p)\) points in the end
Polynomial Time via Network Flow

Votes

Graph

- $c_1$, $c_2$, $c_3$, $c_4$, $c_5$, $c_6$, $c_7$
- $S$, $T$

Construction Votes: Allow to "move" points

Each $c_i$: points from votes

In the end: $\leq$ score ($p$) points

Control with $\leq k'$ voters (in this post-pre-processing version) $\iff$ flow with cost $\leq k'$, flow-amount $\sum c_i \neq p$ score ($c_i$) exists.
Polynomial Time via Network Flow

VOTES

- $c_5 > \cdots > c_1$

CONSTRUCTION

- Votes: Allow to “move” points

Graph

Construction

- Votes: Allow to “move” points

Graph

- Votes: Allow to “move” points
Polynomial Time via Network Flow

**Votes**
- \( c_5 > \cdots > c_1 \)
- \( c_3 > \cdots > c_1 \)

**Construction**
- Votes: Allow to “move” points

**Graph**

![Graph diagram with nodes labeled S, c_1, c_2, c_3, c_4, c_5, c_6, c_7, and T connected by arrows indicating flow with cost 1.](image)

Score functions:
- \( \text{score}(c_5) \)
- \( \text{score}(c_3) \)
- \( \text{score}(c_1) \)
- \( \text{score}(c_2) \)
- \( \text{score}(c_4) \)
- \( \text{score}(c_6) \)
- \( \text{score}(c_7) \)
Polynomial Time via Network Flow

Voters

- \( c_5 > \cdots > c_1 \)
- \( c_3 > \cdots > c_1 \)
- \( c_7 > \cdots > c_4 \)

Graph

- Construction
  - Votes: Allow to “move” points
**Votes**
- $c_5 > \cdots > c_1$
- $c_3 > \cdots > c_1$
- $c_7 > \cdots > c_4$
- $c_2 > \cdots > c_4$

**Graph**

- Construction
  - Votes: Allow to “move” points
Polynomial Time via Network Flow

Voting
- $c_5 > \cdots > c_1$
- $c_3 > \cdots > c_1$
- $c_7 > \cdots > c_4$
- $c_2 > \cdots > c_4$
- $c_2 > \cdots > c_6$

Graph

Construction
- Votes: Allow to "move" points
Polynomial Time via Network Flow

**Votes**
- $c_5 > \cdots > c_1$
- $c_3 > \cdots > c_1$
- $c_7 > \cdots > c_4$
- $c_2 > \cdots > c_4$
- $c_2 > \cdots > c_6$
- $c_5 > \cdots > c_6$

**Graph**

**Construction**
- Votes: Allow to “move” points
Polynomial Time via Network Flow

**Votes**
- $c_5 > \cdots > c_1$
- $c_3 > \cdots > c_1$
- $c_7 > \cdots > c_4$
- $c_2 > \cdots > c_4$
- $c_2 > \cdots > c_6$
- $c_5 > \cdots > c_6$
- $c_1 > \cdots > c_2$

**Graph**

![Graph Diagram]

**Construction**
- Votes: Allow to “move” points
Polynomial Time via Network Flow

**Votes**
- $c_5 > \cdots > c_1$
- $c_3 > \cdots > c_1$
- $c_7 > \cdots > c_4$
- $c_2 > \cdots > c_4$
- $c_2 > \cdots > c_6$
- $c_5 > \cdots > c_6$
- $c_1 > \cdots > c_2$
- $c_3 > \cdots > c_2$

**Graph**

**Construction**
- Votes: Allow to “move” points
Polynomial Time via Network Flow

**Votes**

- $c_5 > \cdots > c_1$
- $c_3 > \cdots > c_1$
- $c_7 > \cdots > c_4$
- $c_2 > \cdots > c_4$
- $c_2 > \cdots > c_6$
- $c_5 > \cdots > c_6$
- $c_1 > \cdots > c_2$
- $c_3 > \cdots > c_2$
- $c_5 > \cdots > c_6$

**Construction**

- Votes: Allow to “move” points

**Graph**

![Graph with nodes S, C1, C2, C3, C4, C5, C6, C7, and T connected by directed edges with labels 1 and 2. The edge from C1 to C2 has cost 1.]

**Score**

- $\text{score}(c_5)$
- $\text{score}(c_3)$
- $\text{score}(c_1)$
- $\text{score}(c_2)$
- $\text{score}(c_4)$
- $\text{score}(c_6)$

- $\text{score}(p)$
- $\text{score}(p)$
- $\text{score}(p)$
- $\text{score}(p)$

- Cost 1
Polynomial Time via Network Flow

**Votes**
- $c_5 > \cdots > c_1$
- $c_3 > \cdots > c_1$
- $c_7 > \cdots > c_4$
- $c_2 > \cdots > c_4$
- $c_2 > \cdots > c_6$
- $c_5 > \cdots > c_6$
- $c_1 > \cdots > c_2$
- $c_3 > \cdots > c_2$
- $c_5 > \cdots > c_6$
- $c_3 > \cdots > c_2$

**Graph**

**Construction**
- Votes: Allow to “move” points
Polynomial Time via Network Flow

**Votes**
- $c_5 > \cdots > c_1$
- $c_3 > \cdots > c_1$
- $c_7 > \cdots > c_4$
- $c_2 > \cdots > c_4$
- $c_2 > \cdots > c_6$
- $c_5 > \cdots > c_6$
- $c_1 > \cdots > c_2$
- $c_3 > \cdots > c_2$
- $c_5 > \cdots > c_6$
- $c_3 > \cdots > c_2$
- $c_5 > \cdots > c_6$

**Graph**

**Construction**
- Votes: Allow to “move” points
Polynomial Time via Network Flow

Votes
- $c_5 > \cdots > c_1$
- $c_3 > \cdots > c_1$
- $c_7 > \cdots > c_4$
- $c_2 > \cdots > c_4$
- $c_2 > \cdots > c_6$
- $c_5 > \cdots > c_6$
- $c_1 > \cdots > c_2$
- $c_3 > \cdots > c_2$
- $c_5 > \cdots > c_6$
- $c_3 > \cdots > c_2$
- $c_5 > \cdots > c_6$

Graph

Construction
- Votes: Allow to “move” points
- Each $c_i$: points from votes

In the end: $\leq \text{score}(p)$ points

Control with $\leq k'v$ voters (in this post-pre-processing version) $\iff$ flow with cost $\leq k'$, flow-amount $\sum_{c_i \neq p} \text{score}(c_i)$ exists.
Polynomial Time via Network Flow

Votes

- \(c_5 > \cdots > c_1\)
- \(c_3 > \cdots > c_1\)
- \(c_7 > \cdots > c_4\)
- \(c_2 > \cdots > c_4\)
- \(c_2 > \cdots > c_6\)
- \(c_5 > \cdots > c_6\)
- \(c_1 > \cdots > c_2\)
- \(c_3 > \cdots > c_2\)
- \(c_5 > \cdots > c_6\)
- \(c_3 > \cdots > c_2\)
- \(c_5 > \cdots > c_6\)

Graph

- \(score(c_3)\)
- \(score(c_5)\)
- \(c_1\)
- \(c_2\)
- \(c_3\)
- \(c_4\)
- \(c_5\)
- \(c_6\)
- \(c_7\)
- \(S\)
- \(T\)

Construction

- Votes: Allow to “move” points
- Each \(c_i\): points from votes

In the end: \(\leq\) \(score(p)\) points

Control with \(\leq k'\) voters (in this post-pre-processing version) ⇔ flow with cost \(\leq k'\), flow-amount \(\sum c_i \neq p\) exists.
### Construction

- **Votes:** Allow to “move” points
- **Each** $c_i$: points from votes

### Graph

- **$c_5 > \cdots > c_1$**
- **$c_3 > \cdots > c_1$**
- **$c_7 > \cdots > c_4$**
- **$c_2 > \cdots > c_4$**
- **$c_2 > \cdots > c_6$**
- **$c_5 > \cdots > c_6$**
- **$c_1 > \cdots > c_2$**
- **$c_3 > \cdots > c_2$**
- **$c_5 > \cdots > c_6$**
- **$c_3 > \cdots > c_2$**
- **$c_5 > \cdots > c_6$**

In the end: $\leq \text{score}(c_i)$ points

Control with $\leq k'v$ voters (in this post-pre-processing version) $\iff$ flow with cost $\leq k'$, flow-amount $\sum c_i \neq p$ exists.
Polynomial Time via Network Flow

**Votes**
- $c_5 > \cdots > c_1$
- $c_3 > \cdots > c_1$
- $c_7 > \cdots > c_4$
- $c_2 > \cdots > c_4$
- $c_2 > \cdots > c_6$
- $c_5 > \cdots > c_6$
- $c_1 > \cdots > c_2$
- $c_3 > \cdots > c_2$
- $c_5 > \cdots > c_6$
- $c_3 > \cdots > c_2$
- $c_5 > \cdots > c_6$

**Construction**
- Votes: Allow to “move” points
- Each $c_i$: points from votes

**Graph**

In the end: $\leq \text{score}(p)$ points

Control with $\leq k'$ voters (in this post-pre-processing version) $\iff$ flow with cost $\leq k'$, flow-amount $\sum c_i \neq p$ score($c_i$) exists.
Polynomial Time via Network Flow

**Votes**
- $c_5 > \ldots > c_1$
- $c_3 > \ldots > c_1$
- $c_7 > \ldots > c_4$
- $c_2 > \ldots > c_4$
- $c_2 > \ldots > c_6$
- $c_5 > \ldots > c_6$
- $c_1 > \ldots > c_2$
- $c_3 > \ldots > c_2$
- $c_5 > \ldots > c_6$
- $c_3 > \ldots > c_2$
- $c_5 > \ldots > c_6$

**Graph**

**Construction**
- Votes: Allow to “move” points
- Each $c_i$: points from votes
- In the end: $\leq \text{score}(p)$ points
Polynomial Time via Network Flow

**Votes**
- $c_5 > \cdots > c_1$
- $c_3 > \cdots > c_1$
- $c_7 > \cdots > c_4$
- $c_2 > \cdots > c_4$
- $c_2 > \cdots > c_6$
- $c_5 > \cdots > c_6$
- $c_1 > \cdots > c_2$
- $c_3 > \cdots > c_2$
- $c_5 > \cdots > c_6$
- $c_3 > \cdots > c_2$
- $c_5 > \cdots > c_6$

**Graph**

**Construction**
- Votes: Allow to “move” points
- Each $c_i$: points from votes
- In the end: $\leq \text{score}(p)$ points
Polynomial Time via Network Flow

**Votes**
- $c_5 > \cdots > c_1$
- $c_3 > \cdots > c_1$
- $c_7 > \cdots > c_4$
- $c_2 > \cdots > c_4$
- $c_2 > \cdots > c_6$
- $c_5 > \cdots > c_6$
- $c_1 > \cdots > c_2$
- $c_3 > \cdots > c_2$
- $c_5 > \cdots > c_6$
- $c_3 > \cdots > c_2$
- $c_5 > \cdots > c_6$

**Graph**

**Construction**
- Votes: Allow to “move” points
- Each $c_i$: points from votes
- In the end: $\leq \text{score}(p)$ points
Polynomial Time via Network Flow

**Votes**
- $c_5 > \cdots > c_1$
- $c_3 > \cdots > c_1$
- $c_7 > \cdots > c_4$
- $c_2 > \cdots > c_4$
- $c_2 > \cdots > c_6$
- $c_5 > \cdots > c_6$
- $c_3 > \cdots > c_2$
- $c_1 > \cdots > c_2$
- $c_5 > \cdots > c_6$
- $c_3 > \cdots > c_2$
- $c_5 > \cdots > c_6$

**Construction**
- Votes: Allow to “move” points
- Each $c_i$: points from votes
- In the end: $\leq \text{score}(p)$ points

**Graph**
- Edith Hemaspaandra, Lane Hemaspaandra, Henning Schnoor - A Control Dichotomy for Pure Scoring Rules

---

**Control with** $\leq k'v$ voters (in this post-pre-processing version) $\iff$ flow with cost $\leq k'$, flow-amount $\sum c_i \neq p$ score exist.
Polynomial Time via Network Flow

**Votes**
- $c_5 > \cdots > c_1$
- $c_3 > \cdots > c_1$
- $c_7 > \cdots > c_4$
- $c_2 > \cdots > c_4$
- $c_2 > \cdots > c_6$
- $c_5 > \cdots > c_6$
- $c_1 > \cdots > c_2$
- $c_3 > \cdots > c_2$
- $c_5 > \cdots > c_6$
- $c_3 > \cdots > c_2$
- $c_5 > \cdots > c_6$

**Graph**

**Construction**
- Votes: Allow to “move” points
- Each $c_i$: points from votes
- In the end: $\leq \text{score}(p)$ points

Control with $\leq k'$ voters (in this post-pre-processing version) $\iff$ flow with cost $\leq k'$, flow-amount $\sum_{c_i \not= p} \text{score}(c_i)$ exists.
A Control Dichotomy for Pure Scoring Rules

**Theorem**

\( f \)-CCAV is in P for \( f \) ultimately equivalent to:

- **k-Approval**, \( k \leq 3 \): \( f(n) = (1, \ldots, 1, 0, \ldots, 0) \)
- **k-Veto**, \( k \leq 2 \): \( f(n) = (1, \ldots, 1, 0, \ldots, 0) \)

**Gen. 2-App.** \( f(n) = (\alpha, \beta, 0, \ldots, 0) \) for fixed \( \alpha, \beta \)

**Appr/Veto** \( f(n) = (2, 1, \ldots, 1, 0) \)

For all pure \( f \) not ultimately equivalent to one of these, \( f \)-CCAV is NP-complete.
Hard Cases: Case Distinction (these are the $\alpha_4 = \alpha_{n-2}$ cases)

**Theorem**

$f$-CCAV is NP-complete for

- $(\alpha, \beta, \gamma, 0, \ldots, 0)$ for $\alpha \geq \beta \geq \gamma > 0, \alpha \neq \gamma$.
- $(\alpha_1, \alpha_2, \alpha_3, \alpha_4, \ldots, \alpha_4, \alpha_5, 0)$ with $\alpha_2 > \alpha_4 > 0$.
- $(\alpha_1, \alpha_2, \ldots, \alpha_2, 0)$ with $\alpha_1 \notin \{\alpha_2, 2\alpha_2\}, \alpha_2 > 0$.
- $(\alpha_1, \alpha_2, \ldots, \alpha_2, \alpha_5, 0)$ with $\alpha_1 > \alpha_2 > \alpha_5$.
- $(\alpha_1, \ldots, \alpha_1, \alpha_5, 0)$ with $\alpha_1 > \alpha_5 > 0$. 

Corollary (Cor. of the theorem at the left plus some other cases that come from results related to the next slide)

For a pure generator $f$ with $f$ is not ultimately equivalent to a generator in the P list, $f$ uses at most 6 different coefficients.
Hard Cases: Case Distinction (these are the $\alpha_4 = \alpha_{n-2}$ cases)

**Theorem**

$f$-CCAV is NP-complete for

- $(\alpha, \beta, \gamma, 0, \ldots, 0)$ for $\alpha \geq \beta \geq \gamma > 0$, $\alpha \neq \gamma$.
- $(\alpha_1, \alpha_2, \alpha_3, \alpha_4, \ldots, \alpha_4, \alpha_5, 0)$ with $\alpha_2 > \alpha_4 > 0$.
- $(\alpha_1, \alpha_2, \ldots, \alpha_2, 0)$ with $\alpha_1 \notin \{\alpha_2, 2\alpha_2\}$, $\alpha_2 > 0$.
- $(\alpha_1, \alpha_2, \ldots, \alpha_2, \alpha_5, 0)$ with $\alpha_1 > \alpha_2 > \alpha_5$.
- $(\alpha_1, \ldots, \alpha_1, \alpha_5, 0)$ with $\alpha_1 > \alpha_5 > 0$.

**Corollary**

(Cor. of the theorem at the left plus some other cases that come from results related to the next slide) For a pure generator $f$ with

- $f$ is not ultimately equivalent to a generator in the P list,
- $f$ uses at most 6 different coefficients

$f$-CCAV is NP-complete.

Direct reductions (point counting and turning of 3DM into election-like instances).
What happens with more than six values?

Analysis

\((\alpha_1, \alpha_2, \alpha_3, \alpha_4, \ldots, \alpha_{n-2}, \alpha_{n-1}, \alpha_n)\)

- seven values: \(\alpha_4 \neq \alpha_{n-2}\)
- top four positions “better” than lowest three.
- (middle positions can be “good” or “bad.”)
- Like 4-approval or like 3-veto.
Conclusion

Contributions

- Dichotomy theorem for CCAV for PSRs.
Conclusion

Contributions

- Dichotomy theorem for CCAV for PSRs.
- Descriptive richness of different PSR definitions.
Conclusion

Contributions

- Dichotomy theorem for CCAV for PSRs.
- Descriptive richness of different PSR definitions.

Open Questions

- Complexity of CCAV for other voting systems?
Conclusion

Contributions

- Dichotomy theorem for CCAV for PSRs.
- Descriptive richness of different PSR definitions.

Open Questions

- Complexity of CCAV for other voting systems?
- Complexity of related problems
  - control by deleting voters
Conclusion

Contributions
- Dichotomy theorem for CCAV for PSRs.
- Descriptive richness of different PSR definitions.

Open Questions
- Complexity of CCAV for other voting systems?
- Complexity of related problems
  - control by deleting voters
  - bribery
Conclusion

Contributions

- Dichotomy theorem for CCAV for PSRs.
- Descriptive richness of different PSR definitions.

Open Questions

- Complexity of CCAV for other voting systems?
- Complexity of related problems
  - control by deleting voters
  - bribery
  - manipulation
Conclusion

Contributions
- Dichotomy theorem for CCAV for PSRs.
- Descriptive richness of different PSR definitions.

Open Questions
- Complexity of CCAV for other voting systems?
- Complexity of related problems
  - control by deleting voters
  - bribery
  - manipulation

Thank You!
A Control Dichotomy for Pure Scoring Rules

Edith Hemaspaandra, Lane Hemaspaandra, Henning Schnoor

(The TR version is in arXiv.org and a conference version is in AAAI-14.)
Overview

1. Introduction: Elections and Control
2. The Complexity of the Control Problem
3. A Class of Election Systems: Scoring Rules
4. The Price of Purity
5. A Dichotomy Theorem
   - Polynomial Time Results
   - Hardness Results
6. Conclusion
Overview

1. Introduction: Elections and Control
2. The Complexity of the Control Problem
3. A Class of Election Systems: Scoring Rules
4. The Price of Purity
5. A Dichotomy Theorem
   - Polynomial Time Results
   - Hardness Results
6. Conclusion
Elections

**Election**

**Voters** elect **candidate** using an **election system**
Election

Voters elect candidate using an election system

Which system is “good”? 

- Political parliament
- Web page ranking
- Agreement on a single plan
Elections

**Election**

**Voters** elect **candidate** using an **election system**

**Which system is “good”?**

Depends on the situation!

- Political parliament
- Web page ranking
- Agreement on a single plan

**Common Criteria**

Winners are those that are “liked most” by the voters.
Controlling an Election

Election Idea

- Voters vote sincerely
- votes are counted correctly
- result reflects preferences of the majority
Controlling an Election

Election Idea
- Voters vote sincerely
- votes are counted correctly
- result reflects preferences of the majority

Practice
Involved parties can be dishonest...
- Voters can vote dishonestly

strategic voting
Controlling an Election

Election Idea

- Voters vote sincerely
- votes are counted correctly
- result reflects preferences of the majority

Practice

Involved parties can be dishonest . . .

- Voters can vote dishonestly
- counting can go wrong

strategic voting
manipulation of algorithms
Controlling an Election

Election Idea

- Voters vote sincerely
- votes are counted correctly
- result reflects preferences of the majority

Practice

Involved parties can be disonest . . .

- Voters can vote dishonestly
- counting can go wrong
- election chair can control the election:
  - add / delete candidates or voters
  - set up voting districts
Controlling an Election

Election Idea
- Voters vote sincerely
- votes are counted correctly
- result reflects preferences of the majority

Practice
Involved parties can be dishonest . . .
- Voters can vote dishonestly
- counting can go wrong
- election chair can control the election:
  - add / delete candidates or voters
  - set up voting districts

strategic voting
manipulation of algorithms
← this talk
Control by Adding Voters

Observation

Easier to change **whether** someone votes than **whom** she votes.

- voter turnout depends on weather
- bad weather good for CDU
Control by Adding Voters

Observation

Easier to change **whether** someone votes than **whom** she votes.

- voter turnout depends on weather
- bad weather good for CDU
Control by Adding Voters

Observation
Easier to change whether someone votes than whom she votes.
- voter turnout depends on weather
- bad weather good for CDU

Assumption

Situation
- Voters $R$ decided to vote
- Voters $U$ undecided whether they will
- Every voter knows how she would vote
**Control by Adding Voters**

**Observation**

Easier to change *whether* someone votes than *whom* she votes.
- voter turnout depends on weather
- bad weather good for CDU

**Assumption**

**Situation**
- Voters $R$ decided to vote
- Voters $U$ undecided whether they will
- Every voter knows *how* she would vote

**Election authority (chair)**
- knows preferences of each voter in $R$ and $U$
- has resources to make $k$ people in $U$ vote
Overview

1 Introduction: Elections and Control

2 The Complexity of the Control Problem

3 A Class of Election Systems: Scoring Rules

4 The Price of Purity

5 A Dichotomy Theorem
   • Polynomial Time Results
   • Hardness Results

6 Conclusion
Control as a Computational Problem

Election Chair Point of View

**Knows:**
- voting system
- candidates
- votes of *registered voters*
- votes of *potential voters*
- cost limit $k$

Complexity of Control Problem

PTIME-solvable Chair can exercise control in practice

NP-hard voting system is computationally resistant

Edith Hemaspaandra, Lane Hemaspaandra, Henning Schnoor - A Control Dichotomy for Pure Scoring Rules
Control as a Computational Problem

Election Chair Point of View

Knows:
- voting system
- candidates
- votes of registered voters
- votes of potential voters
- cost limit $k$

Question:
Which $k$ potential voters should she add?
Control as a Computational Problem

Election Chair Point of View

Knows:
- voting system
- candidates
- votes of registered voters
- votes of potential voters
- cost limit $k$

Question:
Which $k$ potential voters should she add?

Complexity of Control Problem

PTIME-solvable Chair can exercise control in practice
Control as a Computational Problem

**Election Chair Point of View**

**Knows:**
- voting system
- candidates
- votes of **registered voters**
- votes of **potential voters**
- cost limit $k$

**Question:**
Which $k$ potential voters should she add?

**Complexity of Control Problem**

- **PTIME-solvable** Chair can exercise control in practice
- **NP-hard** voting system is **computationally resistant** to control
Constructive Control by Adding Voters

**CCAV**

**Problem:** CCAV

**Input:** Voters \( R \), potential voters \( U \), candidate \( p \), \( k \in \mathbb{N} \)

**Question:** Can we add at most \( k \) votes from \( U \) such that \( p \) is a winner of the election?
Constructive Control by Adding Voters

**CCA\(V\)**

- **Problem:** C\(C\)AV
- **Input:** Voters \(R\), potential voters \(U\), candidate \(p\), \(k \in \mathbb{N}\)
- **Question:** Can we add at most \(k\) votes from \(U\) such that \(p\) is a winner of the election?

Each election system gives different CCA\(V\) problem!
## Constructive Control by Adding Voters

### CCAV

**Problem:** $\mathcal{E}$-CCAV

**Input:** Voters $R$, potential voters $U$, candidate $p$, $k \in \mathbb{N}$

**Question:** Can we add at most $k$ votes from $U$ such that $p$ is a winner of the election when election system $\mathcal{E}$ is used?

Each election system gives different CCAV problem!
Constructive Control by Adding Voters

**CCAV**

**Problem:** $\mathcal{E}$-CCAV

**Input:** Voters $R$, potential voters $U$, candidate $p$, $k \in \mathbb{N}$

**Question:** Can we add at most $k$ votes from $U$ such that $p$ is a winner of the election when election system $\mathcal{E}$ is used?

Each election system gives different CCAV problem!

**Hope**

If $\mathcal{E}$-CCAV is NP-hard, the chair cannot control $\mathcal{E}$-elections in practice.

**Research Question**

Given $\mathcal{E}$, what is the complexity of $\mathcal{E}$-CCAV?
Overview

1. Introduction: Elections and Control

2. The Complexity of the Control Problem

3. A Class of Election Systems: Scoring Rules

4. The Price of Purity

5. A Dichotomy Theorem
   - Polynomial Time Results
   - Hardness Results

6. Conclusion
Towards a Dichotomy

Goal: Complexity of every $\mathcal{E}$-CCAV

- What is an election system?
- How can we find uniform description of election systems?
Towards a Dichotomy

Goal: Complexity of every $\mathcal{E}$-CCAV

- **What is an election system?**
- **How can we find uniform description of election systems?**

**Election System**

- **Input** Set $C$ of candidates, multiset $V$ of votes (orders on $C$)
- **Output** Set $W \subseteq C$ of **winners** (tie-breaking may be needed)
Towards a Dichotomy

**Goal: Complexity of every $\mathcal{E}$-CCAV**

- What is an election system?
- **How can we find uniform description of election systems?**

**Election System**

*Input* Set $C$ of candidates, multiset $V$ of votes (orders on $C$)

*Output* Set $W \subseteq C$ of **winners** (tie-breaking may be needed)

**Characterization of $\mathcal{E}$-CCAV**

Large class of $\mathcal{E}$ in common framework: **Scoring rules**.
Example Election Systems

**Plurality**

- each voter selects a candidate
- the most-often selected candidate wins
## Example Election Systems

<table>
<thead>
<tr>
<th>Plurality</th>
<th>$k$-Approval</th>
</tr>
</thead>
<tbody>
<tr>
<td>- each voter selects a candidate</td>
<td>- Same, but each voter selects $k$ candidates.</td>
</tr>
<tr>
<td>- the most-often selected candidate wins</td>
<td></td>
</tr>
</tbody>
</table>
Example Election Systems

**Plurality**
- each voter selects a candidate
- the most-often selected candidate wins

**k-Approval**
- Same, but each voter selects $k$ candidates.

**k-Veto**
- each voter chooses $k$ candidates
- least-often selected candidates win
Example Election Systems

<table>
<thead>
<tr>
<th>Plurality</th>
<th>$k$-Approval</th>
</tr>
</thead>
<tbody>
<tr>
<td>- each voter selects a candidate</td>
<td>- Same, but each voter selects $k$ candidates.</td>
</tr>
<tr>
<td>- the most-often selected candidate wins</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$k$-Veto</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>- each voter chooses $k$ candidates</td>
<td></td>
</tr>
<tr>
<td>- least-often selected candidates win</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Borda Count</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>- voters rank $n$ candidates best to worst</td>
<td></td>
</tr>
<tr>
<td>- $i$-th candidate gets $n - i$ points</td>
<td></td>
</tr>
</tbody>
</table>
## Example Election Systems

<table>
<thead>
<tr>
<th>Plurality</th>
<th>$k$-Approval</th>
</tr>
</thead>
<tbody>
<tr>
<td>- each voter selects a candidate</td>
<td>Same, but each voter selects $k$ candidates.</td>
</tr>
<tr>
<td>- the most-often selected candidate wins</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$k$-Veto</th>
</tr>
</thead>
<tbody>
<tr>
<td>- each voter chooses $k$ candidates</td>
</tr>
<tr>
<td>- least-often selected candidates win</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Borda Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>- voters rank $n$ candidates best to worst</td>
</tr>
<tr>
<td>- $i$-th candidate gets $n - i$ points</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dowdall</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Same, but with points $1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \frac{1}{5}, \ldots$</td>
</tr>
</tbody>
</table>

Edith Hemaspaandra, Lane Hemaspaandra, Henning Schnoor - A Control Dichotomy for Pure Scoring Rules
### Generalization

**Scoring Rules**

$\alpha = (\alpha_1, \ldots, \alpha_n)$ defines **scoring rule**:

- voters ranks candidates best to worst
- $i$-th position $i$ gets $\alpha_i$ points
- candidates with most points win
Generalization

**Scoring Rules**

\[ \alpha = (\alpha_1, \ldots, \alpha_n) \] defines **scoring rule**:

- voters ranks candidates best to worst
- \( i \)-th position \( i \) gets \( \alpha_i \) points
- candidates with most points win

**Examples**

- \( k \)-approval: \( \alpha = (1, \ldots, 1, 0, \ldots, 0) \)
- Dowdall: \( (1, \frac{1}{2}, \frac{1}{3}, \ldots, \frac{1}{n}) \)
- Borda: \( (n-1, n-2, \ldots, 1, 0) \)
Generalization

Scoring Rules

$\alpha = (\alpha_1, \ldots, \alpha_n)$ defines **scoring rule**:
- voters ranks candidates best to worst
- $i$-th position $i$ gets $\alpha_i$ points
- candidates with most points win

Examples

- $k$-approval: $\alpha = (1, \ldots, 1, 0, \ldots, 0)$
- $k$-veto: $\alpha = (1, 1, 1, 0, \ldots, 0)$

Schreiben Sie Platz- oder Rangnummern (1, 2, ...) neben einige oder alle Namen.

- [ ] Max Muster
- [1] Tanja Tüchtig
- [3] Hans Schmidt
- [ ] Markus Möglich
- [2] Maria Hügel
Generalization

Scoring Rules

$\alpha = (\alpha_1, \ldots, \alpha_n)$ defines scoring rule:
- voters ranks candidates best to worst
- $i$-th position $i$ gets $\alpha_i$ points
- candidates with most points win

Examples

- $k$-approval: $\alpha = (1, \ldots, 1, 0, \ldots, 0)$
- $k$-veto: $\alpha = (1, 1, 1, 0, \ldots, 0)$
- Borda: $(n-1, n-2, \ldots, 1, 0)$
Generalization

Scoring Rules

\( \alpha = (\alpha_1, \ldots, \alpha_n) \) defines **scoring rule**:

- voters ranks candidates best to worst
- \( i \)-th position \( i \) gets \( \alpha_i \) points
- candidates with most points win

Examples

- **k**-approval: \( \alpha = (1, \ldots, 1, 0, \ldots, 0) \)
- **k**-veto: \( \alpha = (1, 1, 1, 0, \ldots, 0) \)
- Borda: \( (n - 1, n - 2, \ldots, 1, 0) \)
- Dowdall: \( (1, \frac{1}{2}, \frac{1}{3}, \ldots, \frac{1}{n}) \)
Scoring Rules: An Example

Election System

- 6 candidates
- \( \alpha = (17, 9, 5, 2, 2, -1) \)

Example Vote

<table>
<thead>
<tr>
<th></th>
<th>Alice:</th>
<th>Bob:</th>
<th>Charlie:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( c_3 )</td>
<td>( c_2 )</td>
<td>( c_4 )</td>
</tr>
<tr>
<td>( c_2 )</td>
<td>&gt;</td>
<td>&gt;</td>
<td>&gt;</td>
</tr>
<tr>
<td>( c_3 )</td>
<td>&gt;</td>
<td>&gt;</td>
<td>&gt;</td>
</tr>
<tr>
<td>( c_1 )</td>
<td>&gt;</td>
<td>&gt;</td>
<td>&gt;</td>
</tr>
<tr>
<td>( c_5 )</td>
<td>&gt;</td>
<td>&gt;</td>
<td>&gt;</td>
</tr>
<tr>
<td>( c_4 )</td>
<td>&gt;</td>
<td>&gt;</td>
<td>&gt;</td>
</tr>
<tr>
<td>( c_6 )</td>
<td>&gt;</td>
<td>&gt;</td>
<td>&gt;</td>
</tr>
<tr>
<td>( c_1 )</td>
<td>&gt;</td>
<td>&gt;</td>
<td>&gt;</td>
</tr>
<tr>
<td>( c_2 )</td>
<td>&gt;</td>
<td>&gt;</td>
<td>&gt;</td>
</tr>
<tr>
<td>( c_3 )</td>
<td>&gt;</td>
<td>&gt;</td>
<td>&gt;</td>
</tr>
<tr>
<td>( c_5 )</td>
<td>&gt;</td>
<td>&gt;</td>
<td>&gt;</td>
</tr>
</tbody>
</table>

Count

\[
\begin{align*}
\text{c}_1 & = 2 + 5 + 9 = 16 \\
\text{c}_2 & = 9 + 17 + 2 = 28 \\
\text{c}_3 & = 17 + 9 + 2 = 28 \\
\text{c}_4 & = -1 + 2 + 17 = 18 \\
\text{c}_5 & = 2 + 2 - 1 = 1 \\
\text{c}_6 & = 5 - 1 + 5 = 9
\end{align*}
\]

\( \text{c}_2 \) and \( \text{c}_3 \) win the election.

More Candidates?

This "system" only works for 6 candidates.
Scoring Rules: An Example

**Election System**
- 6 candidates
- $\alpha = (17, 9, 5, 2, 2, -1)$

**Example Vote**

<table>
<thead>
<tr>
<th></th>
<th>17</th>
<th>9</th>
<th>5</th>
<th>2</th>
<th>2</th>
<th>-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>c3</td>
<td>&gt;</td>
<td>c2</td>
<td>&gt;</td>
<td>c6</td>
<td>&gt;</td>
</tr>
<tr>
<td>Bob</td>
<td>c2</td>
<td>&gt;</td>
<td>c3</td>
<td>&gt;</td>
<td>c1</td>
<td>&gt;</td>
</tr>
<tr>
<td>Charlie</td>
<td>c4</td>
<td>&gt;</td>
<td>c1</td>
<td>&gt;</td>
<td>c6</td>
<td>&gt;</td>
</tr>
</tbody>
</table>

$c_2$ and $c_3$ win the election.

More Candidates?
This “system” only works for 6 candidates.
Scoring Rules: An Example

**Election System**
- 6 candidates
- $\alpha = (17, 9, 5, 2, 2, -1)$

**Example Vote**

<table>
<thead>
<tr>
<th></th>
<th>17</th>
<th>9</th>
<th>5</th>
<th>2</th>
<th>2</th>
<th>-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>$c_3$ &gt; $c_2$ &gt; $c_6$ &gt; $c_1$ &gt; $c_5$ &gt; $c_4$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bob</td>
<td>$c_2$ &gt; $c_3$ &gt; $c_1$ &gt; $c_5$ &gt; $c_4$ &gt; $c_6$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Charlie</td>
<td>$c_4$ &gt; $c_1$ &gt; $c_6$ &gt; $c_2$ &gt; $c_3$ &gt; $c_5$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Count**

$c_1: 2 + 5 + 9 = 16$
Scoring Rules: An Example

Election System
- 6 candidates
- $\alpha = (17, 9, 5, 2, 2, -1)$

Example Vote

<table>
<thead>
<tr>
<th>Candidate</th>
<th>Alice</th>
<th>Bob</th>
<th>Charlie</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$c_3$</td>
<td>$c_2$</td>
<td>$c_4$</td>
</tr>
<tr>
<td>17</td>
<td>$&gt;$</td>
<td>$&gt;$</td>
<td>$&gt;$</td>
</tr>
<tr>
<td>9</td>
<td>$&gt;$</td>
<td>$&gt;$</td>
<td>$&gt;$</td>
</tr>
<tr>
<td>5</td>
<td>$&gt;$</td>
<td>$&gt;$</td>
<td>$&gt;$</td>
</tr>
<tr>
<td>2</td>
<td>$&gt;$</td>
<td>$&gt;$</td>
<td>$&gt;$</td>
</tr>
<tr>
<td>2</td>
<td>$&gt;$</td>
<td>$&gt;$</td>
<td>$&gt;$</td>
</tr>
<tr>
<td>-1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Count

- $c_1: 2 + 5 + 9 = 16$
- $c_2: 9 + 17 + 2 = 28$

Bob and Charlie win the election.
Scoring Rules: An Example

Election System
- 6 candidates
- $\alpha = (17, 9, 5, 2, 2, -1)$

Example Vote

<table>
<thead>
<tr>
<th></th>
<th>17</th>
<th>9</th>
<th>5</th>
<th>2</th>
<th>2</th>
<th>-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice:</td>
<td>$c_3$</td>
<td>$c_2$</td>
<td>$c_6$</td>
<td>$c_1$</td>
<td>$c_5$</td>
<td>$c_4$</td>
</tr>
<tr>
<td>Bob:</td>
<td>$c_2$</td>
<td>$c_3$</td>
<td>$c_1$</td>
<td>$c_5$</td>
<td>$c_4$</td>
<td>$c_6$</td>
</tr>
<tr>
<td>Charlie:</td>
<td>$c_4$</td>
<td>$c_1$</td>
<td>$c_6$</td>
<td>$c_2$</td>
<td>$c_3$</td>
<td>$c_5$</td>
</tr>
</tbody>
</table>

Count

<table>
<thead>
<tr>
<th>$c_1$</th>
<th>$c_2$</th>
<th>$c_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 + 5 + 9</td>
<td>9 + 17 + 2</td>
<td>17 + 9 + 2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>= 16</td>
<td>= 28</td>
<td>= 28</td>
</tr>
</tbody>
</table>
### Scoring Rules: An Example

<table>
<thead>
<tr>
<th>Election System</th>
<th>Example Vote</th>
</tr>
</thead>
<tbody>
<tr>
<td>- 6 candidates</td>
<td>- Alice: $c_3 &gt; c_2 &gt; c_6 &gt; c_1 &gt; c_5 &gt; c_4$</td>
</tr>
<tr>
<td>- $\alpha = (17, 9, 5, 2, 2, -1)$</td>
<td>- Bob: $c_2 &gt; c_3 &gt; c_1 &gt; c_5 &gt; c_4 &gt; c_6$</td>
</tr>
<tr>
<td></td>
<td>- Charlie: $c_4 &gt; c_1 &gt; c_6 &gt; c_2 &gt; c_3 &gt; c_5$</td>
</tr>
</tbody>
</table>

#### Count

<table>
<thead>
<tr>
<th>Candidate</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c_1$</td>
<td>$2 + 5 + 9 = 16$</td>
</tr>
<tr>
<td>$c_2$</td>
<td>$9 + 17 + 2 = 28$</td>
</tr>
<tr>
<td>$c_3$</td>
<td>$17 + 9 + 2 = 28$</td>
</tr>
<tr>
<td>$c_4$</td>
<td>$-1 + 2 + 17 = 18$</td>
</tr>
</tbody>
</table>

Bob and Charlie win the election.
Scoring Rules: An Example

**Election System**

- 6 candidates
- \( \alpha = (17, 9, 5, 2, 2, -1) \)

**Example Vote**

<table>
<thead>
<tr>
<th></th>
<th>17</th>
<th>9</th>
<th>5</th>
<th>2</th>
<th>2</th>
<th>-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>( c_3 )</td>
<td>( &gt; c_2 )</td>
<td>( &gt; c_6 )</td>
<td>( &gt; c_1 )</td>
<td>( &gt; c_5 )</td>
<td>( &gt; c_4 )</td>
</tr>
<tr>
<td>Bob</td>
<td>( c_2 )</td>
<td>( &gt; c_3 )</td>
<td>( &gt; c_1 )</td>
<td>( &gt; c_5 )</td>
<td>( &gt; c_4 )</td>
<td>( &gt; c_6 )</td>
</tr>
<tr>
<td>Charlie</td>
<td>( c_4 )</td>
<td>( &gt; c_1 )</td>
<td>( &gt; c_6 )</td>
<td>( &gt; c_2 )</td>
<td>( &gt; c_3 )</td>
<td>( &gt; c_5 )</td>
</tr>
</tbody>
</table>

**Count**

| \( c_1 \) | 2 + 5 + 9 = 16 |
| \( c_2 \) | 9 + 17 + 2 = 28 |
| \( c_3 \) | 17 + 9 + 2 = 28 |
| \( c_4 \) | -1 + 2 + 17 = 18 |
| \( c_5 \) | 2 + 2 - 1 = 1 |
Scoring Rules: An Example

**Election System**
- 6 candidates
- \( \alpha = (17, 9, 5, 2, 2, -1) \)

**Example Vote**

<table>
<thead>
<tr>
<th></th>
<th>17</th>
<th>9</th>
<th>5</th>
<th>2</th>
<th>2</th>
<th>-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>( \ast_3 )</td>
<td>( \ast_2 )</td>
<td>( \ast_6 )</td>
<td>( \ast_1 )</td>
<td>( \ast_5 )</td>
<td>( \ast_4 )</td>
</tr>
<tr>
<td>Bob</td>
<td>( \ast_2 )</td>
<td>( \ast_3 )</td>
<td>( \ast_1 )</td>
<td>( \ast_5 )</td>
<td>( \ast_4 )</td>
<td>( \ast_6 )</td>
</tr>
<tr>
<td>Charlie</td>
<td>( \ast_4 )</td>
<td>( \ast_1 )</td>
<td>( \ast_6 )</td>
<td>( \ast_2 )</td>
<td>( \ast_3 )</td>
<td>( \ast_5 )</td>
</tr>
</tbody>
</table>

**Count**

<table>
<thead>
<tr>
<th>Candidate</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \ast_1 )</td>
<td>2 + 5 + 9 = 16</td>
</tr>
<tr>
<td>( \ast_2 )</td>
<td>9 + 17 + 2 = 28</td>
</tr>
<tr>
<td>( \ast_3 )</td>
<td>17 + 9 + 2 = 28</td>
</tr>
<tr>
<td>( \ast_4 )</td>
<td>-1 + 2 + 17 = 18</td>
</tr>
<tr>
<td>( \ast_5 )</td>
<td>2 + 2 - 1 = 1</td>
</tr>
<tr>
<td>( \ast_6 )</td>
<td>5 - 1 + 5 = 9</td>
</tr>
</tbody>
</table>

Bob and Charlie win the election.
Scoring Rules: An Example

Election System

- 6 candidates
- \( \alpha = (17, 9, 5, 2, 2, -1) \)

Example Vote

<table>
<thead>
<tr>
<th></th>
<th>17</th>
<th>9</th>
<th>5</th>
<th>2</th>
<th>2</th>
<th>-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>c_3 &gt; c_2 &gt; c_6 &gt; c_1 &gt; c_5 &gt; c_4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bob</td>
<td>c_2 &gt; c_3 &gt; c_1 &gt; c_5 &gt; c_4 &gt; c_6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Charlie</td>
<td>c_4 &gt; c_1 &gt; c_6 &gt; c_2 &gt; c_3 &gt; c_5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Count

- \( c_1 = 2 + 5 + 9 = 16 \)
- \( c_2 = 9 + 17 + 2 = 28 \)
- \( c_3 = 17 + 9 + 2 = 28 \)
- \( c_4 = -1 + 2 + 17 = 18 \)
- \( c_5 = 2 + 2 - 1 = 1 \)
- \( c_6 = 5 - 1 + 5 = 9 \)

\( c_2 \) and \( c_3 \) win the election.
Scoring Rules: An Example

**Election System**
- 6 candidates
- \( \alpha = (17, 9, 5, 2, 2, -1) \)

**Example Vote**

<table>
<thead>
<tr>
<th>Candidate</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice:</td>
<td>17</td>
</tr>
<tr>
<td>Bob:</td>
<td>9</td>
</tr>
<tr>
<td>Charlie:</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>-1</td>
</tr>
</tbody>
</table>

| Alice: C3 > C2 > C6 > C1 > C5 > C4 |
| Bob: C2 > C3 > C1 > C5 > C4 > C6 |
| Charlie: C4 > C1 > C6 > C2 > C3 > C5 |

**Count**

<table>
<thead>
<tr>
<th>Candidate</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>2 + 5 + 9 = 16</td>
</tr>
<tr>
<td>C2</td>
<td>9 + 17 + 2 = 28</td>
</tr>
<tr>
<td>C3</td>
<td>17 + 9 + 2 = 28</td>
</tr>
<tr>
<td>C4</td>
<td>-1 + 2 + 17 = 18</td>
</tr>
<tr>
<td>C5</td>
<td>2 + 2 - 1 = 1</td>
</tr>
<tr>
<td>C6</td>
<td>5 - 1 + 5 = 9</td>
</tr>
</tbody>
</table>

\( c_2 \) and \( c_3 \) win the election.

**More Candidates?**

This “system” only works for 6 candidates.
Scoring Rule Families

Scoring Rules

- Defined by vector $\alpha = (\alpha_1, \ldots, \alpha_n)$
- Election system for $n$ candidates

**Want**: system for arbitrary number of candidates
Scoring Rule Families

Scoring Rules
- Defined by vector $\alpha = (\alpha_1, \ldots, \alpha_n)$
- Election system for $n$ candidates

**Want**: system for arbitrary number of candidates

Generalized Scoring Rules
- **generator**: function $f$ such that $f(n)$ is scoring vector of length $n$.
- generator $f \mapsto$ election system $\mathcal{E}_f$

Identify $f$ and $\mathcal{E}_f$. Talk about $f$-CCAV.
Overview

1. Introduction: Elections and Control
2. The Complexity of the Control Problem
3. A Class of Election Systems: Scoring Rules
4. The Price of Purity
5. A Dichotomy Theorem
   - Polynomial Time Results
   - Hardness Results
6. Conclusion
Uniformity and Other Conditions

Restrictions: Generators $f$ should be “nice”!

**Computationally** $f$ should be PTIME-computable (in the number of candidates).

**Conceptually** This does not make much sense:
- $f(1) = (1)$
Uniformity and Other Conditions

Restrictions: Generators $f$ should be “nice”!

**Computationally** $f$ should be PTIME-computable (in the number of candidates).

**Conceptually** This does not make much sense:

- $f(1) = (1)$
- $f(2) = (3, 1)$
Uniformity and Other Conditions

**Restrictions:** Generators $f$ should be “nice”!

**Computationally** $f$ should be PTIME-computable (in the number of candidates).

**Conceptually** This does not make much sense:

- $f(1) = (1)$
- $f(2) = (3, 1)$
- $f(3) = (1, 0, 0)$
Uniformity and Other Conditions

Restrictions: Generators $f$ should be “nice”!

**Computationally** $f$ should be PTIME-computable (in the number of candidates).

**Conceptually** This does not make much sense:

- $f(1) = (1)$
- $f(2) = (3, 1)$
- $f(3) = (1, 0, 0)$
- $f(4) = (1, 0, 0, 0)$
- $f(5) = (7, 5, 5, 0, 0)$
- $f(6) = (1, 0, 0, 0, 0)$

Vectors $f(n)$ and $f(n+1)$ should be related.
Restrictions: Generators $f$ should be “nice”!

Computationally $f$ should be PTIME-computable (in the number of candidates).

Conceptually This does not make much sense:

- $f(1) = (1)$
- $f(2) = (3, 1)$
- $f(3) = (1, 0, 0)$
- $f(4) = (1, 0, 0, 0)$
- $f(5) = (7, 5, 5, 0, 0)$
Uniformity and Other Conditions

Restrictions: Generators $f$ should be “nice”!

**Computationally** $f$ should be PTIME-computable (in the number of candidates).

**Conceptually** This does not make much sense:

- $f(1) = (1)$
- $f(2) = (3, 1)$
- $f(3) = (1, 0, 0)$
- $f(4) = (1, 0, 0, 0)$
- $f(5) = (7, 5, 5, 0, 0)$
- $f(6) = (1, 0, 0, 0, 0, 0)$
- ...
Uniformity and Other Conditions

Restrictions: Generators $f$ should be “nice”!

Computationally $f$ should be PTIME-computable (in the number of candidates).

Conceptually This does not make much sense:

- $f(1) = (1)$
- $f(2) = (3, 1)$
- $f(3) = (1, 0, 0)$
- $f(4) = (1, 0, 0, 0)$
- $f(5) = (7, 5, 5, 0, 0)$
- $f(6) = (1, 0, 0, 0, 0, 0)$
- ... 

Vectors $f(n)$ and $f(n + 1)$ should be related.
Pure Scoring Rules

Purity Constraint

Pure generator $f$: p-time computable and vector $f(n + 1)$ obtained from $f(n)$ by adding single coefficient.
**Pure Scoring Rules**

---

### Purity Constraint

**Pure** generator $f$: p-time computable and vector $f(n + 1)$ obtained from $f(n)$ by adding single coefficient.

### Examples

- **3-Approval** $f(1) = (1)$
Pure Scoring Rules

Purity Constraint

Pure generator $f$: p-time computable and vector $f(n + 1)$ obtained from $f(n)$ by adding single coefficient.

Examples

3-Approval $f(2) = (1, 1)$
**Pure Scoring Rules**

### Purity Constraint

Pure generator $f$: p-time computable and vector $f(n + 1)$ obtained from $f(n)$ by adding single coefficient.

### Examples

| 3-Approval | $f(3) = (1, 1, 1)$ |
Pure Scoring Rules

**Purity Constraint**

Pure generator $f$: p-time computable and vector $f(n + 1)$ obtained from $f(n)$ by adding single coefficient.

**Examples**

- **3-Approval** $f(4) = (1, 1, 1, 0)$
Pure Scoring Rules

Purity Constraint

**Pure** generator \( f \): p-time computable and vector \( f(n + 1) \) obtained from \( f(n) \) by adding single coefficient.

Examples

3-Approval \( f(5) = (1, 1, 1, 0, 0) \)
Pure Scoring Rules

Purity Constraint

Pure generator $f$: p-time computable and vector $f(n + 1)$ obtained from $f(n)$ by adding single coefficient.

Examples

3-Approval $f(6) = (1, 1, 1, 0, 0, 0)$
## Pure Scoring Rules

### Purity Constraint

A pure generator $f$: p-time computable and vector $f(n + 1)$ obtained from $f(n)$ by adding a single coefficient.

### Examples

- **3-Approval**
  
  $f(7) = (1, 1, 1, 0, 0, 0, 0)$

- **3-Veto**
  
  $f() = (0)$

- **Borda**
  
  $f() = (0)$
Pure Scoring Rules

**Purity Constraint**

Pure generator \( f \): p-time computable and vector \( f(n + 1) \) obtained from \( f(n) \) by adding single coefficient.

**Examples**

- **3-Approval** \( f(7) = (1, 1, 1, 0, 0, 0, 0) \)
- **3-Veto** \( f(1) = (0) \)
Pure Scoring Rules

Purity Constraint

Pure generator $f$: p-time computable and vector $f(n + 1)$ obtained from $f(n)$ by adding single coefficient.

Examples

3-Approval $f(7) = (1, 1, 1, 0, 0, 0, 0)$

3-Veto $f(2) = (0, 0)$
Pure Scoring Rules

**Purity Constraint**

Pure generator $f$: p-time computable and vector $f(n + 1)$ obtained from $f(n)$ by adding single coefficient.

**Examples**

3-Approval  $f(7) = (1, 1, 1, 0, 0, 0, 0)$

3-Veto  $f(3) = (0, 0, 0)$
Pure Scoring Rules

Purity Constraint

Pure generator $f$: p-time computable and vector $f(n + 1)$ obtained from $f(n)$ by adding single coefficient.

Examples

- **3-Approval** $f(7) = (1, 1, 1, 0, 0, 0, 0)$
- **3-Veto** $f(4) = (1, 0, 0, 0)$
Pure Scoring Rules

Purity Constraint

Pure generator $f$: p-time computable and vector $f(n + 1)$ obtained from $f(n)$ by adding single coefficient.

Examples

3-Approval $f(7) = (1, 1, 1, 0, 0, 0, 0, 0)$
3-Veto $f(5) = (1, 1, 0, 0, 0)$
Pure Scoring Rules

**Purity Constraint**

Pure generator $f$: p-time computable and vector $f(n + 1)$ obtained from $f(n)$ by adding single coefficient.

**Examples**

3-Approval $f(7) = (1, 1, 1, 0, 0, 0, 0, 0)$

3-Veto $f(6) = (1, 1, 1, 0, 0, 0)$
Pure Scoring Rules

Purity Constraint

Pure generator $f$: p-time computable and vector $f(n + 1)$ obtained from $f(n)$ by adding single coefficient.

Examples

3-Approval $f(7) = (1, 1, 1, 0, 0, 0, 0)$

3-Veto $f(7) = (1, 1, 1, 1, 0, 0, 0)$
Pure Scoring Rules

Purity Constraint

Pure generator $f$: p-time computable and vector $f(n + 1)$ obtained from $f(n)$ by adding single coefficient.

Examples

3-Approval $f(7) = (1, 1, 1, 0, 0, 0, 0)$

3-Veto $f(7) = (1, 1, 1, 1, 0, 0, 0)$

Borda $f(1) = (0)$
Pure Scoring Rules

Purity Constraint

Pure generator \( f \): p-time computable and vector \( f(n + 1) \) obtained from \( f(n) \) by adding single coefficient.

Examples

- **3-Approval** \( f(7) = (1, 1, 1, 0, 0, 0, 0) \)
- **3-Veto** \( f(7) = (1, 1, 1, 1, 0, 0, 0) \)
- **Borda** \( f(2) = (1, 0) \)
Pure Scoring Rules

Purity Constraint

Pure generator \( f \): p-time computable and vector \( f(n + 1) \) obtained from \( f(n) \) by adding single coefficient.

Examples

- **3-Approval** \( f(7) = (1, 1, 1, 0, 0, 0, 0, 0) \)
- **3-Veto** \( f(7) = (1, 1, 1, 1, 0, 0, 0) \)
- **Borda** \( f(3) = (2, 1, 0) \)
Pure Scoring Rules

**Purity Constraint**

Pure generator $f$: p-time computable and vector $f(n + 1)$ obtained from $f(n)$ by adding single coefficient.

**Examples**

- **3-Approval**  
  \[ f(7) = (1, 1, 1, 0, 0, 0, 0) \]

- **3-Veto**  
  \[ f(7) = (1, 1, 1, 1, 0, 0, 0) \]

- **Borda**  
  \[ f(4) = (3, 2, 1, 0) \]
## Pure Scoring Rules

### Purity Constraint

Pure generator $f$: p-time computable and vector $f(n + 1)$ obtained from $f(n)$ by adding single coefficient.

### Examples

<table>
<thead>
<tr>
<th>Scoring Rule</th>
<th>Vector $f(n)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>3-Approval</td>
<td>$(1, 1, 1, 0, 0, 0, 0)$</td>
</tr>
<tr>
<td>3-Veto</td>
<td>$(1, 1, 1, 1, 0, 0, 0)$</td>
</tr>
<tr>
<td>Borda</td>
<td>$(4, 3, 2, 1, 0)$</td>
</tr>
</tbody>
</table>
Pure Scoring Rules

Purity Constraint

Pure generator $f$: p-time computable and vector $f(n + 1)$ obtained from $f(n)$ by adding single coefficient.

Examples

3-Approval $f(7) = (1, 1, 1, 0, 0, 0, 0, 0)$

3-Veto $f(7) = (1, 1, 1, 1, 0, 0, 0, 0)$

Borda $f(6) = (5, 4, 3, 2, 1, 0)$
Pure Scoring Rules

Purity Constraint

Pure generator $f$: p-time computable and vector $f(n + 1)$ obtained from $f(n)$ by adding single coefficient.

Examples

3-Approval $f(7) = (1, 1, 1, 0, 0, 0, 0)$

3-Veto $f(7) = (1, 1, 1, 1, 0, 0, 0)$

Borda $f(7) = (6, 5, 4, 3, 2, 1, 0)$
Pure Scoring Rules

Purity Constraint

Pure generator $f$: p-time computable and vector $f(n + 1)$ obtained from $f(n)$ by adding single coefficient.

Examples

- **3-Approval** $f(7) = (1, 1, 1, 0, 0, 0, 0)$
- **3-Veto** $f(7) = (1, 1, 1, 1, 0, 0, 0)$
- **Borda** $f(7) = (6, 5, 4, 3, 2, 1, 0)$
Equivalent Generators

### Equivalence of Scoring Rules

Three Scoring Rules:

<table>
<thead>
<tr>
<th>Rule</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rule 1</td>
<td>$f_1(1) = 1$</td>
</tr>
<tr>
<td>Rule 2</td>
<td>$f_2(1) = 3$</td>
</tr>
<tr>
<td>Rule 3</td>
<td>$f_3(1) = 6$</td>
</tr>
</tbody>
</table>

These rules always choose the same winner!

Observation: $f_1$ and $f_2$ equivalent if $f_1$ is an affine transformation of $f_2$.

Affine transformation (as used in this talk will mean): $\alpha_i \rightarrow c \cdot \alpha_i + d$ with $c > 0$ (different $c$, $d$ for each vector length allowed).
Equivalent Generators

**Equivalence of Scoring Rules**

Three Scoring Rules:

- **Rule 1** \( f_1(2) = (2, 1) \)
- **Rule 2** \( f_2(2) = (4, 3) \)
- **Rule 3** \( f_3(2) = (8, 6) \)
Equivalent Generators

Equivalence of Scoring Rules

Three Scoring Rules:

- Rule 1 \( f_1(3) = (3, 2, 1) \)
- Rule 2 \( f_2(3) = (5, 4, 3) \)
- Rule 3 \( f_3(3) = (10, 8, 6) \)
Equivalent Generators

Equivalence of Scoring Rules

Three Scoring Rules:

Rule 1 \( f_1(4) = (4, 3, 2, 1) \)

Rule 2 \( f_2(4) = (6, 5, 4, 3) \)

Rule 3 \( f_3(4) = (12, 10, 8, 6) \)
Equivalent Generators

Equivalence of Scoring Rules

Three Scoring Rules:

- **Rule 1** \( f_1(5) = (5, 4, 3, 2, 1) \)
- **Rule 2** \( f_2(5) = (7, 6, 5, 4, 3) \)
- **Rule 3** \( f_3(5) = (14, 12, 10, 8, 6) \)
Equivalent Generators

Equivalence of Scoring Rules

Three Scoring Rules:

Rule 1 $f_1(6) = (6, 5, 4, 3, 2, 1)$

Rule 2 $f_2(6) = (8, 7, 6, 5, 4, 3)$

Rule 3 $f_3(6) = (16, 14, 12, 10, 8, 6)$
Equivalent Generators

Equivalence of Scoring Rules

Three Scoring Rules:

Rule 1 \( f_1(7) = (7, 6, 5, 4, 3, 2, 1) \)

Rule 2 \( f_2(7) = (9, 8, 7, 6, 5, 4, 3) \)

Rule 3 \( f_3(7) = (18, 16, 14, 12, 10, 8, 6) \)
Equivalent Generators

Equivalence of Scoring Rules

Three Scoring Rules:

Rule 1 $f_1(7) = (7, 6, 5, 4, 3, 2, 1)$
Rule 2 $f_2(7) = (9, 8, 7, 6, 5, 4, 3)$
Rule 3 $f_3(7) = (18, 16, 14, 12, 10, 8, 6)$

These rules always choose the same winner!
Equivalent Generators

Equivalence of Scoring Rules

Three Scoring Rules:

Rule 1  \( f_1(7) = (7, 6, 5, 4, 3, 2, 1) \)

Rule 2  \( f_2(7) = (9, 8, 7, 6, 5, 4, 3) \)

Rule 3  \( f_3(7) = (18, 16, 14, 12, 10, 8, 6) \)

These rules always choose the same winner!
Equivalent Generators

Equivalence of Scoring Rules

Three Scoring Rules:

Rule 1 $f_1(7) = (7, 6, 5, 4, 3, 2, 1)$
Rule 2 $f_2(7) = (9, 8, 7, 6, 5, 4, 3)$
Rule 3 $f_3(7) = (18, 16, 14, 12, 10, 8, 6)$

These rules always choose the same winner!
Equivalent Generators

Equivalence of Scoring Rules

Three Scoring Rules:

Rule 1 \( f_1(7) = (7, 6, 5, 4, 3, 2, 1) \)

Rule 2 \( f_2(7) = (9, 8, 7, 6, 5, 4, 3) \)

Rule 3 \( f_3(7) = (18, 16, 14, 12, 10, 8, 6) \)

These rules always choose the same winner!

Equivalent

Generators \( f_1 \) and \( f_2 \) equivalent, if they always choose the same winners
Equivalent Generators

Equivalence of Scoring Rules

Three Scoring Rules:

Rule 1  \( f_1(7) = (7, 6, 5, 4, 3, 2, 1) \)

Rule 2  \( f_2(7) = (9, 8, 7, 6, 5, 4, 3) \)

Rule 3  \( f_3(7) = (18, 16, 14, 12, 10, 8, 6) \)

These rules always choose the same winner!

Equivalent

Generators \( f_1 \) and \( f_2 \) equivalent, if they always choose the same winners

Observation

\( f_1 \) and \( f_2 \) equivalent iff \( f_1 \) is affine transformation of \( f_2 \).

Affine transformation (as used in this talk will mean): \( \alpha_i \rightarrow c \cdot \alpha_i + d \) with \( c > 0 \) (different \( c, d \) for each vector length allowed).
Normalized

Previous Work: Normalization

Assume: For each $n$: $f(n) = (\alpha^n_1, \alpha^n_2, \ldots, \alpha^n_n)$ with $\alpha^n_n = 0$
Normalization

**Previous Work: Normalization**

**Assume:** For each $n$: $f(n) = (\alpha^n_1, \alpha^n_2, \ldots, \alpha^n_n)$ with

- $\alpha^n_n = 0$
- $\gcd(\alpha^n_1, \alpha^n_2, \ldots, \alpha^n_{n-1}) = 1$
Normalization

Previous Work: Normalization

**Assume:** For each $n$: $f(n) = (\alpha_1^n, \alpha_2^n, \ldots, \alpha_n^n)$ with

- $\alpha_n^n = 0$
- $\gcd(\alpha_1^n, \alpha_2^n, \ldots, \alpha_{n-1}^n) = 1$
- each $\alpha_i^n$ is in $\mathbb{N}$
Normalization

Previous Work: Normalization

Assume: For each $n$: $f(n) = (\alpha_1^n, \alpha_2^n, \ldots, \alpha_n^n)$ with
- $\alpha_n^n = 0$
- $\gcd(\alpha_1^n, \alpha_2^n, \ldots, \alpha_{n-1}^n) = 1$
- each $\alpha_i^n$ is in $\mathbb{N}$

For each $n$: achieved by affine transformation.
Normalization

Previous Work: Normalization

Assume: For each $n$: $f(n) = (\alpha_1^n, \alpha_2^n, \ldots, \alpha_n^n)$ with

- $\alpha_n^n = 0$
- $\gcd(\alpha_1^n, \alpha_2^n, \ldots, \alpha_{n-1}^n) = 1$
- each $\alpha_i^n$ is in $\mathbb{N}$

For each $n$: achieved by affine transformation.

Question

Do we lose something?
Missing: The naturals are not enough

Dowdall System

Scores \((1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \frac{1}{5}, \ldots)\)
Missing: The naturals are not enough

Dowdall System

Scores \((1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \frac{1}{5}, \ldots)\)

Dowdall with natural numbers?

(1)

Note: The naturals are not enough as the scores do not include fractions. Dowdall System with natural numbers is not possible since it requires non-integer scores.
Missing: The naturals are not enough

**Dowdall System**

Scores \((1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \frac{1}{5}, \ldots)\)

**Dowdall with natural numbers?**

\((1)\)

\[\hat{=} (1)\]
Missing: The naturals are not enough

**Dowdall System**

Scores \((1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \frac{1}{5}, \ldots)\)

**Dowdall with natural numbers?**

\[(1) \hat{=} (1)\]
\[(1, \frac{1}{2}) \hat{=} (1)\]
Missing: The naturals are not enough

### Dowdall System

Scores \((1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \frac{1}{5}, \ldots)\)

### Dowdall with natural numbers?

\[
\begin{align*}
(1) & \\
(1, \frac{1}{2}) & = (\frac{2}{2}, \frac{1}{2})
\end{align*}
\]

\(\hat{=} (1)\)

Not a pure generator!

Purify by multiplication

Works only for finite length!
Missing: The naturals are not enough

**Dowdall System**

Scores \((1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \frac{1}{5}, \ldots)\)

**Dowdall with natural numbers?**

\[
\begin{align*}
(1) &= (1) \\
(1, \frac{1}{2}) &= (\frac{2}{2}, \frac{1}{2}) \quad \hat{=} (1) \\
\hat{=} (2, 1)
\end{align*}
\]
Missing: The naturals are not enough

Dowdall System

Scores \((1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \frac{1}{5}, \ldots)\)

Dowdall with natural numbers?

\[
(1) = (\frac{2}{2}, \frac{1}{2}) \quad \hat{=} (1) \\
(1, \frac{1}{2}) = (\frac{2}{2}, \frac{1}{2}) \quad \hat{=} (2, 1) \\
(1, \frac{1}{2}, \frac{1}{3})
\]
Missing: The naturals are not enough

Dowdall System

Scores \(1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \frac{1}{5}, \ldots\)

Dowdall with natural numbers?

\[
\begin{align*}
(1) & \\
(1, \frac{1}{2}) & = (\frac{2}{2}, \frac{1}{2}) \\
(1, \frac{1}{2}, \frac{1}{3}) & = (\frac{6}{6}, \frac{3}{6}, \frac{2}{6}) \\
\end{align*}
\]

\(\hat{=}\) (1) \\
\(\hat{=}\) (2, 1)

Not a pure generator! Purify by multiplication. Works only for finite length!
Missing: The naturals are not enough

### Dowdall System

Scores \((1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \frac{1}{5}, \ldots)\)

### Dowdall with natural numbers?

<table>
<thead>
<tr>
<th>Score</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>((1))</td>
<td>(\hat{=} (1))</td>
</tr>
<tr>
<td>((1, \frac{1}{2}))</td>
<td>(\hat{=} \left(\frac{2}{2}, \frac{1}{2}\right))</td>
</tr>
<tr>
<td>((1, \frac{1}{2}, \frac{1}{3}))</td>
<td>(\hat{=} \left(\frac{6}{6}, \frac{3}{6}, \frac{2}{6}\right))</td>
</tr>
</tbody>
</table>

Not a pure generator! Purify by multiplication. Works only for finite length!
Missing: The naturals are not enough

Dowdall System

Scores \((1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \frac{1}{5}, \ldots)\)

Dowdall with natural numbers?

\[
\begin{align*}
(1) & = (1) \\
(1, \frac{1}{2}) & = (\frac{2}{2}, \frac{1}{2}) \quad \hat{=} (1) \\
(1, \frac{1}{2}, \frac{1}{3}) & = (\frac{6}{6}, \frac{3}{6}, \frac{2}{6}) \quad \hat{=} (2, 1) \\
(1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}) & = (\frac{12}{12}, \frac{6}{12}, \frac{4}{12}, \frac{3}{12}) \quad \hat{=} (6, 3, 2)
\end{align*}
\]
Missing: The naturals are not enough

Dowdall System

Scores \((1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \frac{1}{5}, \ldots)\)

Dowdall with natural numbers?

\[
\begin{align*}
(1, \frac{1}{2}) & = (\frac{2}{2}, \frac{1}{2}) \\
(1, \frac{1}{2}, \frac{1}{3}) & = (\frac{6}{6}, \frac{3}{6}, \frac{2}{6}) \\
(1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}) & = (\frac{12}{12}, \frac{6}{12}, \frac{4}{12}, \frac{3}{12})
\end{align*}
\]

\(\hat{=} (1)\)

\(\hat{=} (2, 1)\)

\(\hat{=} (6, 3, 2)\)
The naturals are not enough

Dowdall System

Scores \((1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \frac{1}{5}, \ldots)\)

Dowdall with natural numbers?

\[(1)\]
\[(1, \frac{1}{2}) \equiv (\frac{2}{2}, \frac{1}{2})\]
\[(1, \frac{1}{2}, \frac{1}{3}) \equiv (\frac{6}{6}, \frac{3}{6}, \frac{2}{6})\]
\[(1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}) \equiv (\frac{12}{12}, \frac{6}{12}, \frac{4}{12}, \frac{3}{12})\]

Not a pure generator!

Purify by multiplication
Works only for finite length!
Missing: The naturals are not enough

Dowdall System

Scores \((1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \frac{1}{5}, \ldots)\)

Dowdall with natural numbers?

\[
\begin{align*}
(1) & \equiv (1) \\
(1, \frac{1}{2}) & = (\frac{2}{2}, \frac{1}{2}) \\
(1, \frac{1}{2}, \frac{1}{3}) & = (\frac{6}{6}, \frac{3}{6}, \frac{2}{6}) \\
(1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}) & = (\frac{12}{12}, \frac{6}{12}, \frac{4}{12}, \frac{3}{12}) \\
(1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \frac{1}{5}) & \equiv (12, 6, 4, 3)
\end{align*}
\]
Missing: The naturals are not enough

Dowdall System

Scores \((1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \frac{1}{5}, \ldots)\)

Dowdall with natural numbers?

\[(1)\]
\[(1, \frac{1}{2})\]
\[(1, \frac{1}{2}, \frac{1}{3})\]
\[(1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4})\]
\[(1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \frac{1}{5})\]
\[= \left(\frac{2}{2}, \frac{1}{2}\right)\]
\[= \left(\frac{6}{6}, \frac{3}{6}, \frac{2}{6}\right)\]
\[= \left(\frac{12}{12}, \frac{6}{12}, \frac{4}{12}, \frac{3}{12}\right)\]
\[= \left(\frac{60}{60}, \frac{30}{60}, \frac{20}{60}, \frac{15}{60}, \frac{12}{60}\right)\]
\[\hat{=} (1)\]
\[\hat{=} (2, 1)\]
\[\hat{=} (6, 3, 2)\]
\[\hat{=} (12, 6, 4, 3)\]
Missing: The naturals are not enough

### Dowdall System

Scores \((1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \frac{1}{5}, \ldots)\)

### Dowdall with natural numbers?

\[
\begin{align*}
(1) & \Rightarrow (1) \\
(1, \frac{1}{2}) & \Rightarrow (\frac{2}{2}, \frac{1}{2}) \\
(1, \frac{1}{2}, \frac{1}{3}) & \Rightarrow (\frac{6}{6}, \frac{3}{6}, \frac{2}{6}) \\
(1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}) & \Rightarrow (\frac{12}{12}, \frac{6}{12}, \frac{4}{12}, \frac{3}{12}) \\
(1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \frac{1}{5}) & \Rightarrow (\frac{60}{60}, \frac{30}{60}, \frac{20}{60}, \frac{15}{60}, \frac{12}{60}) \\
\end{align*}
\]

Not a pure generator!

Purify by multiplication
Works only for finite length!
Missing: The naturals are not enough

Dowdall System

Scores \((1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \frac{1}{5}, \ldots)\)

Dowdall with natural numbers?

\[
\begin{align*}
(1) & \\
(1, \frac{1}{2}) & = (\frac{2}{2}, \frac{1}{2}) \\
(1, \frac{1}{2}, \frac{1}{3}) & = (\frac{6}{6}, \frac{3}{6}, \frac{2}{6}) \\
(1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}) & = (\frac{12}{12}, \frac{6}{12}, \frac{4}{12}, \frac{3}{12}) \\
(1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \frac{1}{5}) & = (\frac{60}{60}, \frac{30}{60}, \frac{20}{60}, \frac{15}{60}, \frac{12}{60}) \\
(1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \frac{1}{5}, \frac{1}{6}) & \\
\end{align*}
\]

\(\hat{=} (1)\)

\(\hat{=} (1, \frac{1}{2})\)

\(\hat{=} (2, 1)\)

\(\hat{=} (6, 3, 2)\)

\(\hat{=} (12, 6, 4, 3)\)

\(\hat{=} (60, 30, 20, 15, 12)\)
Missing: The naturals are not enough

Dowdall System

Scores \((1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \frac{1}{5}, \ldots)\)

Dowdall with natural numbers?

\[
\begin{align*}
(1) & \quad = (\frac{2}{2}, \frac{1}{1}) \\
(1, \frac{1}{2}) & \quad = (\frac{2}{2}, \frac{1}{1}) \\
(1, \frac{1}{2}, \frac{1}{3}) & \quad = (\frac{6}{6}, \frac{3}{3}, \frac{2}{2}) \\
(1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}) & \quad = (\frac{12}{12}, \frac{6}{6}, \frac{4}{4}, \frac{3}{3}) \\
(1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \frac{1}{5}) & \quad = (\frac{60}{60}, \frac{30}{60}, \frac{20}{60}, \frac{15}{60}, \frac{12}{60}) \\
(1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \frac{1}{5}, \frac{1}{6}) & \quad = (\frac{60}{60}, \frac{30}{60}, \frac{20}{60}, \frac{15}{60}, \frac{12}{60}, \frac{10}{60})
\end{align*}
\]

\[\wedge (1) \quad \wedge (2, 1) \quad \wedge (6, 3, 2) \quad \wedge (12, 6, 4, 3) \quad \wedge (60, 30, 20, 15, 12)\]
Dowdall System

Scores \((1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \frac{1}{5}, \ldots)\)

Dowdall with natural numbers?

\[
\begin{align*}
(1) & = (\frac{2}{2}, \frac{1}{2}) \\
(1, \frac{1}{2}) & = (\frac{2}{6}, \frac{3}{6}, \frac{2}{6}) \\
(1, \frac{1}{2}, \frac{1}{3}) & = (\frac{12}{12}, \frac{6}{12}, \frac{4}{12}, \frac{3}{12}) \\
(1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}) & = (\frac{60}{60}, \frac{30}{60}, \frac{20}{60}, \frac{15}{60}, \frac{12}{60}) \\
(1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \frac{1}{5}) & = (\frac{60}{60}, \frac{30}{60}, \frac{20}{60}, \frac{15}{60}, \frac{12}{60}, \frac{10}{60}) \\
\end{align*}
\]

\(\wedge (1)\)

\(\wedge (2, 1)\)

\(\wedge (6, 3, 2)\)

\(\wedge (12, 6, 4, 3)\)

\(\wedge (60, 30, 20, 15, 12)\)

\(\wedge (60, 30, 20, 15, 12, 10)\)

Not a pure generator!

Works only for finite length!
Missing: The naturals are not enough

Dowdall System

Scores \((1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \frac{1}{5}, \ldots)\)

Dowdall with natural numbers?

\[
\begin{align*}
(1) \\
(1, \frac{1}{2}) & = \left(\frac{2}{2}, \frac{1}{2}\right) \\
(1, \frac{1}{2}, \frac{1}{3}) & = \left(\frac{6}{6}, \frac{3}{6}, \frac{2}{6}\right) \\
(1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}) & = \left(\frac{12}{12}, \frac{6}{12}, \frac{4}{12}, \frac{3}{12}\right) \\
(1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \frac{1}{5}) & = \left(\frac{60}{60}, \frac{30}{60}, \frac{20}{60}, \frac{15}{60}, \frac{12}{60}\right) \\
(1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \frac{1}{5}, \frac{1}{6}) & = \left(\frac{60}{60}, \frac{30}{60}, \frac{20}{60}, \frac{15}{60}, \frac{12}{60}, \frac{10}{60}\right)
\end{align*}
\]

Not a pure generator!
Missing: The naturals are not enough

Dowdall System

Scores \((1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \frac{1}{5}, \ldots)\)

Dowdall with natural numbers?

\[
\begin{align*}
(1) \\
(1, \frac{1}{2}) &= \left(\frac{2}{2}, \frac{1}{2}\right) \\
(1, \frac{1}{2}, \frac{1}{3}) &= \left(\frac{6}{6}, \frac{3}{6}, \frac{2}{6}\right) \\
(1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}) &= \left(\frac{12}{12}, \frac{6}{12}, \frac{4}{12}, \frac{3}{12}\right) \\
(1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \frac{1}{5}) &= \left(\frac{60}{60}, \frac{30}{60}, \frac{20}{60}, \frac{15}{60}, \frac{12}{60}\right) \\
(1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \frac{1}{5}, \frac{1}{6}) &= \left(\frac{60}{60}, \frac{30}{60}, \frac{20}{60}, \frac{15}{60}, \frac{12}{60}, \frac{10}{60}\right)
\end{align*}
\]

\(\hat{=} (60)\)

\(\hat{=} (60, 30)\)

\(\hat{=} (60, 30, 20)\)

\(\hat{=} (60, 30, 20, 15)\)

\(\hat{=} (60, 30, 20, 15, 12)\)

\(\hat{=} (60, 30, 20, 15, 12, 10)\)

Not a pure generator!

- Purify by multiplication
Missing: The naturals are not enough

Dowdall System
Scores \((1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \frac{1}{5}, \ldots)\)

Dowdall with natural numbers?

\[
\begin{align*}
(1) &= (\frac{2}{2}, \frac{1}{2}) \\
(1, \frac{1}{2}) &= (\frac{6}{6}, \frac{3}{6}, \frac{2}{6}) \\
(1, \frac{1}{2}, \frac{1}{3}) &= (\frac{12}{12}, \frac{6}{12}, \frac{4}{12}, \frac{3}{12}) \\
(1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}) &= (\frac{60}{60}, \frac{30}{60}, \frac{20}{60}, \frac{15}{60}, \frac{12}{60}) \\
(1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \frac{1}{5}) &= (\frac{60}{60}, \frac{30}{60}, \frac{20}{60}, \frac{15}{60}, \frac{12}{60}, \frac{10}{60}) \\
(1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \frac{1}{5}, \frac{1}{6}) &= (\frac{60}{60}, \frac{30}{60}, \frac{20}{60}, \frac{15}{60}, \frac{12}{60}, \frac{10}{60}, \frac{9}{60}) \\
(1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \frac{1}{5}, \frac{1}{6}, \frac{1}{7}) &= (\frac{60}{60}, \frac{30}{60}, \frac{20}{60}, \frac{15}{60}, \frac{12}{60}, \frac{10}{60}, \frac{9}{60}, \frac{8}{60})
\end{align*}
\]

\(\hat{=} (60)\)
\(\hat{=} (60, 30)\)
\(\hat{=} (60, 30, 20)\)
\(\hat{=} (60, 30, 20, 15)\)
\(\hat{=} (60, 30, 20, 15, 12)\)
\(\hat{=} (60, 30, 20, 15, 12, 10)\)

Not a pure generator!

- Purify by multiplication
- Works only for finite length!
Missing: The naturals are not enough

Dowdall System

Scores \( (1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \frac{1}{5}, \ldots) \)

Dowdall with natural numbers?

\[
\begin{align*}
(1) & \\
(1, \frac{1}{2}) & = (\frac{2}{2}, \frac{1}{2}) \\
(1, \frac{1}{2}, \frac{1}{3}) & = (\frac{6}{6}, \frac{3}{6}, \frac{2}{6}) \\
(1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}) & = (\frac{12}{12}, \frac{6}{12}, \frac{4}{12}, \frac{3}{12}) \\
(1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \frac{1}{5}) & = (\frac{60}{60}, \frac{30}{60}, \frac{20}{60}, \frac{15}{60}, \frac{12}{60}) \\
(1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \frac{1}{5}, \frac{1}{6}) & = (\frac{60}{60}, \frac{30}{60}, \frac{20}{60}, \frac{15}{60}, \frac{12}{60}, \frac{10}{60}) \\
(1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \frac{1}{5}, \frac{1}{6}, \frac{1}{7}) & = (\frac{420}{420}, \frac{210}{420}, \frac{140}{420}, \frac{105}{420}, \frac{84}{420}, \frac{70}{420}, \frac{60}{420})
\end{align*}
\]

\( \hat{=} \) (60)

\( \hat{=} \) (60, 30)

\( \hat{=} \) (60, 30, 20)

\( \hat{=} \) (60, 30, 20, 15)

\( \hat{=} \) (60, 30, 20, 15, 12)

\( \hat{=} \) (60, 30, 20, 15, 12, 10)

Not a pure generator!

- Purify by multiplication
- Works only for finite length!
Missing: The naturals are not enough

**Dowdall System**

Scores \((1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \frac{1}{5}, \ldots)\)

**Dowdall with natural numbers?**

\[
\begin{align*}
(1) & \\
(1, \frac{1}{2}) & = (\frac{2}{2}, \frac{1}{2}) \\
(1, \frac{1}{2}, \frac{1}{3}) & = (\frac{6}{6}, \frac{3}{6}, \frac{2}{6}) \\
(1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}) & = (\frac{12}{12}, \frac{6}{12}, \frac{4}{12}, \frac{3}{12}) \\
(1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \frac{1}{5}) & = (\frac{60}{60}, \frac{30}{60}, \frac{20}{60}, \frac{15}{60}, \frac{12}{60}) \\
(1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \frac{1}{5}, \frac{1}{6}) & = (\frac{420}{420}, \frac{210}{420}, \frac{140}{420}, \frac{105}{420}, \frac{84}{420}, \frac{70}{420}, \frac{60}{420}) \\
\end{align*}
\]

\[\hat{=} (60)\]
\[\hat{=} (60, 30)\]
\[\hat{=} (60, 30, 20)\]
\[\hat{=} (60, 30, 20, 15)\]
\[\hat{=} (60, 30, 20, 15, 12)\]
\[\hat{=} (60, 30, 20, 15, 12, 10)\]
\[\hat{=} (420, 210, 140, 105, 84, 70, 60)\]

Not a pure generator!

- Purify by multiplication
- Works only for finite length!
Missing: The naturals are not enough

Dowdall System

Scores \((1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \frac{1}{5}, \ldots)\)

Dowdall with natural numbers?

\[
\begin{align*}
(1) & \cong (60) \\
(1, \frac{1}{2}) & = (\frac{2}{2}, \frac{1}{2}) \\
(1, \frac{1}{2}, \frac{1}{3}) & = (\frac{6}{6}, \frac{3}{6}, \frac{2}{6}) \\
(1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}) & = (\frac{12}{12}, \frac{6}{12}, \frac{4}{12}, \frac{3}{12}) \\
(1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \frac{1}{5}) & = (\frac{60}{60}, \frac{30}{60}, \frac{20}{60}, \frac{15}{60}, \frac{12}{60}) \\
(1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \frac{1}{5}, \frac{1}{6}) & = (\frac{60}{60}, \frac{30}{60}, \frac{20}{60}, \frac{15}{60}, \frac{12}{60}, \frac{10}{60}) \\
(1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \frac{1}{5}, \frac{1}{6}, \frac{1}{7}) & = (\frac{420}{420}, \frac{210}{420}, \frac{140}{420}, \frac{105}{420}, \frac{84}{420}, \frac{70}{420}, \frac{60}{420})
\end{align*}
\]

Not a pure generator!

- Purify by multiplication
- Works only for finite length!

Dowdall cannot be expressed with natural numbers.
Consequence

When we require purity:

- Universe for coefficients matters (\(\mathbb{Q}, \mathbb{Z}, \mathbb{N}\))
- Choosing coefficients from \(\mathbb{N}\) is a proper restriction
Expressivity

**Consequence**

When we require purity:
- Universe for coefficients matters ($\mathbb{Q}$, $\mathbb{Z}$, $\mathbb{N}$)
- Choosing coefficients from $\mathbb{N}$ is a proper restriction

**Other two Assumptions?**
- Last coefficient 0?
- Greatest common divisor of (non-zero) coefficients always 1?
Expressivity

Consequence

When we require purity:

- Universe for coefficients matters ($\mathbb{Q}, \mathbb{Z}, \mathbb{N}$)
- Choosing coefficients from $\mathbb{N}$ is a proper restriction

Other two Assumptions?

- Last coefficient 0?
- Greatest common divisor of (non-zero) coefficients always 1?

Both incompatible with purity.
Theorem: A missing Generator

For some pure generator $f_1$ there is no pure generator $f_2$ with

- $f_1$ and $f_2$ equivalent, and
- $f_2$ has last coefficient zero.
Theorem: A missing Generator

For some pure generator $f_1$ there is no pure generator $f_2$ with

- $f_1$ and $f_2$ equivalent, and
- $f_2$ has last coefficient zero.

Proof

$f_1(1) = (5)$
Theorem: A missing Generator

For some pure generator $f_1$ there is no pure generator $f_2$ with

- $f_1$ and $f_2$ equivalent, and
- $f_2$ has last coefficient zero.

Proof

$f_1(2) = (6, 5)$
Last Coefficient 0 is a restriction

**Theorem: A missing Generator**

For some **pure** generator $f_1$ there is no **pure** generator $f_2$ with

- $f_1$ and $f_2$ equivalent, and
- $f_2$ has last coefficient zero.

**Proof**

$f_1(3) = (7, 6, 5)$
Theorem: A missing Generator

For some pure generator $f_1$ there is no pure generator $f_2$ with

- $f_1$ and $f_2$ equivalent, and
- $f_2$ has last coefficient zero.

Proof

$f_1(4) = (8, 7, 6, 5)$
Theorem: A missing Generator

For some pure generator \( f_1 \) there is no pure generator \( f_2 \) with

- \( f_1 \) and \( f_2 \) equivalent, and
- \( f_2 \) has last coefficient zero.

Proof

\[ f_1(5) = (8, 7, 6, 5, 0) \]
Theorem: A missing Generator

For some pure generator $f_1$ there is no pure generator $f_2$ with
- $f_1$ and $f_2$ equivalent, and
- $f_2$ has last coefficient zero.

Proof

$f_1(5) = (8, 7, 6, 5, 0)$

Assume there is an equivalent normalized $f_2$. 
Last Coefficient 0 is a restriction

Theorem: A missing Generator

For some pure generator $f_1$ there is no pure generator $f_2$ with
- $f_1$ and $f_2$ equivalent, and
- $f_2$ has last coefficient zero.

Proof

$f_1(5) = (8, 7, 6, 5, 0)$

Assume there is an equivalent normalized $f_2$.

- $f_1(4) = (8, 7, 6, 5)$
Last Coefficient 0 is a restriction

Theorem: A missing Generator

For some pure generator $f_1$ there is no pure generator $f_2$ with

- $f_1$ and $f_2$ equivalent, and
- $f_2$ has last coefficient zero.

Proof

Assume there is an equivalent normalized $f_2$.

- $f_1(4) = (8, 7, 6, 5)$
- $f_2(4) = (3k, 2k, k, 0)$
Last Coefficient 0 is a restriction

**Theorem: A missing Generator**

For some **pure** generator $f_1$ there is no **pure** generator $f_2$ with
- $f_1$ and $f_2$ equivalent, and
- $f_2$ has last coefficient zero.

**Proof**

$f_1(5) = (8, 7, 6, 5, 0)$

Assume there is an equivalent normalized $f_2$.

- $f_1(4) = (8, 7, 6, 5)$
- $f_2(4) = (3k, 2k, k, 0)$
- $f_1(5) = (8, 7, 6, 5, 0)$
Last Coefficient 0 is a restriction

**Theorem: A missing Generator**

For some pure generator $f_1$ there is no pure generator $f_2$ with

- $f_1$ and $f_2$ equivalent, and
- $f_2$ has last coefficient zero.

**Proof**

$f_1(5) = (8, 7, 6, 5, 0)$

Assume there is an equivalent normalized $f_2$.

- $f_1(4) = (8, 7, 6, 5)$
- $f_2(4) = (3k, 2k, k, 0)$
- $f_1(5) = (8, 7, 6, 5, 0)$
- $f_2(5) = (3k, 2k, k, 0)$
Last Coefficient 0 is a restriction

Theorem: A missing Generator

For some pure generator \( f_1 \) there is no pure generator \( f_2 \) with
- \( f_1 \) and \( f_2 \) equivalent, and
- \( f_2 \) has last coefficient zero.

Proof

\[
f_1(5) = (8, 7, 6, 5, 0)
\]

Assume there is an equivalent normalized \( f_2 \).

- \( f_1(4) = (8, 7, 6, 5) \)
- \( f_2(4) = (3k, 2k, k, 0) \)
- \( f_1(5) = (8, 7, 6, 5, 0) \)
- \( f_2(5) = (?, 3k, 2k, k, 0) \)
Theorem: A missing Generator

For some pure generator $f_1$ there is no pure generator $f_2$ with

- $f_1$ and $f_2$ equivalent, and
- $f_2$ has last coefficient zero.

Proof

$f_1(5) = (8, 7, 6, 5, 0)$

Assume there is an equivalent normalized $f_2$.

- $f_1(4) = (8, 7, 6, 5)$
- $f_2(4) = (3k, 2k, k, 0)$
- $f_1(5) = (8, 7, 6, 5, 0)$
- $f_2(5) = (3k, ?, 2k, k, 0)$
Theorem: A missing Generator

For some pure generator $f_1$ there is no pure generator $f_2$ with

- $f_1$ and $f_2$ equivalent, and
- $f_2$ has last coefficient zero.

Proof

$f_1(5) = (8, 7, 6, 5, 0)$

Assume there is an equivalent normalized $f_2$.

- $f_1(4) = (8, 7, 6, 5)$
- $f_2(4) = (3k, 2k, k, 0)$
- $f_1(5) = (8, 7, 6, 5, 0)$
- $f_2(5) = (3k, 2k, ?, k, 0)$
Last Coefficient 0 is a restriction

**Theorem: A missing Generator**

For some **pure** generator $f_1$ there is no **pure** generator $f_2$ with
- $f_1$ and $f_2$ equivalent, and
- $f_2$ has last coefficient zero.

**Proof**

$f_1(5) = (8, 7, 6, 5, 0)$

Assume there is an equivalent normalized $f_2$.

- $f_1(4) = (8, 7, 6, 5)$
- $f_2(4) = (3k, 2k, k, 0)$

- $f_1(5) = (8, 7, 6, 5, 0)$
- $f_2(5) = (3k, 2k, k, ?, 0)$
Last Coefficient 0 is a restriction

**Theorem: A missing Generator**

For some **pure** generator $f_1$ there is no **pure** generator $f_2$ with

- $f_1$ and $f_2$ equivalent, and
- $f_2$ has last coefficient zero.

**Proof**

$f_1(5) = (8, 7, 6, 5, 0)$

Assume there is an equivalent normalized $f_2$.

- $f_1(4) = (8, 7, 6, 5)$
- $f_2(4) = (3k, 2k, k, 0)$
- $f_1(5) = (8, 7, 6, 5, 0)$
- $f_2(5) = (3k, 2k, k, 0, ?)$
Last Coefficient 0 is a restriction

**Theorem: A missing Generator**

For some pure generator $f_1$ there is no pure generator $f_2$ with
- $f_1$ and $f_2$ equivalent, and
- $f_2$ has last coefficient zero.

**Proof**

$f_1(5) = (8, 7, 6, 5, 0)$

Assume there is an equivalent normalized $f_2$.

- $f_1(4) = (8, 7, 6, 5)$
- $f_2(4) = (3k, 2k, k, 0)$
- $f_1(5) = (8, 7, 6, 5, 0)$
- $f_2(5) = (3k, 2k, k, 0, ?)$

$f_2$ is different on elections with 5 candidates already.
Purity Condition

Just Seen: Normalization kills purity

**Consequence:** Purity is too strong.
Purity Condition

**Just Seen:** Normalization kills purity

**Consequence:** Purity is too strong.

**New Purity Condition**

Generator $f$ **flexible**: for each $n$, $f(n+1)$ obtained from $f(n)$ by
- applying affine transformation,
- adding new coefficient.
Purity Condition

Just Seen: Normalization kills purity

**Consequence**: Purity is too strong.

New Purity Condition

Generator $f$ **flexible**: for each $n$, $f(n + 1)$ obtained from $f(n)$ by
- applying affine transformation,
- adding new coefficient.

**Consequence**
- Normalization built-in
- Looking at natural numbers is enough
Purity Condition

Just Seen: Normalization kills purity

Consequence: Purity is too strong.

New Purity Condition

Generator $f$ **flexible**: for each $n$, $f(n + 1)$ obtained from $f(n)$ by
- applying affine transformation,
- adding new coefficient.

Consequence

- Normalization built-in
- Looking at natural numbers is enough

From now on

We study flexible generators with coefficients from $\mathbb{N}$. 
Overview

1 Introduction: Elections and Control

2 The Complexity of the Control Problem

3 A Class of Election Systems: Scoring Rules

4 The Price of Purity

5 A Dichotomy Theorem
   - Polynomial Time Results
   - Hardness Results

6 Conclusion
Complexity Classification

**Consequence**

Flexible generators capture all systems we want to.
Complexity Classification

Consequence
Flexible generators capture all systems we want to.

Theorem
CCAV for the following $f$ (and equivalent) is solvable in PTIME:

- for $k \leq 3$:
  - $k$-Approval:
    $$f(n) = (1, ..., 1, n-k, 0, ..., 0)$$
  - $k$-Veto:
    $$f(n) = (k, 1, ..., 1, n-k, 0, ..., 0)$$

- Gen. 2-App.:
  $$f(n) = (\alpha, \beta, 0, ..., 0)$$

- Appr/Veto:
  $$f(n) = (2, 1, 1, ..., 1, 0)$$

For all flexible generators $f$ not equivalent to one of these, $f$-CCAV is NP-complete.
Complexity Classification

Consequence

Flexible generators capture all systems we want to.

Theorem

CCAV for the following $f$ (and equivalent) is solvable in PTIME:

$k$-Approval , $k \leq 3$: $f(n) = (1, \ldots, 1, 0, \ldots, 0)$
Complexity Classification

Consequence
Flexible generators capture all systems we want to.

Theorem
CCAV for the following $f$ (and equivalent) is solvable in PTIME:

- $k$-Approval, $k \leq 3$: $f(n) = (1, \ldots, 1, 0, \ldots, 0)$
- $k$-Veto, $k \leq 2$: $f(n) = (1, \ldots, 1, 0, \ldots, 0)$
Complexity Classification

Consequence
Flexible generators capture all systems we want to.

Theorem

CCAV for the following $f$ (and equivalent) is solvable in PTIME:

- $k$-Approval, $k \leq 3$: $f(n) = (1, \ldots, 1, 0, \ldots, 0)$
- $k$-Veto, $k \leq 2$: $f(n) = (1, \ldots, 1, 0, \ldots, 0)$
- Gen. 2-App. $f(n) = (\alpha, \beta, 0, \ldots, 0)$ for fixed $\alpha, \beta$
Complexity Classification

Consequence
Flexible generators capture all systems we want to.

Theorem

CCAV for the following $f$ (and equivalent) is solvable in PTIME:

- $k$-Approval, $k \leq 3$: $f(n) = (1, \ldots, 1, 0, \ldots, 0)$
- $k$-Veto, $k \leq 2$: $f(n) = (1, \ldots, 1, 0, \ldots, 0)$
- Gen. 2-App. $f(n) = (\alpha, \beta, 0, \ldots, 0)$ for fixed $\alpha, \beta$
- Appr/Veto $f(n) = (2, 1, \ldots, 1, 0)$
Complexity Classification

Consequence
Flexible generators capture all systems we want to.

Theorem

CCAV for the following \( f \) (and equivalent) is solvable in PTIME:

- **k-Approval**, \( k \leq 3 \):
  \[ f(n) = (1, \ldots, 1, 0, \ldots, 0) \]

- **k-Veto**, \( k \leq 2 \):
  \[ f(n) = (1, \ldots, 1, 0, \ldots, 0) \]

- **Gen. 2-App.**
  \[ f(n) = (\alpha, \beta, 0, \ldots, 0) \text{ for fixed } \alpha, \beta \]

- **Appr/Veto**
  \[ f(n) = (2, 1, \ldots, 1, 0) \]

For all flexible generators \( f \) not equivalent to one of these, \( f \)-CCAV is NP-complete.
Overview

1. Introduction: Elections and Control
2. The Complexity of the Control Problem
3. A Class of Election Systems: Scoring Rules
4. The Price of Purity
5. A Dichotomy Theorem
   - Polynomial Time Results
   - Hardness Results
6. Conclusion
A polynomial-time result

**Theorem**

CCAV for 3-Approval can be solved in polynomial time.
A polynomial-time result

**Theorem**

CCAV for 3-Approval can be solved in polynomial time.

3-Approval: \((1, 1, 1, 0, \ldots, 0)\)
A polynomial-time result

**Theorem**

CCAV for 3-Approval can be solved in polynomial time.

3-Approval: \((1, 1, 1, 0, \ldots, 0)\)

**Proof**

**Task:** Given voters \(R\) and \(U\), candidate \(p\), number \(k\)
A polynomial-time result

Theorem

CCAV for 3-Approval can be solved in polynomial time.

3-Approval: \( (1, 1, 1, 0, \ldots, 0) \)

Proof

**Task:** Given voters \( R \) and \( U \), candidate \( p \), number \( k \)

Find \( U' \subseteq U \) with \(|U'| \leq k\) such that \( p \) wins if \( R \cup U' \) vote
A polynomial-time result

**Theorem**

CCAV for 3-Approval can be solved in polynomial time.

**3-Approval:** $(1, 1, 1, 0, \ldots, 0)$

**Proof**

**Task:** Given voters $R$ and $U$, candidate $p$, number $k$

Find $U' \subseteq U$ with $|U'| \leq k$ such that $p$ wins if $R \cup U'$ vote

- Vote $(c_1, c_2, c_3, \ldots, c_n)$ **approves** $c_1$, $c_2$ and $c_3$
A polynomial-time result

**Theorem**

CCAV for 3-Approval can be solved in polynomial time.

**3-Approval:** $(1, 1, 1, 0, \ldots, 0)$

**Proof**

**Task:** Given voters $R$ and $U$, candidate $p$, number $k$

Find $U' \subseteq U$ with $|U'| \leq k$ such that $p$ wins if $R \cup U'$ vote

- Vote $(c_1, c_2, c_3, \ldots, c_n)$ **approves** $c_1$, $c_2$ and $c_3$
- Only choose votes that approve $p$
A polynomial-time result

**Theorem**

CCAV for 3-Approval can be solved in polynomial time.

3-Approval: $(1, 1, 1, 0, \ldots, 0)$

**Proof**

**Task**: Given voters $R$ and $U$, candidate $p$, number $k$

Find $U' \subseteq U$ with $|U'| \leq k$ such that $p$ wins if $R \cup U'$ vote

- Vote $(c_1, c_2, c_3, \ldots, c_n)$ **approves** $c_1, c_2$ and $c_3$
- Only choose votes that approve $p$
- Forget about all other votes
A polynomial-time result

**Theorem**

CCAV for 3-Approval can be solved in polynomial time.

3-Approval: \((1, 1, 1, 0, \ldots, 0)\)

**Proof**

**Task:** Given voters \(R\) and \(U\), candidate \(p\), number \(k\)  
Find \(U' \subseteq U\) with \(|U'| \leq k\) such that \(p\) wins if \(R \cup U'\) vote

- Vote \((c_1, c_2, c_3, \ldots, c_n)\) **approves** \(c_1, c_2\) and \(c_3\)
- Only choose votes that approve \(p\)
- Forget about all other votes
- W.l.o.g.: add exactly \(k\) votes
A polynomial-time result

**Theorem**

CCAV for 3-Approval can be solved in polynomial time.

3-Approval: $(1, 1, 1, 0, \ldots, 0)$

**Proof**

**Task:** Given voters $R$ and $U$, candidate $p$, number $k$  

Find $U' \subseteq U$ with $|U'| \leq k$ such that $p$ wins if $R \cup U'$ vote

- Vote $(c_1, c_2, c_3, \ldots, c_n)$ **approves** $c_1, c_2$ and $c_3$
- Only choose votes that approve $p$
- Forget about all other votes
- W.l.o.g.: add exactly $k$ votes

$\rightarrow$ final score of $p$: $\text{final}_p = \text{current}_p + k$
A polynomial-time result

**Theorem**

CCAV for 3-Approval can be solved in polynomial time.

**3-Approval:** \((1, 1, 1, 0, \ldots, 0)\)

**Proof**

**Task:** Given voters \(R\) and \(U\), candidate \(p\), number \(k\)
Find \(U' \subseteq U\) with \(|U'| \leq k\) such that \(p\) wins if \(R \cup U'\) vote

- Vote \((c_1, c_2, c_3, \ldots, c_n)\) **approves** \(c_1, c_2\) and \(c_3\)
- Only choose votes that approve \(p\)
- Forget about all other votes
- W.l.o.g.: add exactly \(k\) votes

\(\rightarrow\) final score of \(p\): \(\text{final}_p = \text{current}_p + k\)

\(\rightarrow\) each \(c \neq p\): may gain at most \(\text{maxgain}_c = \text{final}_p - \text{current}_c\)
A polynomial-time result

Theorem

CCAV for 3-Approval can be solved in polynomial time.
A polynomial-time result

**Theorem**

CCAV for 3-Approval can be solved in polynomial time.

**Proof (continued)**

**Situation:**

- we add \( k \) voters
A polynomial-time result

Theorem

CCAV for 3-Approval can be solved in polynomial time.

Proof (continued)

Situation:

- we add \( k \) voters
- each approves \( p \) and two others
A polynomial-time result

**Theorem**

CCAV for 3-Approval can be solved in polynomial time.

**Proof (continued)**

**Situation:**
- we add $k$ voters
- each approves $p$ and two others
- each candidate $c \neq p$ may gain at most $\text{maxgain}_c$ points
A polynomial-time result

**Theorem**

CCAV for 3-Approval can be solved in polynomial time.

**Proof (continued)**

Situation:
- we add $k$ voters
- each approves $p$ and two others
- each candidate $c \neq p$ may gain at most $\text{maxgain}_c$ points

![Image of colored figures with numerical values below]

- $\leq 10$
- $\leq 7$
- $\leq 6$
- $\leq 1$
- $\leq 0$
- $\leq -1$
A polynomial-time result

Theorem

CCAV for 3-Approval can be solved in polynomial time.

Proof (continued)

Situation:

- we add $k$ voters
- each approves $p$ and two others
- each candidate $c \neq p$ may gain at most $\text{maxgain}_c$ points
A polynomial-time result

**Theorem**

CCAV for 3-Approval can be solved in polynomial time.

**Proof (continued)**

**Situation:**
- we add $k$ voters
- each approves $p$ and two others
- each candidate $c \neq p$ may gain at most $\max \text{gain}_c$ points
A polynomial-time result: Edge Matching algorithm

Votes
A polynomial-time result: Edge Matching algorithm

Votes

- $p > c_1 > c_3 > \ldots$

Graph

- $c_1$ to $c_3$
- $c_3$ to $c_4$
- $c_4$ to $c_5$
- $c_5$ to $c_6$
- $c_6$ to $c_7$

Other systems

- 2-Veto $(1, 1, \ldots, 1, 0, 0)$: similar.
- $(\alpha, \beta, 0, \ldots, 0)$: similar.
A polynomial-time result: Edge Matching algorithm

Votes
- \( p > c_1 > c_3 > \ldots \)
- \( p > c_2 > c_4 > \ldots \)

Graph

\begin{align*}
&c_1 & c_2 \\
&| & |
\end{align*}

\begin{align*}
&c_7 \\
&| & |
\end{align*}

\begin{align*}
&c_3 \\
&| & |
\end{align*}

\begin{align*}
&c_4
\end{align*}

\begin{align*}
&c_5 \\
&| & |
\end{align*}

\begin{align*}
&c_6
\end{align*}
A polynomial-time result: Edge Matching algorithm

Votes
- $p > c_1 > c_3 > \ldots$
- $p > c_2 > c_4 > \ldots$
- $p > c_2 > c_3 > \ldots$

Graph

- $c_1 \rightarrow c_2$
- $c_1 \rightarrow c_3$
- $c_2 \rightarrow c_4$
- $c_3 \rightarrow c_6$
- $c_4 \rightarrow c_7$
- $c_5 \rightarrow c_6$
A polynomial-time result: Edge Matching algorithm

Votes
- \( p > c_1 > c_3 > \ldots \)
- \( p > c_2 > c_4 > \ldots \)
- \( p > c_2 > c_3 > \ldots \)
- \( p > c_5 > c_4 > \ldots \)

Graph

\( c_1 \) \( c_2 \) \( c_3 \) \( c_4 \) \( c_5 \) \( c_6 \) \( c_7 \)
A polynomial-time result: Edge Matching algorithm

Votes
- \( p > c_1 > c_3 > \ldots \)
- \( p > c_2 > c_4 > \ldots \)
- \( p > c_2 > c_3 > \ldots \)
- \( p > c_5 > c_4 > \ldots \)
- \( p > c_6 > c_3 > \ldots \)
A polynomial-time result: Edge Matching algorithm

**Votes**
- \( p > c_1 > c_3 > \ldots \)
- \( p > c_2 > c_4 > \ldots \)
- \( p > c_2 > c_3 > \ldots \)
- \( p > c_5 > c_4 > \ldots \)
- \( p > c_6 > c_3 > \ldots \)
- \( p > c_7 > c_1 > \ldots \)

**Graph**
A polynomial-time result: Edge Matching algorithm

<table>
<thead>
<tr>
<th>Votes</th>
</tr>
</thead>
<tbody>
<tr>
<td>• p &gt; c₁ &gt; c₃ &gt; ...</td>
</tr>
<tr>
<td>• p &gt; c₂ &gt; c₄ &gt; ...</td>
</tr>
<tr>
<td>• p &gt; c₂ &gt; c₃ &gt; ...</td>
</tr>
<tr>
<td>• p &gt; c₅ &gt; c₄ &gt; ...</td>
</tr>
<tr>
<td>• p &gt; c₆ &gt; c₃ &gt; ...</td>
</tr>
<tr>
<td>• p &gt; c₇ &gt; c₁ &gt; ...</td>
</tr>
<tr>
<td>• p &gt; c₆ &gt; c₇ &gt; ...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Graph</th>
</tr>
</thead>
<tbody>
<tr>
<td>C₃ → C₁, C₄ → C₂, C₇ → C₆, C₅ → C₃</td>
</tr>
</tbody>
</table>

Graph: C₇, C₆, C₅, C₄, C₃, C₂, C₁

Other systems: 2-Veto (1, 1, ..., 1, 0, 0); similar. (α, β, 0, ..., 0); similar.
A polynomial-time result: Edge Matching algorithm

Votes
- \( p > c_1 > c_3 > \ldots \)
- \( p > c_2 > c_4 > \ldots \)
- \( p > c_2 > c_3 > \ldots \)
- \( p > c_5 > c_4 > \ldots \)
- \( p > c_6 > c_3 > \ldots \)
- \( p > c_7 > c_1 > \ldots \)
- \( p > c_6 > c_7 > \ldots \)
- \( p > c_3 > c_6 > \ldots \)

Graph

\[
\begin{align*}
C_1 &\rightarrow C_3 \\
C_1 &\rightarrow C_4 \\
C_2 &\rightarrow C_3 \\
C_2 &\rightarrow C_4 \\
C_3 &\rightarrow C_5 \\
C_5 &\rightarrow C_6 \\
C_7 &\rightarrow C_4 \\
C_6 &\rightarrow C_7 \\
\end{align*}
\]
A polynomial-time result: Edge Matching algorithm

<table>
<thead>
<tr>
<th>Votes</th>
</tr>
</thead>
<tbody>
<tr>
<td>( p &gt; c_1 &gt; c_3 &gt; \ldots )</td>
</tr>
<tr>
<td>( p &gt; c_2 &gt; c_4 &gt; \ldots )</td>
</tr>
<tr>
<td>( p &gt; c_2 &gt; c_3 &gt; \ldots )</td>
</tr>
<tr>
<td>( p &gt; c_5 &gt; c_4 &gt; \ldots )</td>
</tr>
<tr>
<td>( p &gt; c_6 &gt; c_3 &gt; \ldots )</td>
</tr>
<tr>
<td>( p &gt; c_7 &gt; c_1 &gt; \ldots )</td>
</tr>
<tr>
<td>( p &gt; c_6 &gt; c_7 &gt; \ldots )</td>
</tr>
<tr>
<td>( p &gt; c_3 &gt; c_6 &gt; \ldots )</td>
</tr>
<tr>
<td>( p &gt; c_3 &gt; c_5 &gt; \ldots )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Graph</th>
</tr>
</thead>
<tbody>
<tr>
<td>( c_1 ) ( \rightarrow ) ( c_2 ) ( \rightarrow ) ( c_3 ) ( \rightarrow ) ( c_4 ) ( \rightarrow ) ( c_7 ) ( \rightarrow ) ( c_6 ) ( \rightarrow ) ( c_5 )</td>
</tr>
<tr>
<td>( c_3 ) ( \rightarrow ) ( c_7 ) ( \rightarrow ) ( c_6 ) ( \rightarrow ) ( c_5 ) ( \rightarrow ) ( c_3 )</td>
</tr>
<tr>
<td>( p &gt; c_1 &gt; c_3 &gt; \ldots )</td>
</tr>
<tr>
<td>( p &gt; c_2 &gt; c_4 &gt; \ldots )</td>
</tr>
<tr>
<td>( p &gt; c_2 &gt; c_3 &gt; \ldots )</td>
</tr>
<tr>
<td>( p &gt; c_5 &gt; c_4 &gt; \ldots )</td>
</tr>
<tr>
<td>( p &gt; c_6 &gt; c_3 &gt; \ldots )</td>
</tr>
<tr>
<td>( p &gt; c_7 &gt; c_1 &gt; \ldots )</td>
</tr>
<tr>
<td>( p &gt; c_6 &gt; c_7 &gt; \ldots )</td>
</tr>
<tr>
<td>( p &gt; c_3 &gt; c_6 &gt; \ldots )</td>
</tr>
<tr>
<td>( p &gt; c_3 &gt; c_5 &gt; \ldots )</td>
</tr>
</tbody>
</table>
A polynomial-time result: Edge Matching algorithm

### Votes
- $p > c_1 > c_3 > \ldots$
- $p > c_2 > c_4 > \ldots$
- $p > c_2 > c_3 > \ldots$
- $p > c_5 > c_4 > \ldots$
- $p > c_6 > c_3 > \ldots$
- $p > c_7 > c_1 > \ldots$
- $p > c_6 > c_7 > \ldots$
- $p > c_3 > c_6 > \ldots$
- $p > c_3 > c_5 > \ldots$

### Graph

![Graph](image)

### Situation
- choose $k$ votes
- (edges)
A polynomial-time result: Edge Matching algorithm

**Votes**
- \( p > c_1 > c_3 > \ldots \)
- \( p > c_2 > c_4 > \ldots \)
- \( p > c_2 > c_3 > \ldots \)
- \( p > c_5 > c_4 > \ldots \)
- \( p > c_6 > c_3 > \ldots \)
- \( p > c_7 > c_1 > \ldots \)
- \( p > c_6 > c_7 > \ldots \)
- \( p > c_3 > c_6 > \ldots \)
- \( p > c_3 > c_5 > \ldots \)

**Graph**

**Situation**
- choose \( k \) votes (edges)
- each candidate \( c_i \neq p \): bound on gained points
A polynomial-time result: Edge Matching algorithm

<table>
<thead>
<tr>
<th>Votes</th>
</tr>
</thead>
<tbody>
<tr>
<td>• (p &gt; c_1 &gt; c_3 &gt; \ldots)</td>
</tr>
<tr>
<td>• (p &gt; c_2 &gt; c_4 &gt; \ldots)</td>
</tr>
<tr>
<td>• (p &gt; c_2 &gt; c_3 &gt; \ldots)</td>
</tr>
<tr>
<td>• (p &gt; c_5 &gt; c_4 &gt; \ldots)</td>
</tr>
<tr>
<td>• (p &gt; c_6 &gt; c_3 &gt; \ldots)</td>
</tr>
<tr>
<td>• (p &gt; c_7 &gt; c_1 &gt; \ldots)</td>
</tr>
<tr>
<td>• (p &gt; c_6 &gt; c_7 &gt; \ldots)</td>
</tr>
<tr>
<td>• (p &gt; c_3 &gt; c_6 &gt; \ldots)</td>
</tr>
<tr>
<td>• (p &gt; c_3 &gt; c_5 &gt; \ldots)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Graph</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Diagram showing a graph with labeled candidates: (c_1, c_2, c_3, c_4, c_5, c_6, c_7)]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Situation</th>
</tr>
</thead>
<tbody>
<tr>
<td>• choose (k) votes (edges)</td>
</tr>
<tr>
<td>• each candidate (c_i \neq p): bound on gained points</td>
</tr>
</tbody>
</table>

PTIME-solvable Matching Problem

**Problem:** Simple \(b\)-edge matching of multigraphs

**Input:** Multigraph \(G = (E, V)\), function \(b: V \rightarrow \mathbb{N}\), number \(k\)

**Question:**
A polynomial-time result: Edge Matching algorithm

**Votes**
- $p > c_1 > c_3 > \ldots$
- $p > c_2 > c_4 > \ldots$
- $p > c_2 > c_3 > \ldots$
- $p > c_5 > c_4 > \ldots$
- $p > c_6 > c_3 > \ldots$
- $p > c_7 > c_1 > \ldots$
- $p > c_6 > c_7 > \ldots$
- $p > c_3 > c_6 > \ldots$
- $p > c_3 > c_5 > \ldots$

**Graph**

**Situation**
- choose $k$ votes (edges)
- each candidate $c_i \neq p$: bound on gained points

**PTIME-solvable Matching Problem**

**Problem:** Simple $b$-edge matching of multigraphs

**Input:** Multigraph $G = (E, V)$, function $b: V \rightarrow \mathbb{N}$, number $k$

**Question:** Is there $E' \subseteq E$ with $|E'| = k$ and each $v \in V$ is incident to at most $b(v)$ edges in $E'$?
A polynomial-time result: Edge Matching algorithm

Votes
- \( p > c_1 > c_3 > \ldots \)
- \( p > c_2 > c_4 > \ldots \)
- \( p > c_2 > c_3 > \ldots \)
- \( p > c_5 > c_4 > \ldots \)
- \( p > c_6 > c_3 > \ldots \)
- \( p > c_7 > c_1 > \ldots \)
- \( p > c_6 > c_7 > \ldots \)
- \( p > c_3 > c_6 > \ldots \)
- \( p > c_3 > c_5 > \ldots \)

Situation
- choose \( k \) votes (edges)
- each candidate \( c_i \neq p \): bound on gained points

Graph

PTIME-solvable Matching Problem

**Problem:** Simple \( b \)-edge matching of multigraphs

**Input:** Multigraph \( G = (E, V) \), function \( b: V \rightarrow \mathbb{N} \), number \( k \)

**Question:** Is there \( E' \subseteq E \) with \( |E'| = k \) and each \( v \in V \) is incident to at most \( b(v) \) edges in \( E' \)?

Other systems

**2-Veto** \((1, 1, \ldots, 1, 0, 0)\): similar.
A polynomial-time result: Edge Matching algorithm

Votes
- \( p > c_1 > c_3 > \ldots \)
- \( p > c_2 > c_4 > \ldots \)
- \( p > c_2 > c_3 > \ldots \)
- \( p > c_5 > c_4 > \ldots \)
- \( p > c_6 > c_3 > \ldots \)
- \( p > c_7 > c_1 > \ldots \)
- \( p > c_6 > c_7 > \ldots \)
- \( p > c_3 > c_6 > \ldots \)
- \( p > c_3 > c_5 > \ldots \)

Graph

Situation
- choose \( k \) votes (edges)
- each candidate \( c_i \neq p \): bound on gained points

PTIME-solvable Matching Problem

Problem: Simple \( b \)-edge matching of multigraphs
Input: Multigraph \( G = (E, V) \), function \( b: V \rightarrow \mathbb{N} \), number \( k \)
Question: Is there \( E' \subseteq E \) with \( |E'| = k \) and each \( v \in V \) is incident to at most \( b(v) \) edges in \( E' \)?

Other systems

2-Veto \((1, 1, \ldots, 1, 0, 0)\): similar.
\((\alpha, \beta, 0, \ldots, 0)\): similar.
Second polynomial-time result

“Hybrid” between 1-Approval and 1-Veto

Generator $(2, 1, \ldots, 1, 0)$
    or $(1, 0, \ldots, 0, -1)$
Second polynomial-time result

“Hybrid” between 1-Approval and 1-Veto

Generator $(2, 1, \ldots, 1, 0)$ or $(1, 0, \ldots, 0, -1)$

- Approve of 1 candidate
- Veto another

Proof

Favourite candidate $p$, voters $R$ and $U$, $k$ voters can be added
Second polynomial-time result

“Hybrid” between 1-Approval and 1-Veto

Generator \((2, 1, \ldots, 1, 0)\) or \((1, 0, \ldots, 0, -1)\)
- Approve of 1 candidate
- Veto another

Proof

Favourite candidate \(p\), voters \(R\) and \(U\), \(k\) voters can be added
- Add no voter with \(p\) in last place
Second polynomial-time result

“Hybrid” between 1-Approval and 1-Veto

Generator $(2, 1, \ldots, 1, 0)$
or $(1, 0, \ldots, 0, -1)$

- Approve of 1 candidate
- Veto another

Proof

Favourite candidate $p$, voters $R$ and $U$, $k$ voters can be added

- Add no voter with $p$ in last place
- Add all voters with $p$ in first place
  - Greedy strategy if we can’t use all
Second polynomial-time result

“Hybrid” between 1-Approval and 1-Veto

Generator \((2, 1, \ldots, 1, 0)\)
\text{or} \((1, 0, \ldots, 0, -1)\)

- Approve of 1 candidate
- Veto another

Proof

Favourite candidate \(p\), voters \(R\) and \(U\), \(k\) voters can be added

- Add no voter with \(p\) in last place
- Add all voters with \(p\) in first place
  - Greedy strategy if we can’t use all
- After this step: only votes \(c_1 > \cdots > c_2\) with \(p \notin \{c_1, c_2\}\).
Second polynomial-time result: \((1, 0, \ldots, 0, -1)\)

**Situation**
- Only votes \(c_1 > \cdots > c_2\) for \(p \notin \{c_1, c_2\}\)
- Select at most \(k'\) of these
Second polynomial-time result: \((1, 0, \ldots, 0, -1)\)

<table>
<thead>
<tr>
<th>Situation</th>
<th>Effect of Vote</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Only votes (c_1 &gt; \cdots &gt; c_2) for (p \notin {c_1, c_2})</td>
<td>Adding voter (c_1 &gt; \cdots &gt; c_2):</td>
</tr>
<tr>
<td>• Select at most (k') of these</td>
<td>• adds point to (c_1)</td>
</tr>
<tr>
<td></td>
<td>• subtracts point from (c_2)</td>
</tr>
</tbody>
</table>
Second polynomial-time result: \((1, 0, \ldots, 0, -1)\)

**Situation**
- Only votes \(c_1 > \cdots > c_2\) for \(p \notin \{c_1, c_2\}\)
- Select at most \(k'\) of these

**Effect of Vote**
Adding voter \(c_1 > \cdots > c_2\):
- adds point to \(c_1\)
- subtracts point from \(c_2\)
\(\leadsto\) “transfers” point from \(c_2\) to \(c_1\)
Second polynomial-time result: \((1, 0, \ldots, 0, -1)\)

**Situation**
- Only votes \(c_1 > \cdots > c_2\) for \(p \notin \{c_1, c_2\}\)
- Select at most \(k'\) of these

**Effect of Vote**
Adding voter \(c_1 > \cdots > c_2\):
- adds point to \(c_1\)
- subtracts point from \(c_2\)
\(\sim\) "transfers" point from \(c_2\) to \(c_1\)

**Approach**
- Vote additions “transfer” points between candidates
- Score of \(p\) is fixed (after preprocessing)
- **Goal**: For each \(c_i \neq p\), shift surplus points away
Polynomial Time via Network Flow: \((1, 0, \ldots, 0, -1)\)

**Network Setup**
- one node for each \(c_i \neq p\)
- source, target nodes \(S\) and \(T\)
Polynomial Time via Network Flow: $\langle 1, 0, \ldots, 0, -1 \rangle$

**Network Setup**
- one node for each $c_i \neq p$
- source, target nodes $S$ and $T$

**Flows in Network**
- each $c_i$: gets $\text{score}(c_i)$ points
### Network Setup
- one node for each $c_i \neq p$
- source, target nodes $S$ and $T$

### Flows in Network
- each $c_i$: gets $score(c_i)$ points
- each potential vote allows to “move” one point
  - This costs one “unit”
Polynomial Time via Network Flow: $(1, 0, \ldots, 0, -1)$

**Network Setup**
- one node for each $c_i \neq p$
- source, target nodes $S$ and $T$

**Flows in Network**
- each $c_i$: gets $\text{score}(c_i)$ points
- each potential vote allows to “move” one point
  - This costs one “unit”
- each $c_i \neq p$: at most $\text{score}(p)$ points in the end
Polynomial Time via Network Flow

Votes

Graph

Construction Votes: Allow to "move" points

Each $c_i$: points from votes

In the end: $\leq \text{score}(p)$ points

Control with $\leq k$ voters $\iff$ flow with cost $\leq k$, value $\sum c_i \neq p$ score($c_i$) exists.
Polynomial Time via Network Flow

Votes
- $c_5 > \cdots > c_1$

Graph

Construction
- Votes: Allow to "move" points
Polynomial Time via Network Flow

**Votes**
- \(c_5 > \cdots > c_1\)
- \(c_3 > \cdots > c_1\)

**Graph**

![Diagram showing a graph with nodes labeled S, C1, C2, C3, C4, C5, C6, C7, and T, with edges labeled 1 and cost 1.]

**Construction**
- Votes: Allow to “move” points
Polynomial Time via Network Flow

**Votes**
- $c_5 > \cdots > c_1$
- $c_3 > \cdots > c_1$
- $c_7 > \cdots > c_4$

**Graph**

**Construction**
- Votes: Allow to “move” points
Polynomial Time via Network Flow

Votes
- $c_5 > \cdots > c_1$
- $c_3 > \cdots > c_1$
- $c_7 > \cdots > c_4$
- $c_2 > \cdots > c_4$

Graph

Construction
- Votes: Allow to “move” points
Polynomial Time via Network Flow

**Votes**
- $c_5 > \cdots > c_1$
- $c_3 > \cdots > c_1$
- $c_7 > \cdots > c_4$
- $c_2 > \cdots > c_4$
- $c_2 > \cdots > c_6$

**Graph**

Construction
- Votes: Allow to “move” points

Edith Hemaspaandra, Lane Hemaspaandra, Henning Schnoor - A Control Dichotomy for Pure Scoring Rules
Polynomial Time via Network Flow

**Votes**
- $c_5 > \cdots > c_1$
- $c_3 > \cdots > c_1$
- $c_7 > \cdots > c_4$
- $c_2 > \cdots > c_4$
- $c_2 > \cdots > c_6$
- $c_5 > \cdots > c_6$

**Graph**
![Graph Diagram]

**Construction**
- Votes: Allow to “move” points
Polynomial Time via Network Flow

**Votes**
- \( c_5 > \cdots > c_1 \)
- \( c_3 > \cdots > c_1 \)
- \( c_7 > \cdots > c_4 \)
- \( c_2 > \cdots > c_4 \)
- \( c_2 > \cdots > c_6 \)
- \( c_5 > \cdots > c_6 \)
- \( c_1 > \cdots > c_2 \)

**Graph**

**Construction**
- Votes: Allow to “move” points
Polynomial Time via Network Flow

**Votes**
- $c_5 > \cdots > c_1$
- $c_3 > \cdots > c_1$
- $c_7 > \cdots > c_4$
- $c_2 > \cdots > c_4$
- $c_2 > \cdots > c_6$
- $c_5 > \cdots > c_6$
- $c_1 > \cdots > c_2$
- $c_3 > \cdots > c_2$

**Graph**

**Construction**
- Votes: Allow to “move” points

**Example:**
- Graph with nodes $S$, $T$, and edges with weights.
- Example construction:
  - Allow to “move” points from one candidate to another.
Polynomial Time via Network Flow

**Votes**
- $c_5 > \cdots > c_1$
- $c_3 > \cdots > c_1$
- $c_7 > \cdots > c_4$
- $c_2 > \cdots > c_4$
- $c_2 > \cdots > c_6$
- $c_5 > \cdots > c_6$
- $c_1 > \cdots > c_2$
- $c_3 > \cdots > c_2$
- $c_5 > \cdots > c_6$

**Construction**
- Votes: Allow to “move” points
Polynomial Time via Network Flow

Votes

- $c_5 > \cdots > c_1$
- $c_3 > \cdots > c_1$
- $c_7 > \cdots > c_4$
- $c_2 > \cdots > c_4$
- $c_2 > \cdots > c_6$
- $c_5 > \cdots > c_6$
- $c_1 > \cdots > c_2$
- $c_3 > \cdots > c_2$
- $c_5 > \cdots > c_6$
- $c_3 > \cdots > c_2$

Graph

Construction

- Votes: Allow to “move” points
Constructing Votes: Allow to “move” points

**Votes**
- \(c_5 > \cdots > c_1\)
- \(c_3 > \cdots > c_1\)
- \(c_7 > \cdots > c_4\)
- \(c_2 > \cdots > c_4\)
- \(c_2 > \cdots > c_6\)
- \(c_5 > \cdots > c_6\)
- \(c_1 > \cdots > c_2\)
- \(c_3 > \cdots > c_2\)
- \(c_5 > \cdots > c_6\)
- \(c_3 > \cdots > c_2\)
- \(c_5 > \cdots > c_6\)

**Graph**

[Diagram of a network flow graph]

**Construction**

- **Votes**: Allow to “move” points
Polynomial Time via Network Flow

**Votes**
- $c_5 > \cdots > c_1$
- $c_3 > \cdots > c_1$
- $c_7 > \cdots > c_4$
- $c_2 > \cdots > c_4$
- $c_2 > \cdots > c_6$
- $c_5 > \cdots > c_6$
- $c_1 > \cdots > c_2$
- $c_3 > \cdots > c_2$
- $c_5 > \cdots > c_6$
- $c_3 > \cdots > c_2$
- $c_5 > \cdots > c_6$

**Construction**
- Votes: Allow to “move” points
- Each $c_i$: points from votes

**Graph**
- In the end: $\leq \text{score}(p)$ points
- Control with $\leq k$ voters $\Leftrightarrow$ flow with cost $\leq k$, value $\sum c_i \neq p \text{ score}(c_i)$ exists.
Polynomial Time via Network Flow

**Votes**
- \( c_5 > \cdots > c_1 \)
- \( c_3 > \cdots > c_1 \)
- \( c_7 > \cdots > c_4 \)
- \( c_2 > \cdots > c_4 \)
- \( c_2 > \cdots > c_6 \)
- \( c_5 > \cdots > c_6 \)
- \( c_1 > \cdots > c_2 \)
- \( c_3 > \cdots > c_2 \)
- \( c_5 > \cdots > c_6 \)
- \( c_3 > \cdots > c_2 \)
- \( c_5 > \cdots > c_6 \)

**Graph**

**Construction**
- Votes: Allow to “move” points
- Each \( c_i \): points from votes
**Polynomial Time via Network Flow**

**Votes**
- $c_5 > \cdots > c_1$
- $c_3 > \cdots > c_1$
- $c_7 > \cdots > c_4$
- $c_2 > \cdots > c_4$
- $c_2 > \cdots > c_6$
- $c_5 > \cdots > c_6$
- $c_1 > \cdots > c_2$
- $c_3 > \cdots > c_2$
- $c_5 > \cdots > c_6$
- $c_3 > \cdots > c_2$
- $c_5 > \cdots > c_6$

**Graph**

```
S ---> score(c_5) 1 --> c_5 1 --> c_2 1 --> 1---> c_6 1 --> c_4 2 --> 1---> c_3 1---> score(c_3) 1 --> c_3
```

```
T
```

**Construction**
- Votes: Allow to “move” points
- Each $c_i$: points from votes

---

Edith Hemaspaandra, Lane Hemaspaandra, Henning Schnoor - A Control Dichotomy for Pure Scoring Rules
### Votes
- \( c_5 > \cdots > c_1 \)
- \( c_3 > \cdots > c_1 \)
- \( c_7 > \cdots > c_4 \)
- \( c_2 > \cdots > c_4 \)
- \( c_2 > \cdots > c_6 \)
- \( c_5 > \cdots > c_6 \)
- \( c_1 > \cdots > c_2 \)
- \( c_3 > \cdots > c_2 \)
- \( c_5 > \cdots > c_6 \)
- \( c_3 > \cdots > c_2 \)
- \( c_5 > \cdots > c_6 \)

### Construction
- Votes: Allow to “move” points
- Each \( c_i \): points from votes

### Graph

In the end: \( \leq \) score(\( p \)) points

Control with \( \leq k \) voters \( \iff \) flow with cost \( \leq k \), value \( \sum c_i \neq p \) score(\( c_i \)) exists.

\( S \) → cost 1
\( T \) → cost 0
Polynomial Time via Network Flow

VOTES

- $c_5 > \cdots > c_1$
- $c_3 > \cdots > c_1$
- $c_7 > \cdots > c_4$
- $c_2 > \cdots > c_4$
- $c_2 > \cdots > c_6$
- $c_5 > \cdots > c_6$
- $c_1 > \cdots > c_2$
- $c_3 > \cdots > c_2$
- $c_5 > \cdots > c_6$
- $c_3 > \cdots > c_2$
- $c_5 > \cdots > c_6$

GRAPH

Construction

- Votes: Allow to “move” points
- Each $c_i$: points from votes
- In the end: $\leq \text{score}(p)$ points
Polynomial Time via Network Flow

**Votes**
- $c_5 > \cdots > c_1$
- $c_3 > \cdots > c_1$
- $c_7 > \cdots > c_4$
- $c_2 > \cdots > c_4$
- $c_2 > \cdots > c_6$
- $c_5 > \cdots > c_6$
- $c_1 > \cdots > c_2$
- $c_3 > \cdots > c_2$
- $c_5 > \cdots > c_6$
- $c_3 > \cdots > c_2$
- $c_5 > \cdots > c_6$

**Construction**
- Votes: Allow to “move” points
- Each $c_i$: points from votes
- In the end: $\leq \text{score}(p)$ points

**Graph**

- $S$ to $T$
- $S$ to $score(p)$
- $T$ to $score(p)$
- $score(c_3)$ to $score(c_7)$
- $score(c_1)$ to $score(c_5)$
- $score(c_4)$ to $score(c_2)$
- $score(c_6)$ to $score(c_1)$
- $c_1$ to $c_2$
- $c_2$ to $c_4$
- $c_4$ to $c_7$
- $c_7$ to $c_5$
- $c_5$ to $c_6$
- $c_6$ to $c_7$
- $c_1$ to $1$
- $c_2$ to $2$
- $c_3$ to $1$
- $c_4$ to $1$
- $c_5$ to $1$
- $c_6$ to $3$
- $c_7$ to $1$

- $\rightarrow$ cost 1
- $\rightarrow$ cost 0

*Construction with $\leq k$ voters $\iff$ flow with cost $\leq k$, value $\sum c_i \neq p$ exists.*
Polynomial Time via Network Flow

**Votes**
- $c_5 > \cdots > c_1$
- $c_3 > \cdots > c_1$
- $c_7 > \cdots > c_4$
- $c_2 > \cdots > c_4$
- $c_2 > \cdots > c_6$
- $c_5 > \cdots > c_6$
- $c_1 > \cdots > c_2$
- $c_3 > \cdots > c_2$
- $c_5 > \cdots > c_6$
- $c_3 > \cdots > c_2$
- $c_5 > \cdots > c_6$

**Graph**

**Construction**
- Votes: Allow to “move” points
- Each $c_i$: points from votes
- In the end: $\leq \text{score}(p)$ points
Polynomial Time via Network Flow

**Votes**
- $c_5 > \cdots > c_1$
- $c_3 > \cdots > c_1$
- $c_7 > \cdots > c_4$
- $c_2 > \cdots > c_4$
- $c_2 > \cdots > c_6$
- $c_5 > \cdots > c_6$
- $c_1 > \cdots > c_2$
- $c_3 > \cdots > c_2$
- $c_5 > \cdots > c_6$
- $c_3 > \cdots > c_2$
- $c_5 > \cdots > c_6$
- $c_3 > \cdots > c_2$
- $c_5 > \cdots > c_6$

**Graph**

**Construction**
- Votes: Allow to “move” points
- Each $c_i$: points from votes
- In the end: $\leq \text{score}(p)$ points
Polynomial Time via Network Flow

**Votes**

- $c_5 > \cdots > c_1$
- $c_3 > \cdots > c_1$
- $c_7 > \cdots > c_4$
- $c_2 > \cdots > c_4$
- $c_2 > \cdots > c_6$
- $c_5 > \cdots > c_6$
- $c_1 > \cdots > c_2$
- $c_3 > \cdots > c_2$
- $c_5 > \cdots > c_6$
- $c_3 > \cdots > c_2$
- $c_5 > \cdots > c_6$

**Graph**

- Construction
  - Votes: Allow to “move” points
  - Each $c_i$: points from votes
  - In the end: $\leq \text{score}(p)$ points

Control with $\leq k$ voters $\iff$
flow with cost $\leq k$, value
$\sum_{c_i \neq p} \text{score}(c_i)$ exists.
Overview

1. Introduction: Elections and Control
2. The Complexity of the Control Problem
3. A Class of Election Systems: Scoring Rules
4. The Price of Purity
5. A Dichotomy Theorem
   - Polynomial Time Results
   - Hardness Results
6. Conclusion
Dichotomy

Halfway There
We proved all polynomial-time cases.
Dichotomy

Halfway There

We proved all polynomial-time cases.

Polynomial-Time

- \( (1, 1, 1, 0, \ldots, 0) \)
- \( (\alpha, \beta, 0, \ldots, 0) \)
- \( (2, 1, \ldots, 1, 0) \)
- \( (1, \ldots, 1, 0) \)
- \( (1, \ldots, 1, 0, 0) \)
Dichotomy

Halfway There
We proved all polynomial-time cases.

<table>
<thead>
<tr>
<th>Polynomial-Time</th>
<th>NP-Hardness</th>
</tr>
</thead>
<tbody>
<tr>
<td>((1, 1, 1, 0, \ldots, 0))</td>
<td>Generators not equivalent to one of these.</td>
</tr>
<tr>
<td>((\alpha, \beta, 0, \ldots, 0))</td>
<td>- no affine transformation</td>
</tr>
<tr>
<td>((2, 1, \ldots, 1, 0))</td>
<td>⇔ different result for some election.</td>
</tr>
<tr>
<td>((1, \ldots, 1, 0))</td>
<td>CCAV is NP-hard for all of these.</td>
</tr>
<tr>
<td>((1, \ldots, 1, 0, 0))</td>
<td></td>
</tr>
</tbody>
</table>
**Dichotomy**

**Halfway There**

We proved all polynomial-time cases.

<table>
<thead>
<tr>
<th>Polynomial-Time</th>
<th>NP-Hardness</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1, 1, 1, 0, \ldots, 0)</td>
<td>Generators <strong>not equivalent</strong> to one of these.</td>
</tr>
<tr>
<td>(\alpha, \beta, 0, \ldots, 0)</td>
<td>- no affine transformation</td>
</tr>
<tr>
<td>(2, 1, \ldots, 1, 0)</td>
<td>\iff different result for <strong>some</strong> election.</td>
</tr>
<tr>
<td>(1, \ldots, 1, 0)</td>
<td>CCAV is NP-hard for all of these.</td>
</tr>
<tr>
<td>(1, \ldots, 1, 0, 0)</td>
<td></td>
</tr>
</tbody>
</table>

**Hardness Proof**

Some individual cases and a general construction.
Theorem

$f$-CCAV is NP-hard for

- $(\alpha, \beta, \gamma, 0, \ldots, 0)$ for $\alpha \geq \beta \geq \gamma > 0$
Case Distinction

**Theorem**

\( f \)-CCAV is NP-hard for

- \((\alpha, \beta, \gamma, 0, \ldots, 0)\) for \(\alpha \geq \beta \geq \gamma > 0\)
- \((\alpha_1, \alpha_2, \alpha_3, \alpha_4, \ldots, \alpha_4, \alpha_5, 0)\) with \(\alpha_2 > \alpha_4 > 0\).
Case Distinction

Theorem

$f$-CCAV is NP-hard for

- $(\alpha, \beta, \gamma, 0, \ldots, 0)$ for
  $\alpha \geq \beta \geq \gamma > 0$
- $(\alpha_1, \alpha_2, \alpha_3, \alpha_4, \ldots, \alpha_4, \alpha_5, 0)$
  with $\alpha_2 > \alpha_4 > 0$.
- $(\alpha_1, \alpha_2, \ldots, \alpha_2, 0)$ with
  $\alpha_1 \notin \{\alpha_2, 2\alpha_2\}$, $\alpha_2 > 0$. 
Case Distinction

Theorem

\(f\)-CCAV is NP-hard for

- \((\alpha, \beta, \gamma, 0, \ldots, 0)\) for \(\alpha \geq \beta \geq \gamma > 0\)
- \((\alpha_1, \alpha_2, \alpha_3, \alpha_4, \ldots, \alpha_4, \alpha_5, 0)\) with \(\alpha_2 > \alpha_4 > 0\).
- \((\alpha_1, \alpha_2, \ldots, \alpha_2, 0)\) with \(\alpha_1 \notin \{2\alpha_2\}, \alpha_2 > 0\).
- \((\alpha_1, \alpha_2, \ldots, \alpha_2, \alpha_5, 0)\) with \(\alpha_1 > \alpha_2 > \alpha_5\).
Case Distinction

**Theorem**

$f$-CCAV is NP-hard for

- $(\alpha, \beta, \gamma, 0, \ldots, 0)$ for $\alpha \geq \beta \geq \gamma > 0$
- $(\alpha_1, \alpha_2, \alpha_3, \alpha_4, \ldots, \alpha_4, \alpha_5, 0)$ with $\alpha_2 > \alpha_4 > 0$.
- $(\alpha_1, \alpha_2, \ldots, \alpha_2, 0)$ with $\alpha_1 \notin \{\alpha_2, 2\alpha_2\}$, $\alpha_2 > 0$.
- $(\alpha_1, \alpha_2, \ldots, \alpha_2, \alpha_5, 0)$ with $\alpha_1 > \alpha_2 > \alpha_5$.
- $(\alpha_1, \ldots, \alpha_1, \alpha_5, 0)$ with $\alpha_1 > \alpha_5 > 0$.

**Corollary**

For a generator $f$ with $f$ is not equivalent to a generator in the list, $f$ uses at most 6 different coefficients $f$-CCAV is NP-complete.
Case Distinction

Theorem

\( f \)-CCAV is NP-hard for

- \((\alpha, \beta, \gamma, 0, \ldots, 0)\) for 
  \(\alpha \geq \beta \geq \gamma > 0\)
- \((\alpha_1, \alpha_2, \alpha_3, \alpha_4, \ldots, \alpha_4, \alpha_5, 0)\) with \(\alpha_2 > \alpha_4 > 0\).
- \((\alpha_1, \alpha_2, \ldots, \alpha_2, 0)\) with \(\alpha_1 \not\in \{ \alpha_2, 2\alpha_2 \}, \alpha_2 > 0\).
- \((\alpha_1, \alpha_2, \ldots, \alpha_2, \alpha_5, 0)\) with \(\alpha_1 > \alpha_2 > \alpha_5\).
- \((\alpha_1, \ldots, \alpha_1, \alpha_5, 0)\) with \(\alpha_1 > \alpha_5 > 0\).

Corollary

For a generator \( f \) with

- \( f \) is not equivalent to a generator in the list,
- \( f \) uses at most 6 different coefficients

\( f \)-CCAV is NP-complete.
Theorem

$f$-CCAV is NP-hard for

1. $(\alpha, \beta, \gamma, 0, \ldots, 0)$ for $\alpha \geq \beta \geq \gamma > 0$
2. $(\alpha_1, \alpha_2, \alpha_3, \alpha_4, \ldots, \alpha_4, \alpha_5, 0)$ with $\alpha_2 > \alpha_4 > 0$.
3. $(\alpha_1, \alpha_2, \ldots, \alpha_2, 0)$ with $\alpha_1 \notin \{\alpha_2, 2\alpha_2\}$, $\alpha_2 > 0$.
4. $(\alpha_1, \alpha_2, \ldots, \alpha_2, \alpha_5, 0)$ with $\alpha_1 > \alpha_2 > \alpha_5$.
5. $(\alpha_1, \ldots, \alpha_1, \alpha_5, 0)$ with $\alpha_1 > \alpha_5 > 0$.

Corollary

For a generator $f$ with

1. $f$ is not equivalent to a generator in the list,
2. $f$ uses at most 6 different coefficients

$f$-CCAV is NP-complete.

Direct reductions, point counting, combinatorics.
What happens with more than six values?

Remember “purity” condition!

- all coefficients from $f(n)$ appear in $f(n + 1)$. 
What happens with more than six values?

Remember “purity” condition!

- all coefficients from \( f(n) \) appear in \( f(n + 1) \).
- 7 different coefficients appear together in \( f(n) \) (for large enough \( n \))
What happens with more than six values?

Remember “purity” condition!

- all coefficients from $f(n)$ appear in $f(n + 1)$.
- 7 different coefficients appear together in $f(n)$ (for large enough $n$)
  - Affine transformations of different values remain different

Analysis

$$(\alpha_1, \alpha_2, \alpha_3, \alpha_4, \ldots, \alpha_{n-2}, \alpha_{n-1}, \alpha_n)$$
What happens with more than six values?

Remember “purity” condition!

- all coefficients from $f(n)$ appear in $f(n + 1)$.
- 7 different coefficients appear together in $f(n)$ (for large enough $n$)
  - Affine transformations of different values remain different

Analysis

$(\alpha_1, \alpha_2, \alpha_3, \alpha_4, \ldots, \alpha_{n-2}, \alpha_{n-1}, \alpha_n)$

- 7 values: $\alpha_4 \neq \alpha_{n-2}$
- top four positions “better” than lowest three.
- (middle positions can be “good” or “bad.”)
What happens with more than six values?

Remember “purity” condition!

- all coefficients from $f(n)$ appear in $f(n + 1)$.
- 7 different coefficients appear together in $f(n)$ (for large enough $n$)
  - Affine transformations of different values remain different

Analysis

$($α_1, α_2, α_3, α_4, \ldots, α_{n-2}, α_{n-1}, α_n)$

- 7 values: $α_4 \neq α_{n-2}$
- top four positions “better” than lowest three.
- (middle positions can be “good” or “bad.”)
A Reduction

Set Cover Problem

\[ M \subseteq X \times Y \times Z \]

with \( |X| = |Y| = |Z| \)
A Reduction

Set Cover Problem

\[ M \subseteq X \times Y \times Z \]
with \( |X| = |Y| = |Z| \)

Choose \( |X| \) elements from \( M \) that agree in no component
A Reduction

Set Cover Problem

$M \subseteq X \times Y \times Z$

with $|X| = |Y| = |Z|$

Choose $|X|$ elements from $M$
that agree in no component

Control Problem

Points from registered voters,
Voters $U$, all vote $p$ first
A Reduction

Set Cover Problem

\[ M \subseteq X \times Y \times Z \]
with \(|X| = |Y| = |Z|\)

Choose \(|X|\) elements from \(M\) that agree in no component

Control Problem

Points from registered voters, Voters \(U\), all vote \(p\) first

Choose \(k\) \(U\)-votes such that nobody has more points than \(p\)
### A Reduction

<table>
<thead>
<tr>
<th>Set Cover Problem</th>
<th>Control Problem</th>
</tr>
</thead>
</table>
| $M \subseteq X \times Y \times Z$  
with $|X| = |Y| = |Z|$  
Choose $|X|$ elements from $M$  
that agree in no component | Points from registered voters,  
Voters $U$, all vote $p$ first  
Choose $k$ $U$-votes such that  
nobody has more points than $p$ |

<table>
<thead>
<tr>
<th>Instance</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(x_1, y_1, z_1)$</td>
<td>$p &gt; x_1 &gt; y_1 &gt; z_1 &gt; \ldots$</td>
</tr>
</tbody>
</table>
A Reduction

**Set Cover Problem**

\[ M \subseteq X \times Y \times Z \]

with \( |X| = |Y| = |Z| \)

Choose \( |X| \) elements from \( M \) that agree in no component

**Control Problem**

Points from registered voters, Voters \( U \), all vote \( p \) first

Choose \( k \) \( U \)-votes such that nobody has more points than \( p \)

**Instance**

- \((x_1, y_1, z_1)\)
- \((x_2, y_2, z_2)\)

**Instance**

- \( p > x_1 > y_1 > z_1 > \ldots \)
- \( p > x_2 > y_2 > z_2 > \ldots \)
## A Reduction

### Set Cover Problem

- **Problem:**
  - \( M \subseteq X \times Y \times Z \)
  - \(|X| = |Y| = |Z|\)
  - Choose \(|X|\) elements from \( M \) that agree in no component

### Control Problem

- **Problem:**
  - Points from registered voters, Voters \( U \), all vote \( p \) first
  - Choose \( k \) \( U \)-votes such that nobody has more points than \( p \)

### Instance

<table>
<thead>
<tr>
<th>Set Cover Problem</th>
<th>Control Problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>((x_1, y_1, z_1))</td>
<td>( p &gt; x_1 &gt; y_1 &gt; z_1 &gt; \ldots )</td>
</tr>
<tr>
<td>((x_2, y_2, z_2))</td>
<td>( p &gt; x_2 &gt; y_2 &gt; z_2 &gt; \ldots )</td>
</tr>
<tr>
<td>((x_1, y_3, z_2))</td>
<td>( p &gt; x_1 &gt; y_3 &gt; z_2 &gt; \ldots )</td>
</tr>
</tbody>
</table>
## A Reduction

### Set Cover Problem

\[ M \subseteq X \times Y \times Z \]

with \(|X| = |Y| = |Z|\)

Choose \(|X|\) elements from \(M\) that agree in no component

### Control Problem

Points from registered voters, Voters \(U\), all vote \(p\) first

Choose \(k\) \(U\)-votes such that nobody has more points than \(p\)

### Instance

- \((x_1, y_1, z_1)\)
- \((x_2, y_2, z_2)\)
- \((x_1, y_3, z_2)\)
- \((x_4, y_3, z_4)\)

### Instance

- \(p > x_1 > y_1 > z_1 > \ldots\)
- \(p > x_2 > y_2 > z_2 > \ldots\)
- \(p > x_1 > y_3 > z_2 > \ldots\)
- \(p > x_4 > y_3 > z_4 > \ldots\)
A Reduction

Set Cover Problem

\[ M \subseteq X \times Y \times Z \]
with \(|X| = |Y| = |Z|\)

Choose \(|X|\) elements from \(M\) that agree in no component

Control Problem

Points from registered voters, Voters \(U\), all vote \(p\) first

Choose \(k\) \(U\)-votes such that nobody has more points than \(p\)

Instance

- \((x_1, y_1, z_1)\)
- \((x_2, y_2, z_2)\)
- \((x_1, y_3, z_2)\)
- \((x_4, y_3, z_4)\)

Instance

- \(p > x_1 > y_1 > z_1 > \ldots\)
- \(p > x_2 > y_2 > z_2 > \ldots\)
- \(p > x_1 > y_3 > z_2 > \ldots\)
- \(p > x_4 > y_3 > z_4 > \ldots\)

Set up points: • Each candidate may be voted only once in “good spot.”
## A Reduction

### Set Cover Problem

*INSTANCE*

\[ M \subseteq X \times Y \times Z \]

with \( |X| = |Y| = |Z| \)

Choose \( |X| \) elements from \( M \) that agree in no component.

### Control Problem

*INSTANCE*

Choose \( k \) \( U \)-votes such that nobody has more points than \( p \).

### Instance

- \((x_1, y_1, z_1)\)
- \((x_2, y_2, z_2)\)
- \((x_1, y_3, z_2)\)
- \((x_4, y_3, z_4)\)

### Set up points:
- Each candidate may be voted only once in “good spot.”
- \(|X|\) votes required for \( p \) to win against “dummy”
- Construction needs careful “padding”
Overview

1 Introduction: Elections and Control
2 The Complexity of the Control Problem
3 A Class of Election Systems: Scoring Rules
4 The Price of Purity
5 A Dichotomy Theorem
   • Polynomial Time Results
   • Hardness Results
6 Conclusion
Conclusion

Contributions

- Flexible Generators: Ensures we get all cases
Conclusion

Contributions

- **Flexible Generators**: Ensures we get all cases
- **Dichotomy**: Complexity of $f$-CCA$^V$ for flexible generators $f$
  - solvable in polynomial time, or
  - NP-complete
Conclusion

Contributions

- Flexible Generators: Ensures we get all cases
- Dichotomy: Complexity of $f$-CCA\(V\) for flexible generators $f$
  - solvable in polynomial time, or
  - NP-complete

Open Questions

- Complexity of CCA\(V\) for other voting systems?
Conclusion

Contributions

- Flexible Generators: Ensures we get all cases
- Dichotomy: Complexity of $f$-CCAV for flexible generators $f$
  - solvable in polynomial time, or
  - NP-complete

Open Questions

- Complexity of CCAV for other voting systems?
- Complexity of related problems
  - control by deleting candidates
Conclusion

Contributions

- Flexible Generators: Ensures we get all cases
- Dichotomy: Complexity of $f$-CCAV for flexible generators $f$
  - solvable in polynomial time, or
  - NP-complete

Open Questions

- Complexity of CCAV for other voting systems?
- Complexity of related problems
  - control by deleting candidates
  - bribery

Edith Hemaspaandra, Lane Hemaspaandra, Henning Schnoor - A Control Dichotomy for Pure Scoring Rules
Conclusion

Contributions

- Flexible Generators: Ensures we get all cases
- Dichotomy: Complexity of $f$-CCAV for flexible generators $f$
  - solvable in polynomial time, or
  - NP-complete

Open Questions

- Complexity of CCAV for other voting systems?
- Complexity of related problems
  - control by deleting candidates
  - bribery
  - manipulation
Conclusion

Contributions

- Flexible Generators: Ensures we get all cases
- Dichotomy: Complexity of $f$-CCAV for flexible generators $f$
  - solvable in polynomial time, or
  - NP-complete

Open Questions

- Complexity of CCAV for other voting systems?
- Complexity of related problems
  - control by deleting candidates
  - bribery
  - manipulation

Thank You!