

Condorcet Computations

Johannes Grabmeier

University of Applied Sciences Deggendorf

Florence, 16.07.2009

Content

- 1 The Ranking Problem
- 2 The Condorcet Problem
- 3 Lehmer Code
- 4 Cusanus and Hungarian Upper Bounds
- 5 Implementation
- 6 Outlook: Schulze's paths

Content

- 1 The Ranking Problem
- 2 The Condorcet Problem
- 3 Lehmer Code
- 4 Cusanus and Hungarian Upper Bounds
- 5 Implementation
- 6 Outlook: Schulze's paths

- Given: a finite number t of **voters**.
- A finite number n of **candidates** labelled by 1 to n stand for an election.
- The voters **totally rank** the n candidates,
- The **election** π is given by t **permutations**

$$\pi_k \in S_n, 1 \leq k \leq t$$

as elements of the symmetric group.

- Vote π_k places candidate $\pi_k(1)$ first, $\pi_k(2)$ second and so on: $\pi_k(i)$ is the number of the candidate placed at rank i in the ordering given by vote π_k .
- Or: the placement of candidate q of vote π is $\pi^{-1}(q)$.
- **Problem:** define and find out the overall result of such an election.

- Given: a finite number t of **voters**.
- A finite number n of **candidates** labelled by 1 to n stand for an election.
- The voters **totally rank** the n candidates,
- The **election** π is given by t **permutations**

$$\pi_k \in S_n, 1 \leq k \leq t$$

as elements of the symmetric group.

- Vote π_k places candidate $\pi_k(1)$ first, $\pi_k(2)$ second and so on: $\pi_k(i)$ is the number of the candidate placed at rank i in the ordering given by vote π_k .
- Or: the placement of candidate q of vote π is $\pi^{-1}(q)$.
- **Problem:** define and find out the overall result of such an election.

- Given: a finite number t of **voters**.
- A finite number n of **candidates** labelled by 1 to n stand for an election.
- The voters **totally rank** the n candidates,
- The **election** π is given by t **permutations**

$$\pi_k \in S_n, 1 \leq k \leq t$$

as elements of the symmetric group.

- Vote π_k places candidate $\pi_k(1)$ first, $\pi_k(2)$ second and so on: $\pi_k(i)$ is the number of the candidate placed at rank i in the ordering given by vote π_k .
- Or: the placement of candidate q of vote π is $\pi^{-1}(q)$.
- **Problem:** define and find out the overall result of such an election.

- Given: a finite number t of **voters**.
- A finite number n of **candidates** labelled by 1 to n stand for an election.
- The voters **totally rank** the n candidates,
- The **election** π is given by t **permutations**

$$\pi_k \in S_n, 1 \leq k \leq t$$

as elements of the symmetric group.

- Vote π_k places candidate $\pi_k(1)$ first, $\pi_k(2)$ second and so on: $\pi_k(i)$ is the number of the candidate placed at rank i in the ordering given by vote π_k .
- Or: the placement of candidate q of vote π is $\pi^{-1}(q)$.
- **Problem:** define and find out the overall result of such an election.

- Given: a finite number t of **voters**.
- A finite number n of **candidates** labelled by 1 to n stand for an election.
- The voters **totally rank** the n candidates,
- The **election** π is given by t **permutations**

$$\pi_k \in S_n, 1 \leq k \leq t$$

as elements of the symmetric group.

- Vote π_k places candidate $\pi_k(1)$ first, $\pi_k(2)$ second and so on: $\pi_k(i)$ is the number of the candidate placed at rank i in the ordering given by vote π_k .
- Or: the placement of candidate q of vote π is $\pi^{-1}(q)$.
- **Problem:** define and find out the overall result of such an election.

- Given: a finite number t of **voters**.
- A finite number n of **candidates** labelled by 1 to n stand for an election.
- The voters **totally rank** the n candidates,
- The **election** π is given by t **permutations**

$$\pi_k \in S_n, 1 \leq k \leq t$$

as elements of the symmetric group.

- Vote π_k places candidate $\pi_k(1)$ first, $\pi_k(2)$ second and so on: $\pi_k(i)$ is the number of the candidate placed at rank i in the ordering given by vote π_k .
- Or: the placement of candidate q of vote π is $\pi^{-1}(q)$.
- **Problem:** define and find out the overall result of such an election.

- Given: a finite number t of **voters**.
- A finite number n of **candidates** labelled by 1 to n stand for an election.
- The voters **totally rank** the n candidates,
- The **election** π is given by t **permutations**

$$\pi_k \in S_n, 1 \leq k \leq t$$

as elements of the symmetric group.

- Vote π_k places candidate $\pi_k(1)$ first, $\pi_k(2)$ second and so on: $\pi_k(i)$ is the number of the candidate placed at rank i in the ordering given by vote π_k .
- Or: the placement of candidate q of vote π is $\pi^{-1}(q)$.
- **Problem**: define and find out the overall result of such an election.

Cusanus suggested aggregation of points in 1433



- Nicolaus Cusanus, 1401-1464,
- Roman Catholic cardinal, German philosopher and mathematician.
- Give points, say from $n - 1$ down to 0 to the descending ranked candidates of a vote.
- Add all points to get an overall ranking.

Cusanus suggested aggregation of points in 1433



- Nicolaus Cusanus, 1401-1464,
- Roman Catholic cardinal, German philosopher and mathematician.
- Give points, say from $n - 1$ down to 0 to the descending ranked candidates of a vote.
- Add all points to get an overall ranking.

Cusanus suggested aggregation of points in 1433



- Nicolaus Cusanus, 1401-1464,
- Roman Catholic cardinal, German philosopher and mathematician.
- Give points, say from $n - 1$ down to 0 to the descending ranked candidates of a vote.
- Add all points to get an overall ranking.

Cusanus suggested aggregation of points in 1433



- Nicolaus Cusanus, 1401-1464,
- Roman Catholic cardinal, German philosopher and mathematician.
- Give points, say from $n - 1$ down to 0 to the descending ranked candidates of a vote.
- Add all points to get an overall ranking.

Quote from Cusanus: De Concordancia Catholica (On the general harmony)

Et tunc in dei nomine secundum suam conscientiam ponderet, quis inter illos omnes **minus idoneus** exsistat, et signet cum puncto incausti supra **primum numerum** simplicem longum punctum et post hoc iudicet, quis **post illum minus idoneus**, et signet **secundum numerum** cum puncto longo simplici et sic continue, quousque veniet **ad optimum** suo iudicio, et ibi signabit **decimum numerum** aut illum numerum, qui numero personarum correspondebit.

...Quibus expletis colligat per **additionem** computista numeros cuiuslibet nominis. Et ille erit tunc imperator, qui **maio rem numerum** habuit.

In the name of God he should ponder, directed by his conscience, who among all candidates is least qualified, and place a simple long mark in ink above the number 1. Thereafter he should decide who is next least suitable, and mark the number 2 with a simple long overline. Thus he continues until he arrives at the best, in his judgment, and there he will mark the number 10, or generally the number corresponding to the number of candidates.

...When all ballots are recorded, the teller should add up the numbers next to each name. The candidate who has the highest total shall be king.

Cusanus: formal definition

Definition

- Candidate q is placed at position $\pi_k^{-1}(q)$ and therefore gets $n - \pi_k^{-1}(q)$ points.
- Define

$$C(\pi_k) := (n - \pi_k^{-1}(1), \dots, (n - \pi_k^{-1}(n)))$$

- and sum over all votes: $\sum_{1 \leq k \leq t} C(\pi_k)$
- Get at least one $\kappa \in \mathcal{S}_n$ with property (for $1 \leq i < n$)

$$\sum_{1 \leq k \leq t} \left(n - \pi_k^{-1}(\kappa(i)) \right) \geq \sum_{1 \leq k \leq t} \left(n - \pi_k^{-1}(\kappa(i+1)) \right)$$

Cusanus: formal definition

Definition

- Candidate q is placed at position $\pi_k^{-1}(q)$ and therefore gets $n - \pi_k^{-1}(q)$ points.
- Define

$$C(\pi_k) := (n - \pi_k^{-1}(1), \dots, (n - \pi_k^{-1}(n)))$$

- and sum over all votes: $\sum_{1 \leq k \leq t} C(\pi_k)$
- Get at least one $\kappa \in \mathcal{S}_n$ with property (for $1 \leq i < n$)

$$\sum_{1 \leq k \leq t} (n - \pi_k^{-1}(\kappa(i))) \geq \sum_{1 \leq k \leq t} (n - \pi_k^{-1}(\kappa(i+1)))$$

Cusanus: formal definition

Definition

- Candidate q is placed at position $\pi_k^{-1}(q)$ and therefore gets $n - \pi_k^{-1}(q)$ points.
- Define

$$C(\pi_k) := (n - \pi_k^{-1}(1), \dots, (n - \pi_k^{-1}(n)))$$

- and sum over all votes: $\sum_{1 \leq k \leq t} C(\pi_k)$
- Get at least one $\kappa \in \mathcal{S}_n$ with property (for $1 \leq i < n$)

$$\sum_{1 \leq k \leq t} \left(n - \pi_k^{-1}(\kappa(i)) \right) \geq \sum_{1 \leq k \leq t} \left(n - \pi_k^{-1}(\kappa(i+1)) \right)$$

Cusanus: formal definition

Definition

- Candidate q is placed at position $\pi_k^{-1}(q)$ and therefore gets $n - \pi_k^{-1}(q)$ points.
- Define

$$C(\pi_k) := (n - \pi_k^{-1}(1), \dots, (n - \pi_k^{-1}(n)))$$

- and sum over all votes: $\sum_{1 \leq k \leq t} C(\pi_k)$
- Get at least one $\kappa \in S_n$ with property (for $1 \leq i < n$)

$$\sum_{1 \leq k \leq t} (n - \pi_k^{-1}(\kappa(i))) \geq \sum_{1 \leq k \leq t} (n - \pi_k^{-1}(\kappa(i+1)))$$

Cusanus: formal definition

Definition

- Candidate q is placed at position $\pi_k^{-1}(q)$ and therefore gets $n - \pi_k^{-1}(q)$ points.
- Define

$$C(\pi_k) := (n - \pi_k^{-1}(1), \dots, (n - \pi_k^{-1}(n)))$$

- and sum over all votes: $\sum_{1 \leq k \leq t} C(\pi_k)$
- Get at least one $\kappa \in S_n$ with property (for $1 \leq i < n$)

$$\sum_{1 \leq k \leq t} (n - \pi_k^{-1}(\kappa(i))) \geq \sum_{1 \leq k \leq t} (n - \pi_k^{-1}(\kappa(i+1)))$$

A defect of the Cusanus method

A **later cancellation** of a — say last ranked candidate — can lead to a **change** of the candidate placed at **rank 1**.

Example

	$t = 5, n = 6 :$	$t = 5, n = 5 :$
π_1	1, 2, 3, 6 , 5, 4	1, 2, 3, 5, 4
π_2	1, 4, 6 , 2, 5, 3	1, 4, 2, 5, 3
π_3	1, 4, 6 , 2, 3, 5	1, 4, 2, 3, 5
π_4	2, 3, 5, 1, 4, 6	2, 3, 5, 1, 4
π_5	2, 5, 3, 1, 4, 6	2, 5, 3, 1, 4
$\sum_k C(\pi_k)$	19 , 18, 11, 10, 9, 8	14, 15 , 8, 6, 7
κ	1, 2, 3, 4, 5, 6	2, 1, 3, 5, 4

Content

- 1 The Ranking Problem
- 2 The Condorcet Problem**
- 3 Lehmer Code
- 4 Cusanus and Hungarian Upper Bounds
- 5 Implementation
- 6 Outlook: Schulze's paths

Borda vs. Condorcet

- Today, this method is named after **Jean-Charles de Borda**.
- In the 18-th century in France his scientific opponent was **Nicolas Marquis de Condorcet**.

Borda vs. Condorcet

- Today, this method is named after **Jean-Charles de Borda**.
- In the 18-th century in France his scientific opponent was **Nicolas Marquis de Condorcet**.

Nicolas de Condorcet



- Marie Jean Antoine **Nicolas** de Caritat, Marquis **de Condorcet** (1743-1794),
 - a French philosopher, mathematician, and early political scientist.
 - Each ranking votes for $\binom{n}{2}$ **pair preferences**.
 - A permutation is the outcome of the voting, if its $\binom{n}{2}$ pairing preferences earn a maximal number of votes.

Nicolas de Condorcet



- Marie Jean Antoine **Nicolas** de Caritat, Marquis **de Condorcet** (1743-1794),
- a French philosopher, mathematician, and early political scientist.
- Each ranking votes for $\binom{n}{2}$ **pair preferences**.
- A permutation is the outcome of the voting, if its $\binom{n}{2}$ pairing preferences earn a maximal number of votes.

Nicolas de Condorcet



- Marie Jean Antoine **Nicolas** de Caritat, Marquis **de Condorcet** (1743-1794),
- a French philosopher, mathematician, and early political scientist.
- Each ranking votes for $\binom{n}{2}$ **pair preferences**.
- A permutation is the outcome of the voting, if its $\binom{n}{2}$ pairing preferences earn a maximal number of votes.

Nicolas de Condorcet



- Marie Jean Antoine **Nicolas** de Caritat, Marquis **de Condorcet** (1743-1794),
- a French philosopher, mathematician, and early political scientist.
- Each ranking votes for $\binom{n}{2}$ **pair preferences**.
- A permutation is the outcome of the voting, if its $\binom{n}{2}$ pairing preferences earn a maximal number of votes.

The Condorcet problem

Definition

- A vote π_k **supports** an ordered pair (p, q) of candidates, $1 \leq p, q \leq n, p \neq q$, if it places p before q , i.e. if $\pi_k^{-1}(p) < \pi_k^{-1}(q)$.
- For given t votes $\pi_k \in S_n, 1 \leq k \leq t$ the number of **Condorcet votes** of a permutation $\rho \in S_n$ is defined to be

$$v(\rho) := \sum_{1 \leq k \leq t} \sum_{1 \leq p < q \leq n} [\pi_k^{-1}(\rho(p)) < \pi_k^{-1}(\rho(q))]^1 \quad (1)$$

- The **Condorcet problem** is to find a solution to the optimization problem of maximizing v on S_n .

¹ $[b]$ evaluates to 1, if the condition b is true, otherwise to 0.

The Condorcet problem

Definition

- A vote π_k **supports** an ordered pair (p, q) of candidates, $1 \leq p, q \leq n, p \neq q$, if it places p before q , i.e. if $\pi_k^{-1}(p) < \pi_k^{-1}(q)$.
- For given t votes $\pi_k \in S_n, 1 \leq k \leq t$ the number of **Condorcet votes** of a permutation $\rho \in S_n$ is defined to be

$$v(\rho) := \sum_{1 \leq k \leq t} \sum_{1 \leq p < q \leq n} [\pi_k^{-1}(\rho(p)) < \pi_k^{-1}(\rho(q))]^1 \quad (1)$$

- The **Condorcet problem** is to find a solution to the optimization problem of maximizing v on S_n .

¹ $[b]$ evaluates to 1, if the condition b is true, otherwise to 0.

The Condorcet problem

Definition

- A vote π_k **supports** an ordered pair (p, q) of candidates, $1 \leq p, q \leq n, p \neq q$, if it places p before q , i.e. if $\pi_k^{-1}(p) < \pi_k^{-1}(q)$.
- For given t votes $\pi_k \in S_n, 1 \leq k \leq t$ the number of **Condorcet votes** of a permutation $\rho \in S_n$ is defined to be

$$v(\rho) := \sum_{1 \leq k \leq t} \sum_{1 \leq p < q \leq n} [\pi_k^{-1}(\rho(p)) < \pi_k^{-1}(\rho(q))]^1 \quad (1)$$

- The **Condorcet problem** is to find a solution to the optimization problem of maximizing v on S_n .

¹ $[b]$ evaluates to 1, if the condition b is true, otherwise to 0.

The Condorcet problem

Definition

- A vote π_k **supports** an ordered pair (p, q) of candidates, $1 \leq p, q \leq n, p \neq q$, if it places p before q , i.e. if $\pi_k^{-1}(p) < \pi_k^{-1}(q)$.
- For given t votes $\pi_k \in S_n, 1 \leq k \leq t$ the number of **Condorcet votes** of a permutation $\rho \in S_n$ is defined to be

$$v(\rho) := \sum_{1 \leq k \leq t} \sum_{1 \leq p < q \leq n} [\pi_k^{-1}(\rho(p)) < \pi_k^{-1}(\rho(q))]^1 \quad (1)$$

- The **Condorcet problem** is to find a solution to the optimization problem of maximizing v on S_n .

¹ $[b]$ evaluates to 1, if the condition b is true, otherwise to 0.

The Condorcet paradox

Example

	$t = 5, n = 6 :$	v		$t = 5, n = 5 :$	v
	1, 2, 3, 6 , 5, 4			1, 2, 3, 5, 4	
	1, 4, 6 , 2, 5, 3			1, 4, 2, 5, 3	
	1, 4, 6 , 2, 3, 5			1, 4, 2, 3, 5	
	2, 3, 5, 1, 4, 6			2, 3, 5, 1, 4	
	2, 5, 3, 1, 4, 6			2, 5, 3, 1, 4	
$\rho_{6,1} :=$	1, 2, 3, 4, 6, 5	53	$\rho_{5,1} :=$	1, 2, 3, 5, 4	36
$\rho_{6,2} :=$	1, 2, 3, 5, 4, 6	53	$\rho_{5,2} :=$	1, 2, 3, 4, 5	35

$n = 6$: 2 solutions for Condorcet problem, $\rho_{6,1}$ and $\rho_{6,2}$,
 $n = 5$: unique

Content

- 1 The Ranking Problem
- 2 The Condorcet Problem
- 3 Lehmer Code**
- 4 Cusanus and Hungarian Upper Bounds
- 5 Implementation
- 6 Outlook: Schulze's paths

Complexity of computing Condorcet votes

- A standard implementation for computing the Condorcet votes for one ranking $\rho \in S_n$ alone requires $O(t n^2)$ operations.
- Can one do better?
- Yes: use Lehmer to code the permutations.

Complexity of computing Condorcet votes

- A standard implementation for computing the Condorcet votes for one ranking $\rho \in S_n$ alone requires $O(t n^2)$ operations.
- Can one do **better**?
- Yes: use **Lehmer** to code the permutations.

Complexity of computing Condorcet votes

- A standard implementation for computing the Condorcet votes for one ranking $\rho \in S_n$ alone requires $O(t n^2)$ operations.
- Can one do **better**?
- Yes: use **Lehmer** to code the permutations.

Lehmer code I

- A permutation π from S_n can be coded by a vector from $\underline{n} \times \underline{n-1} \times \dots \times \underline{1}$, where $\underline{i} := \{0, 1, \dots, i-1\}$.
- To establish such a bijection, for $\pi \in S_n$ and $1 \leq i \leq n$ the **inversions** (i, j) , i.e. the pairs with $i < j \leq n$ and $\pi(i) > \pi(j)$ and denote this number by $l_i(\pi)$ are counted.
- The permutation

$$\pi := [\pi(1), \pi(2), \dots, \pi(n)] := \begin{pmatrix} 1 & 2 & \dots & n \\ \pi(1) & \pi(2) & \dots & \pi(n) \end{pmatrix}$$

is coded by the sequence of these numbers

$$L(\pi) := \overline{\{i < j : \pi(i) > \pi(j)\}}_{1 \leq i \leq n}.$$

- Example in S_9 :

$$L([8, 5, 6, 3, 2, 7, 4, 1, 9]) = \overline{7, 4, 4, 2, 1, 2, 1, 0, 0}.$$

Lehmer code I

- A permutation π from S_n can be coded by a vector from $\underline{n} \times \underline{n-1} \times \dots \times \underline{1}$, where $\underline{i} := \{0, 1, \dots, i-1\}$.
- To establish such a bijection, for $\pi \in S_n$ and $1 \leq i \leq n$ the **inversions** (i, j) , i.e. the pairs with $i < j \leq n$ and $\pi(i) > \pi(j)$ and denote this number by $l_i(\pi)$ are counted.
- The permutation

$$\pi := [\pi(1), \pi(2), \dots, \pi(n)] := \begin{pmatrix} 1 & 2 & \dots & n \\ \pi(1) & \pi(2) & \dots & \pi(n) \end{pmatrix}$$

is coded by the sequence of these numbers

$$L(\pi) := \overline{|\{i < j : \pi(i) > \pi(j)\}|}_{1 \leq i \leq n}.$$

- Example in S_9 :

$$L([8, 5, 6, 3, 2, 7, 4, 1, 9]) = \overline{7, 4, 4, 2, 1, 2, 1, 0, 0}.$$

Lehmer code I

- A permutation π from S_n can be coded by a vector from $\underline{n} \times \underline{n-1} \times \dots \times \underline{1}$, where $\underline{i} := \{0, 1, \dots, i-1\}$.
- To establish such a bijection, for $\pi \in S_n$ and $1 \leq i \leq n$ the **inversions** (i, j) , i.e. the pairs with $i < j \leq n$ and $\pi(i) > \pi(j)$ and denote this number by $l_i(\pi)$ are counted.
- The permutation

$$\pi := [\pi(1), \pi(2), \dots, \pi(n)] := \begin{pmatrix} 1 & 2 & \dots & n \\ \pi(1) & \pi(2) & \dots & \pi(n) \end{pmatrix}$$

is coded by the sequence of these numbers

$$L(\pi) := \overline{|\{i < j : \pi(i) > \pi(j)\}|}_{1 \leq i \leq n}.$$

- Example in S_9 :

$$L([8, 5, 6, 3, 2, 7, 4, 1, 9]) = \overline{7, 4, 4, 2, 1, 2, 1, 0, 0}.$$

Lehmer code I

- A permutation π from S_n can be coded by a vector from $\underline{n} \times \underline{n-1} \times \dots \times \underline{1}$, where $\underline{i} := \{0, 1, \dots, i-1\}$.
- To establish such a bijection, for $\pi \in S_n$ and $1 \leq i \leq n$ the **inversions** (i, j) , i.e. the pairs with $i < j \leq n$ and $\pi(i) > \pi(j)$ and denote this number by $l_i(\pi)$ are counted.
- The permutation

$$\pi := [\pi(1), \pi(2), \dots, \pi(n)] := \begin{pmatrix} 1 & 2 & \dots & n \\ \pi(1) & \pi(2) & \dots & \pi(n) \end{pmatrix}$$

is coded by the sequence of these numbers

$$L(\pi) := \overline{|\{i < j : \pi(i) > \pi(j)\}|}_{1 \leq i \leq n}.$$

- Example in S_9 :

$$L([8, 5, 6, 3, 2, 7, 4, 1, 9]) = \overline{7, 4, 4, 2, 1, 2, 1, 0, 0}.$$

Recall properties of the Lehmer code

- Let an **elementary transposition** $s_i := (i, i + 1)$ operate on π with $L(\pi) = \overline{l_1, \dots, l_n}$:

- descent $l_i > l_{i+1}$ ($\Leftrightarrow \pi(i) > \pi(i + 1)$)

$$\Rightarrow L(\pi \cdot s_i) = \overline{l_1, \dots, l_{i-1}, l_{i+1}, l_i - 1, \dots, l_n}$$

- weak ascent $l_i \leq l_{i+1}$ ($\Leftrightarrow \pi(i) < \pi(i + 1)$)

$$\Rightarrow L(\pi \cdot s_i) = \overline{l_1, \dots, l_{i-1}, l_{i+1} + 1, l_i, l_{i+2}, \dots, l_n}$$

- **Lehmer degree** $|L| := \sum_{1 \leq i \leq n} l_i$ decreases by 1 (ascent) resp. increases by 1 (weak descent).

- ... **Bruhat order** with ranking function **Lehmer degree**.

- ... shortest length representation of π as a product of elementary transpositions.

Recall properties of the Lehmer code

- Let an **elementary transposition** $s_i := (i, i + 1)$ operate on π with $L(\pi) = \overline{l_1, \dots, l_n}$:

- descent $l_i > l_{i+1}$ ($\Leftrightarrow \pi(i) > \pi(i + 1)$)

$$\Rightarrow L(\pi \cdot s_i) = \overline{l_1, \dots, l_{i-1}, l_{i+1}, l_i - 1, \dots, l_n}$$

- weak ascent $l_i \leq l_{i+1}$ ($\Leftrightarrow \pi(i) < \pi(i + 1)$)

$$\Rightarrow L(\pi \cdot s_i) = \overline{l_1, \dots, l_{i-1}, l_{i+1} + 1, l_i, l_{i+2}, \dots, l_n}$$

- **Lehmer degree** $|L| := \sum_{1 \leq i \leq n} l_i$ decreases by 1 (ascent) resp. increases by 1 (weak descent).

- ... **Bruhat order** with ranking function **Lehmer degree**.

- ... shortest length representation of π as a product of elementary transpositions.

Recall properties of the Lehmer code

- Let an **elementary transposition** $s_i := (i, i + 1)$ operate on π with $L(\pi) = \overline{l_1, \dots, l_n}$:

- descent $l_i > l_{i+1}$ ($\Leftrightarrow \pi(i) > \pi(i + 1)$)

$$\Rightarrow L(\pi \cdot s_i) = \overline{l_1, \dots, l_{i-1}, l_{i+1}, l_i - 1, \dots, l_n}$$

- weak ascent $l_i \leq l_{i+1}$ ($\Leftrightarrow \pi(i) < \pi(i + 1)$)

$$\Rightarrow L(\pi \cdot s_i) = \overline{l_1, \dots, l_{i-1}, l_{i+1} + 1, l_i, l_{i+2}, \dots, l_n}$$

- **Lehmer degree** $|L| := \sum_{1 \leq i \leq n} l_i$ decreases by 1 (ascent) resp. increases by 1 (weak descent).

- ... **Bruhat order** with ranking function **Lehmer degree**.

- ... shortest length representation of π as a product of elementary transpositions.

Recall properties of the Lehmer code

- Let an **elementary transposition** $s_i := (i, i + 1)$ operate on π with $L(\pi) = \overline{l_1, \dots, l_n}$:

- descent $l_i > l_{i+1}$ ($\Leftrightarrow \pi(i) > \pi(i + 1)$)

$$\Rightarrow L(\pi \cdot s_i) = \overline{l_1, \dots, l_{i-1}, l_{i+1}, l_i - 1, \dots, l_n}$$

- weak ascent $l_i \leq l_{i+1}$ ($\Leftrightarrow \pi(i) < \pi(i + 1)$)

$$\Rightarrow L(\pi \cdot s_i) = \overline{l_1, \dots, l_{i-1}, l_{i+1} + 1, l_i, l_{i+2}, \dots, l_n}$$

- **Lehmer degree** $|L| := \sum_{1 \leq i \leq n} l_i$ decreases by 1 (ascent) resp. increases by 1 (weak descent).

- ... **Bruhat order** with ranking function **Lehmer degree**.
- ... shortest length representation of π as a product of elementary transpositions.

Recall properties of the Lehmer code

- Let an **elementary transposition** $s_i := (i, i + 1)$ operate on π with $L(\pi) = \overline{l_1, \dots, l_n}$:

- descent $l_i > l_{i+1}$ ($\Leftrightarrow \pi(i) > \pi(i + 1)$)

$$\Rightarrow L(\pi \cdot s_i) = \overline{l_1, \dots, l_{i-1}, l_{i+1}, l_i - 1, \dots, l_n}$$

- weak ascent $l_i \leq l_{i+1}$ ($\Leftrightarrow \pi(i) < \pi(i + 1)$)

$$\Rightarrow L(\pi \cdot s_i) = \overline{l_1, \dots, l_{i-1}, l_{i+1} + 1, l_i, l_{i+2}, \dots, l_n}$$

- **Lehmer degree** $|L| := \sum_{1 \leq i \leq n} l_i$ decreases by 1 (ascent) resp. increases by 1 (weak descent).

- ... **Bruhat order** with ranking function **Lehmer degree**.

- ... shortest length representation of π as a product of elementary transpositions.

Recall properties of the Lehmer code

- Let an **elementary transposition** $s_i := (i, i + 1)$ operate on π with $L(\pi) = \overline{l_1, \dots, l_n}$:

- descent $l_i > l_{i+1}$ ($\Leftrightarrow \pi(i) > \pi(i + 1)$)

$$\Rightarrow L(\pi \cdot s_i) = \overline{l_1, \dots, l_{i-1}, l_{i+1}, l_i - 1, \dots, l_n}$$

- weak ascent $l_i \leq l_{i+1}$ ($\Leftrightarrow \pi(i) < \pi(i + 1)$)

$$\Rightarrow L(\pi \cdot s_i) = \overline{l_1, \dots, l_{i-1}, l_{i+1} + 1, l_i, l_{i+2}, \dots, l_n}$$

- **Lehmer degree** $|L| := \sum_{1 \leq i \leq n} l_i$ decreases by 1 (ascent) resp. increases by 1 (weak descent).

- ... **Bruhat order** with ranking function **Lehmer degree**.

- ... shortest length representation of π as a product of elementary transpositions.

Voting with Lehmer Code

Lemma

The number $v_{\pi_k}(\rho)$ of votes which a permutation $\rho \in S_n$ receives from a vote $\pi_k \in S_n$ can be expressed by the Lehmer code:

$$v_{\pi_k}(\rho) = \binom{n}{2} - |L(\pi_k^{-1}\rho)| = \binom{n}{2} - |L(\rho^{-1}\pi_k)| \quad (2)$$

Example

We work out an example for $n = 9$ and $t = 6$:

number k	vote π_k	Lehmer code $L(\pi_k)$
1	[8, 5, 6, 3, 2, 7, 4, 1, 9]	7, 4, 4, 2, 1, 2, 1, 0, 0
2	[1, 6, 8, 4, 2, 9, 7, 5, 3]	0, 4, 5, 2, 0, 3, 2, 1, 0
3	[4, 7, 6, 1, 5, 9, 2, 3, 8]	3, 5, 4, 0, 2, 3, 0, 0, 0
4	[7, 4, 6, 3, 2, 1, 5, 8, 9]	6, 3, 4, 2, 1, 0, 0, 0, 0
5	[1, 3, 4, 6, 8, 9, 5, 7, 2]	0, 1, 1, 2, 3, 3, 1, 1, 0
6	[4, 8, 7, 3, 2, 5, 6, 9, 1]	3, 6, 5, 2, 1, 1, 1, 1, 0

The Cusanus ranking is $\kappa := [4, 6, 7, 8, 1, 3, 5, 2, 9]$. We compute its number of Condorcet's votes to be $143 = 216 - 73$, as $6 \cdot \binom{9}{2} = 216$, and $\left(|L(\pi_k^{-1} \rho)| \right)_{1 \leq k \leq 6}$ is given here:

number k	$\pi_k^{-1} \rho$	Lehmer code $L(\pi_k^{-1} \rho)$	degree $ L(\pi_k^{-1} \rho) $
1	[7, 3, 6, 1, 8, 4, 2, 5, 9]	6, 2, 4, 0, 3, 1, 0, 0, 0	16
2	[4, 2, 7, 3, 1, 9, 8, 5, 6]	3, 1, 4, 1, 0, 3, 2, 0, 0	14
3	[1, 3, 2, 9, 4, 8, 5, 7, 6]	0, 1, 0, 5, 0, 3, 0, 1, 0	10
4	[2, 3, 1, 8, 6, 4, 7, 5, 9]	1, 1, 0, 4, 2, 0, 1, 0, 0	9
5	[3, 4, 8, 5, 1, 2, 7, 9, 6]	2, 2, 5, 2, 0, 0, 1, 1, 0	13
6	[1, 7, 3, 2, 9, 4, 6, 5, 8]	0, 5, 1, 0, 4, 0, 1, 0, 0	11
sum:			73

Efficient computation of the Condorcet votes

- Natural representation of a vote π is by the vector $[\pi(1), \dots, \pi(n)]$ of images.
- Hence, an obvious implementation requires $\binom{n}{2}$ operations.
- **Challenge:** can one do better with only $O(n \log(n))$ steps?
- Solution: Modify classical '**merge sort**'-algorithm, a standard sorting algorithm of $O(n \log(n))$ -complexity.
- Idea: add hashtables of number of descents.

Efficient computation of the Condorcet votes

- Natural representation of a vote π is by the vector $[\pi(1), \dots, \pi(n)]$ of images.
- Hence, an obvious implementation requires $\binom{n}{2}$ operations.
- **Challenge:** can one do better with only $O(n \log(n))$ steps?
- Solution: Modify classical '**merge sort**'-algorithm, a standard sorting algorithm of $O(n \log(n))$ -complexity.
- Idea: add hashtables of number of descents.

Efficient computation of the Condorcet votes

- Natural representation of a vote π is by the vector $[\pi(1), \dots, \pi(n)]$ of images.
- Hence, an obvious implementation requires $\binom{n}{2}$ operations.
- **Challenge:** can one do better with only $O(n \log(n))$ steps?
- Solution: Modify classical 'merge sort'-algorithm, a standard sorting algorithm of $O(n \log(n))$ -complexity.
- Idea: add hashtables of number of descents.

Efficient computation of the Condorcet votes

- Natural representation of a vote π is by the vector $[\pi(1), \dots, \pi(n)]$ of images.
- Hence, an obvious implementation requires $\binom{n}{2}$ operations.
- **Challenge:** can one do better with only $O(n \log(n))$ steps?
- Solution: Modify classical '**merge sort**'-algorithm, a standard sorting algorithm of $O(n \log(n))$ -complexity.
- Idea: add hashtables of number of descents.

Efficient computation of the Condorcet votes

- Natural representation of a vote π is by the vector $[\pi(1), \dots, \pi(n)]$ of images.
- Hence, an obvious implementation requires $\binom{n}{2}$ operations.
- **Challenge:** can one do better with only $O(n \log(n))$ steps?
- Solution: Modify classical '**merge sort**'-algorithm, a standard sorting algorithm of $O(n \log(n))$ -complexity.
- Idea: add hashtables of number of descents.

The algorithm `mergeLehmer` I: `split`

- As usual **split** the vector of n images in two parts of equal lengths in case $n = 2m$, of lengths m and $m + 1$ in case $n = 2m + 1$.
- W.l.o.g. we can assume that after a split the two parts are permutations of the listed images.
- The algorithm is applied **recursively** to both parts.
- For **easy access** we store the Lehmer code in hash tables. The keys are the images, the values are the number of descents.
- In case $n = 1$ and a vector $[\pi(i)]$ for some i , the list $[\pi(i)]$ and the hash table $[[\pi(i), 0]]$ is returned.

The algorithm `mergeLehmer` I: `split`

- As usual **split** the vector of n images in two parts of equal lengths in case $n = 2m$, of lengths m and $m + 1$ in case $n = 2m + 1$.
- W.l.o.g. we can assume that after a split the two parts are permutations of the listed images.
- The algorithm is applied **recursively** to both parts.
- For **easy access** we store the Lehmer code in hash tables. The keys are the images, the values are the number of descents.
- In case $n = 1$ and a vector $[\pi(i)]$ for some i , the list $[\pi(i)]$ and the hash table $[[\pi(i), 0]]$ is returned.

The algorithm `mergeLehmer` I: split

- As usual **split** the vector of n images in two parts of equal lengths in case $n = 2m$, of lengths m and $m + 1$ in case $n = 2m + 1$.
- W.l.o.g. we can assume that after a split the two parts are permutations of the listed images.
- The algorithm is applied **recursively** to both parts.
- For **easy access** we store the Lehmer code in hash tables. The keys are the images, the values are the number of descents.
- In case $n = 1$ and a vector $[\pi(i)]$ for some i , the list $[\pi(i)]$ and the hash table $[[\pi(i), 0]]$ is returned.

The algorithm `mergeLehmer` I: split

- As usual **split** the vector of n images in two parts of equal lengths in case $n = 2m$, of lengths m and $m + 1$ in case $n = 2m + 1$.
- W.l.o.g. we can assume that after a split the two parts are permutations of the listed images.
- The algorithm is applied **recursively** to both parts.
- For **easy access** we store the Lehmer code in hash tables. The keys are the images, the values are the number of descents.
- In case $n = 1$ and a vector $[\pi(i)]$ for some i , the list $[\pi(i)]$ and the hash table $[[\pi(i), 0]]$ is returned.

The algorithm `mergeLehmer` I: split

- As usual **split** the vector of n images in two parts of equal lengths in case $n = 2m$, of lengths m and $m + 1$ in case $n = 2m + 1$.
- W.l.o.g. we can assume that after a split the two parts are permutations of the listed images.
- The algorithm is applied **recursively** to both parts.
- For **easy access** we store the Lehmer code in hash tables. The keys are the images, the values are the number of descents.
- In case $n = 1$ and a vector $[\pi(i)]$ for some i , the list $[\pi(i)]$ and the hash table $[[\pi(i), 0]]$ is returned.

The algorithm `mergeLehmer` II: merge

The fusion of the results of the recursive applications to the the left and right parts happens in the following **merging** step:

- We receive a **left** array of decreasing elements (case $n = 2m$)

$$[\pi(i_1) > \pi(i_2) > \dots > \pi(i_m)]$$

and a hash table

$$[[\pi(i_k), l(i_k)], 1 \leq k \leq m]$$

for the Lehmer code of the corresponding permutation of this bijection of order m .

- Same with indices j_k for the **right** part.

- Case 1: $\pi(i_m) > \pi(j_1)$

- Concatenate the left and right vector.
- Copy right hash table and add the pairs $[\pi(i_k), l(i_k) + m]$ as there are m further inversions from whole right part.

The algorithm `mergeLehmer` II: merge

The fusion of the results of the recursive applications to the the left and right parts happens in the following **merging** step:

- We receive a **left** array of decreasing elements (case $n = 2m$)

$$[\pi(i_1) > \pi(i_2) > \dots > \pi(i_m)]$$

and a hash table

$$[[\pi(i_k), l(i_k)], 1 \leq k \leq m]$$

for the Lehmer code of the corresponding permutation of this bijection of order m .

- Same with indices j_k for the **right** part.

- Case 1: $\pi(i_m) > \pi(j_1)$

- Concatenate the left and right vector.
- Copy right hash table and add the pairs $[\pi(i_k), l(i_k) + m]$ as there are m further inversions from whole right part.

The algorithm `mergeLehmer` II: merge

The fusion of the results of the recursive applications to the the left and right parts happens in the following **merging** step:

- We receive a **left** array of decreasing elements (case $n = 2m$)

$$[\pi(i_1) > \pi(i_2) > \dots > \pi(i_m)]$$

and a hash table

$$[[\pi(i_k), l(i_k)], 1 \leq k \leq m]$$

for the Lehmer code of the corresponding permutation of this bijection of order m .

- Same with indices j_k for the **right** part.
- Case 1: $\pi(i_m) > \pi(j_1)$
 - Concatenate the left and right vector.
 - Copy right hash table and add the pairs $[\pi(i_k), l(i_k) + m]$ as there are m further inversions from whole right part.

The algorithm `mergeLehmer` II: merge

The fusion of the results of the recursive applications to the the left and right parts happens in the following **merging** step:

- We receive a **left** array of decreasing elements (case $n = 2m$)

$$[\pi(i_1) > \pi(i_2) > \dots > \pi(i_m)]$$

and a hash table

$$[[\pi(i_k), l(i_k)], 1 \leq k \leq m]$$

for the Lehmer code of the corresponding permutation of this bijection of order m .

- Same with indices j_k for the **right** part.
- Case 1: $\pi(i_m) > \pi(j_1)$
 - Concatenate the left and right vector.
 - Copy right hash table and add the pairs $[\pi(i_k), l(i_k) + m]$ as there are m further inversions from whole right part.

The algorithm `mergeLehmer II`: merge

The fusion of the results of the recursive applications to the the left and right parts happens in the following **merging** step:

- We receive a **left** array of decreasing elements (case $n = 2m$)

$$[\pi(i_1) > \pi(i_2) > \dots > \pi(i_m)]$$

and a hash table

$$[[\pi(i_k), l(i_k)], 1 \leq k \leq m]$$

for the Lehmer code of the corresponding permutation of this bijection of order m .

- Same with indices j_k for the **right** part.
- Case 1: $\pi(i_m) > \pi(j_1)$
 - Concatenate the left and right vector.
 - Copy right hash table and add the pairs $[\pi(i_k), l(i_k) + m]$ as there are m further inversions from whole right part.

The algorithm mergeLehmer III: merge

- Case 2: $\pi(i_m) \leq \pi(j_1)$:
 - Compare the two first elements — say $(\pi(i_l), \pi(j_r))$ — as long as this is possible. Copy hash table, start empty new list.
 - Always put larger element to the new list and cancel it from left or right list.
 - If $\pi(i_l) > (\pi(j_r)$ then put $[\pi(i_l), l(i_l) + m_r]$ into the hash table, m_r is current length of the right list.

The algorithm `mergeLehmer III`: merge

- Case 2: $\pi(i_m) \leq \pi(j_1)$:
 - Compare the two first elements — say $(\pi(i_l), \pi(j_r))$ — as long as this is possible. Copy right hash table, start empty new list.
 - Always put larger element to the new list and cancel it from left or right list.
 - If $\pi(i_l) > (\pi(j_r)$ then put $[\pi(i_l), l(i_l) + m_r]$ into the hash table, m_r is current length of the right list.

The algorithm mergeLehmer III: merge

- Case 2: $\pi(i_m) \leq \pi(j_1)$:
 - Compare the two first elements — say $(\pi(i_l), \pi(j_r))$ — as long as this is possible. Copy right hash table, start empty new list.
 - Always put larger element to the new list and cancel it from left or right list.
 - If $\pi(i_l) > (\pi(j_r)$ then put $[\pi(i_l), l(i_l) + m_r]$ into the hash table, m_r is current length of the right list.

The algorithm `mergeLehmer III`: merge

- Case 2: $\pi(i_m) \leq \pi(j_1)$:
 - Compare the two first elements — say $(\pi(i_l), \pi(j_r))$ — as long as this is possible. Copy hash table, start empty new list.
 - Always put larger element to the new list and cancel it from left or right list.
 - If $\pi(i_l) > (\pi(j_r)$ then put $[\pi(i_l), l(i_l) + m_r]$ into the hash table, m_r is current length of the right list.

Condorcet



The mergeLehmer algorithm

Algorithm 1: A merge sort like algorithm to compute the Lehmer code of a permutation.

Input: $\pi := [\pi(1), \dots, \pi(n)] \in \mathbb{N}^n$ with $\pi(i) \neq \pi(j)$ for $i \neq j$.

Output: The list π sorted in decreasing order. The Lehmer code of π , i.e.

$$L(\pi) = |\{j \geq i : \pi(i) > \pi(j)\}|_{1 \leq i \leq n}.$$

if $n = 1$ **then**

 Return $[\pi(1)]$ and $L(\pi) = \bar{0}$.

else

$m := n \text{ quo } 2$, split π into $\pi_l := [\pi(1), \dots, \pi(m)]$, $\pi_r := [\pi(m+1), \dots, \pi(n)]$;

 Recursively apply `mergeLehmer` to both parts and get two sorted lists s_l, s_r of the elements in π_l and π_r and their Lehmer codes $L(\pi_l)$ and $L(\pi_r)$;

if $\pi_l(m) > \pi_r(1)$ **then**

 Concatenate s_l and s_r to get $s = [n, n-1, \dots, 2, 1]$;

 Merge the two tables by taking all entries of the right table, and all entries of the left table, but the values increased by $n - m$;

else

 Use an algorithm `mergeLehmerInternal` to merge the two lists and their table entries by comparing the current pairs of first entries.

Ideas for improvements

- How can a permutation $\rho \in S_n$ be modified such that the result receives **more** Condorcet votes from vote π_k ?
- Idea: reduce the Lehmer degree of

$$L(\pi_k^{-1} \rho) = \overline{l_1(\pi_k^{-1} \rho), \dots, l_n(\pi_k^{-1} \rho)}$$

- This happens with an increase of 1 by multiplication from right with an elementary transpositions $s_i = (i, i+1)$ for $1 \leq i < n$, if there is a strict descent

$$l_i(\pi_k^{-1} \rho) > l_{i+1}(\pi_k^{-1} \rho).$$

- Consider all votes simultaneously: count number of descents and subtract number of weak descents.

Ideas for improvements

- How can a permutation $\rho \in S_n$ be modified such that the result receives **more** Condorcet votes from vote π_k ?
- Idea: reduce the Lehmer degree of

$$L(\pi_k^{-1} \rho) = \overline{l_1(\pi_k^{-1} \rho), \dots, l_n(\pi_k^{-1} \rho)}$$

- This happens with an increase of 1 by multiplication from right with an elementary transpositions $s_i = (i, i+1)$ for $1 \leq i < n$, if there is a strict descent

$$l_i(\pi_k^{-1} \rho) > l_{i+1}(\pi_k^{-1} \rho).$$

- Consider all votes simultaneously: count number of descents and subtract number of weak descents.

Ideas for improvements

- How can a permutation $\rho \in S_n$ be modified such that the result receives **more** Condorcet votes from vote π_k ?
- Idea: reduce the Lehmer degree of

$$L(\pi_k^{-1} \rho) = \overline{l_1(\pi_k^{-1} \rho), \dots, l_n(\pi_k^{-1} \rho)}$$

- This happens with an increase of 1 by multiplication from right with an elementary transpositions $s_i = (i, i + 1)$ for $1 \leq i < n$, if there is a strict descent

$$l_i(\pi_k^{-1} \rho) > l_{i+1}(\pi_k^{-1} \rho).$$

- Consider all votes simultaneously: count number of descents and subtract number of weak descents.

Ideas for improvements

- How can a permutation $\rho \in S_n$ be modified such that the result receives **more** Condorcet votes from vote π_k ?
- Idea: reduce the Lehmer degree of

$$L(\pi_k^{-1} \rho) = \overline{l_1(\pi_k^{-1} \rho), \dots, l_n(\pi_k^{-1} \rho)}$$

- This happens with an increase of 1 by multiplication from right with an elementary transpositions $s_i = (i, i+1)$ for $1 \leq i < n$, if there is a strict descent

$$l_i(\pi_k^{-1} \rho) > l_{i+1}(\pi_k^{-1} \rho).$$

- Consider all votes simultaneously: count number of descents and subtract number of weak descents.

Improve lemma

Lemma

For given t votes $\pi_k \in S_n$, $1 \leq k \leq t$ the number of **Condorcet votes** of a permutation $\rho \in S_n$ is changed for

$$\rho' := \rho \cdot (i, i+1)$$

by

$$\begin{aligned} \delta_i &:= \sum_{1 \leq k \leq t} \left([l_i(\pi_k^{-1} \rho) > l_{i+1}(\pi_k^{-1} \rho)] - [l_i(\pi_k^{-1} \rho) \leq l_{i+1}(\pi_k^{-1} \rho)] \right) \\ &= 2 \left(\sum_{1 \leq k \leq t} [l_i(\pi_k^{-1} \rho) > l_{i+1}(\pi_k^{-1} \rho)] \right) - t. \end{aligned}$$

Hence, use indices i with maximal positive value δ_i to increase the number of votes.

Example for improvements

Continue the example from S_9 :

- Improve, if possible, the Cusanus ranking

$$\kappa = [4, 6, 7, 8, 1, 3, 5, 2, 9].$$

- Sum of differences is

$$[-2, 0, 2, 2, 0, 0, 0, -2].$$

- Therefore,

$$\kappa \cdot (3, 4) = [4, 6, 8, 7, 1, 3, 5, 2, 9]$$

and

$$\kappa \cdot (4, 5) = [4, 6, 7, 1, 8, 3, 5, 2, 9]$$

have a total of 145 Condorcet votes.

- No further improvements by a single elementary transposition is possible.
- 145 is the optimal number, altogether 23 solutions.

Example for improvements

Continue the example from S_9 :

- Improve, if possible, the Cusanus ranking

$$\kappa = [4, 6, 7, 8, 1, 3, 5, 2, 9].$$

- Sum of differences is

$$[-2, 0, 2, 2, 0, 0, 0, -2].$$

- Therefore,

$$\kappa \cdot (3, 4) = [4, 6, 8, 7, 1, 3, 5, 2, 9]$$

and

$$\kappa \cdot (4, 5) = [4, 6, 7, 1, 8, 3, 5, 2, 9]$$

have a total of 145 Condorcet votes.

- No further improvements by a single elementary transposition is possible.
- 145 is the optimal number, altogether 23 solutions.

Example for improvements

Continue the example from S_9 :

- Improve, if possible, the Cusanus ranking

$$\kappa = [4, 6, 7, 8, 1, 3, 5, 2, 9].$$

- Sum of differences is

$$[-2, 0, 2, 2, 0, 0, 0, -2].$$

- Therefore,

$$\kappa \cdot (3, 4) = [4, 6, 8, 7, 1, 3, 5, 2, 9]$$

and

$$\kappa \cdot (4, 5) = [4, 6, 7, 1, 8, 3, 5, 2, 9]$$

have a total of 145 Condorcet votes.

- No further improvements by a single elementary transposition is possible.
- 145 is the optimal number, altogether 23 solutions.

Example for improvements

Continue the example from S_9 :

- Improve, if possible, the Cusanus ranking

$$\kappa = [4, 6, 7, 8, 1, 3, 5, 2, 9].$$

- Sum of differences is

$$[-2, 0, 2, 2, 0, 0, 0, -2].$$

- Therefore,

$$\kappa \cdot (3, 4) = [4, 6, 8, 7, 1, 3, 5, 2, 9]$$

and

$$\kappa \cdot (4, 5) = [4, 6, 7, 1, 8, 3, 5, 2, 9]$$

have a total of 145 Condorcet votes.

- No further improvements by a single elementary transposition is possible.
- 145 is the optimal number, altogether 23 solutions.

Example for improvements

Continue the example from S_9 :

- Improve, if possible, the Cusanus ranking

$$\kappa = [4, 6, 7, 8, 1, 3, 5, 2, 9].$$

- Sum of differences is

$$[-2, 0, 2, 2, 0, 0, 0, -2].$$

- Therefore,

$$\kappa \cdot (3, 4) = [4, 6, 8, 7, 1, 3, 5, 2, 9]$$

and

$$\kappa \cdot (4, 5) = [4, 6, 7, 1, 8, 3, 5, 2, 9]$$

have a total of 145 Condorcet votes.

- No further improvements by a single elementary transposition is possible.
- 145 is the optimal number, altogether 23 solutions.

Heuristic to compute near to optimal Condorcet solutions

Algorithm 2: A heuristic for a near to optimal Condorcet solution

Input: An election $\pi := (\pi_k)_{1 \leq k \leq t} \in S_n^t$

Output: A permutation ρ with ('many') Condorcets votes $v_\pi(\rho)$

Compute a Cusanus ranking κ and set $\rho := \kappa$;

Compute $(\rho^{-1} \cdot \pi_k)_{1 \leq k \leq t}$, represent the permutations in Lehmer code;

Compute sum of differences of numbers of strict ascents and weak descents

$\delta := (\delta_k)_{1 \leq k \leq t}$;

If all $\delta_k \leq 0$ then stop, otherwise find index i with maximal positive δ_i and compute

$$\rho := \rho \cdot (i, i + 1).$$

Iterate with $\rho := \rho'$;

Content

- 1 The Ranking Problem
- 2 The Condorcet Problem
- 3 Lehmer Code
- 4 Cusanus and Hungarian Upper Bounds**
- 5 Implementation
- 6 Outlook: Schulze's paths

An upper bound for the Condorcet votes I

- Let $(\pi_k) \in S_n^t$ be an election and $\rho \in S_n$. **How many** Condorcet pair agreements can ρ receive from one vote π_k ?
- For every candidate q the Cusanus points C_q records the the number of candidates placed behind him.
- **Fix a candidate q** placed by ρ at position i , i.e. $\rho(i) = q$.
- Bound 1: The ranking ρ has placed exactly $C(\rho)_q = (n - \rho^{-1}(q))$ candidates behind candidate q .
- Bound 2: The ranking π_k has placed exactly $C(\pi_k)_q = (n - \pi_k^{-1}(q))$ candidates behind candidate q .
- Hence π_k can contribute at most

$$\min \left(n - \rho^{-1}(q), n - \pi_k^{-1}(q) \right) = \min \left(n - i, n - \pi_k^{-1}(\rho(i)) \right)$$

votes.

An upper bound for the Condorcet votes I

- Let $(\pi_k) \in S_n^t$ be an election and $\rho \in S_n$. **How many** Condorcet pair agreements can ρ receive from one vote π_k ?
- For every candidate q the Cusanus points C_q records the the number of candidates placed behind him.
- **Fix a candidate q** placed by ρ at position i , i.e. $\rho(i) = q$.
- Bound 1: The ranking ρ has placed exactly $C(\rho)_q = (n - \rho^{-1}(q))$ candidates behind candidate q .
- Bound 2: The ranking π_k has placed exactly $C(\pi_k)_q = (n - \pi_k^{-1}(q))$ candidates behind candidate q .
- Hence π_k can contribute at most

$$\min \left(n - \rho^{-1}(q), n - \pi_k^{-1}(q) \right) = \min \left(n - i, n - \pi_k^{-1}(\rho(i)) \right)$$

votes.

An upper bound for the Condorcet votes I

- Let $(\pi_k) \in S_n^t$ be an election and $\rho \in S_n$. **How many** Condorcet pair agreements can ρ receive from one vote π_k ?
- For every candidate q the Cusanus points C_q records the the number of candidates placed behind him.
- **Fix a candidate q** placed by ρ at position i , i.e. $\rho(i) = q$.
- Bound 1: The ranking ρ has placed exactly $C(\rho)_q = (n - \rho^{-1}(q))$ candidates behind candidate q .
- Bound 2: The ranking π_k has placed exactly $C(\pi_k)_q = (n - \pi_k^{-1}(q))$ candidates behind candidate q .
- Hence π_k can contribute at most

$$\min \left(n - \rho^{-1}(q), n - \pi_k^{-1}(q) \right) = \min \left(n - i, n - \pi_k^{-1}(\rho(i)) \right)$$

votes.

An upper bound for the Condorcet votes I

- Let $(\pi_k) \in S_n^t$ be an election and $\rho \in S_n$. **How many** Condorcet pair agreements can ρ receive from one vote π_k ?
- For every candidate q the Cusanus points C_q records the the number of candidates placed behind him.
- **Fix a candidate q** placed by ρ at position i , i.e. $\rho(i) = q$.
- **Bound 1:** The ranking ρ has placed exactly $C(\rho)_q = (n - \rho^{-1}(q))$ candidates behind candidate q .
- **Bound 2:** The ranking π_k has placed exactly $C(\pi_k)_q = (n - \pi_k^{-1}(q))$ candidates behind candidate q .
- Hence π_k can contribute at most

$$\min \left(n - \rho^{-1}(q), n - \pi_k^{-1}(q) \right) = \min \left(n - i, n - \pi_k^{-1}(\rho(i)) \right)$$

votes.

An upper bound for the Condorcet votes I

- Let $(\pi_k) \in S_n^t$ be an election and $\rho \in S_n$. **How many** Condorcet pair agreements can ρ receive from one vote π_k ?
- For every candidate q the Cusanus points C_q records the the number of candidates placed behind him.
- **Fix a candidate q** placed by ρ at position i , i.e. $\rho(i) = q$.
- Bound 1: The ranking ρ has placed exactly $C(\rho)_q = (n - \rho^{-1}(q))$ candidates behind candidate q .
- Bound 2: The ranking π_k has placed exactly $C(\pi_k)_q = (n - \pi_k^{-1}(q))$ candidates behind candidate q .
- Hence π_k can contribute at most

$$\min \left(n - \rho^{-1}(q), n - \pi_k^{-1}(q) \right) = \min \left(n - i, n - \pi_k^{-1}(\rho(i)) \right)$$

votes.

An upper bound for the Condorcet votes I

- Let $(\pi_k) \in S_n^t$ be an election and $\rho \in S_n$. **How many** Condorcet pair agreements can ρ receive from one vote π_k ?
- For every candidate q the Cusanus points C_q records the the number of candidates placed behind him.
- **Fix a candidate q** placed by ρ at position i , i.e. $\rho(i) = q$.
- Bound 1: The ranking ρ has placed exactly $C(\rho)_q = (n - \rho^{-1}(q))$ candidates behind candidate q .
- Bound 2: The ranking π_k has placed exactly $C(\pi_k)_q = (n - \pi_k^{-1}(q))$ candidates behind candidate q .
- Hence π_k can contribute at most

$$\min \left(n - \rho^{-1}(q), n - \pi_k^{-1}(q) \right) = \min \left(n - i, n - \pi_k^{-1}(\rho(i)) \right)$$

votes.

An upper bound for the Condorcet votes II

■ Set

$$\sigma_i(\rho) := \sum_{1 \leq k \leq t} \min \left(n - i, n - \pi_k^{-1}(\rho(i)) \right),$$

and put together

$$v(\rho) \leq \sum_{1 \leq i \leq n} \sigma_i(\rho).$$

- To get a bound for an **unknown optimal ranking μ** instead of ρ , we have the problem that the inner sum $\sigma_i(\mu)$ depends on the unknown candidate $\mu(i)$ put at place i .

An upper bound for the Condorcet votes II

- Set

$$\sigma_i(\rho) := \sum_{1 \leq k \leq t} \min \left(n - i, n - \pi_k^{-1}(\rho(i)) \right),$$

and put together

$$v(\rho) \leq \sum_{1 \leq i \leq n} \sigma_i(\rho).$$

- To get a bound for an **unknown optimal ranking μ** instead of ρ , we have the problem that the inner sum $\sigma_i(\mu)$ depends on the unknown candidate $\mu(i)$ put at place i .

An upper bound for the Condorcet votes III

- Hence, take the maximum over all possible candidates q being placed at position i by an optimal μ :

$$\sigma_i(\mu) \leq \max_{1 \leq q \leq n} \left(\sum_{1 \leq k \leq t} \min \left(n - i, n - \pi_k^{-1}(q) \right) \right).$$

- Taking in account that

$$\min \left(n - i, n - \pi_k^{-1}(q) \right) = n - \max \left(i, \pi_k^{-1}(q) \right)$$

- we have proved the following lemma:

An upper bound for the Condorcet votes III

- Hence, take the maximum over all possible candidates q being placed at position i by an optimal μ :

$$\sigma_i(\mu) \leq \max_{1 \leq q \leq n} \left(\sum_{1 \leq k \leq t} \min \left(n - i, n - \pi_k^{-1}(q) \right) \right).$$

- Taking in account that

$$\min \left(n - i, n - \pi_k^{-1}(q) \right) = n - \max \left(i, \pi_k^{-1}(q) \right)$$

- we have proved the following lemma:

An upper bound for the Condorcet votes III

- Hence, take the maximum over all possible candidates q being placed at position i by an optimal μ :

$$\sigma_i(\mu) \leq \max_{1 \leq q \leq n} \left(\sum_{1 \leq k \leq t} \min \left(n - i, n - \pi_k^{-1}(q) \right) \right).$$

- Taking in account that

$$\min \left(n - i, n - \pi_k^{-1}(q) \right) = n - \max \left(i, \pi_k^{-1}(q) \right)$$

- we have proved the following lemma:

Condorcet's upper Cusanus bound lemma

Lemma

Let $\pi_k \in S_n$ for $1 \leq k \leq t$ and $\mu \in S_n$ be an optimal Condorcet solution for these t ranking votes. Then

$$\begin{aligned} v(\mu) &\leq \sum_{1 \leq i \leq n} \max_{1 \leq q \leq n} \left(\sum_{1 \leq k \leq t} \min \left(n - i, n - \pi_k^{-1}(q) \right) \right) \\ &= n^2 t - \sum_{1 \leq i \leq n} \min_{1 \leq q \leq n} \left(\sum_{1 \leq k \leq t} \max \left(i, \pi_k^{-1}(q) \right) \right). \end{aligned}$$

Example of a Cusanus upper bound

$$(\sigma_{i,q}) := \left(\sum_{1 \leq k \leq t} \min(n - i, n - \pi_k^{-1}(q)) \right)_{1 \leq i \leq n, 1 \leq q \leq n} :$$

$n - i$	[1,	2,	3,	4,	5,	6,	7,	8,	9]	row max
8	[25,	18,	23,	36,	19,	32,	27,	26,	10]	36
7	[23,	18,	23,	34,	19,	32,	26,	25,	10]	34
6	[21,	18,	22,	31,	18,	31,	24,	23,	10]	31
5	[19,	18,	21,	27,	17,	27,	21,	20,	10]	27
4	[16,	18,	17,	22,	16,	22,	18,	17,	10]	22
3	[13,	14,	13,	17,	14,	17,	15,	13,	10]	17
2	[9,	10,	9,	12,	11,	12,	11,	9,	7]	12
1	[5,	5,	5,	6,	6,	6,	6,	5,	4]	6
0	[0,	0,	0,	0,	0,	0,	0,	0,	0]	0
Cusanus sum of last column:										185

The optimal Condorcet rankings received 145 votes, the bound is 185.

A sharper Hungarian bound

- We can improve the bound further if we spend $O(n^3)$ operations.
- Recall that we estimated $\sigma_i(\mu) \leq \max_{1 \leq q \leq n} \sigma_{i,q}$ where $\sigma_{i,q} := \left(\sum_{1 \leq k \leq t} \min(n - i, n - \pi_k^{-1}(q)) \right)$.
- For a concrete solution μ of the Condorcet problem we could do better namely

$$v_{\pi}(\mu) \leq \sum_{1 \leq i \leq n} \sigma_{i,\mu(i)}.$$

- To find a better bound as before we have to solve the **assignment problem**, i.e. to find a permutation ρ where this sum is maximal.
- The **Hungarian algorithm** from combinatorial optimization solves this problem in $O(n^3)$ steps.

A sharper Hungarian bound

- We can improve the bound further if we spend $O(n^3)$ operations.
- Recall that we estimated $\sigma_i(\mu) \leq \max_{1 \leq q \leq n} \sigma_{i,q}$ where $\sigma_{i,q} := \left(\sum_{1 \leq k \leq t} \min(n - i, n - \pi_k^{-1}(q)) \right)$.
- For a concrete solution μ of the Condorcet problem we could do better namely

$$v_\pi(\mu) \leq \sum_{1 \leq i \leq n} \sigma_{i,\mu(i)}.$$

- To find a better bound as before we have to solve the **assignment problem**, i.e. to find a permutation ρ where this sum is maximal.
- The **Hungarian algorithm** from combinatorial optimization solves this problem in $O(n^3)$ steps.

A sharper Hungarian bound

- We can improve the bound further if we spend $O(n^3)$ operations.
- Recall that we estimated $\sigma_i(\mu) \leq \max_{1 \leq q \leq n} \sigma_{i,q}$ where $\sigma_{i,q} := \left(\sum_{1 \leq k \leq t} \min \left(n - i, n - \pi_k^{-1}(q) \right) \right)$.
- For a concrete solution μ of the Condorcet problem we could do better namely

$$v_\pi(\mu) \leq \sum_{1 \leq i \leq n} \sigma_{i,\mu(i)}.$$

- To find a better bound as before we have to solve the **assignment problem**, i.e. to find a permutation ρ where this sum is maximal.
- The **Hungarian algorithm** from combinatorial optimization solves this problem in $O(n^3)$ steps.

A sharper Hungarian bound

- We can improve the bound further if we spend $O(n^3)$ operations.
- Recall that we estimated $\sigma_i(\mu) \leq \max_{1 \leq q \leq n} \sigma_{i,q}$ where $\sigma_{i,q} := \left(\sum_{1 \leq k \leq t} \min(n - i, n - \pi_k^{-1}(q)) \right)$.
- For a concrete solution μ of the Condorcet problem we could do better namely

$$v_\pi(\mu) \leq \sum_{1 \leq i \leq n} \sigma_{i,\mu(i)}.$$

- To find a better bound as before we have to solve the **assignment problem**, i.e. to find a permutation ρ where this sum is maximal.
- The **Hungarian algorithm** from combinatorial optimization solves this problem in $O(n^3)$ steps.

A sharper Hungarian bound

- We can improve the bound further if we spend $O(n^3)$ operations.
- Recall that we estimated $\sigma_i(\mu) \leq \max_{1 \leq q \leq n} \sigma_{i,q}$ where $\sigma_{i,q} := \left(\sum_{1 \leq k \leq t} \min \left(n - i, n - \pi_k^{-1}(q) \right) \right)$.
- For a concrete solution μ of the Condorcet problem we could do better namely

$$v_\pi(\mu) \leq \sum_{1 \leq i \leq n} \sigma_{i,\mu(i)}.$$

- To find a better bound as before we have to solve the **assignment problem**, i.e. to find a permutation ρ where this sum is maximal.
- The **Hungarian algorithm** from combinatorial optimization solves this problem in $O(n^3)$ steps.

Example of the Hungarian upper bound

In our example from S_9 we get $\rho = [4, 8, 6, 3, 2, 7, 5, 1, 9]$.

$n - i$	[1,	2,	3,	4,	5,	6,	7,	8,	9]	row max
8	[25,	18,	23,	36,	19,	32,	27,	26,	10]	36
7	[23,	18,	23,	34,	19,	32,	26,	25,	10]	34
6	[21,	18,	22,	31,	18,	31,	24,	23,	10]	31
5	[19,	18,	21,	27,	17,	27,	21,	20,	10]	27
4	[16,	18,	17,	22,	16,	22,	18,	17,	10]	22
3	[13,	14,	13,	17,	14,	17,	15,	13,	10]	17
2	[9,	10,	9,	12,	11,	12,	11,	9,	7]	12
1	[5,	5,	5,	6,	6,	6,	6,	5,	4]	6
0	[0,	0,	0,	0,	0,	0,	0,	0,	0]	0
red Hungarian sum:					162	Cusanus sum:				185

The Hungarian bound is 162, Cusanus bound is 185, the correct optimal value is 145.

An Hungarian heuristic to compute near to optimal Condorcet solutions

Algorithm 3: An hungarian heuristic for a near to optimal Condorcet solution

Input: An election $(\pi_k)_{1 \leq k \leq t} \in S_n^t$

Output: A permutation ρ with ('many') Condorcets votes $v_\pi(\rho)$

Compute the matrix

$$(\sigma_{i,q})_{1 \leq i \leq n, 1 \leq q \leq n} := \left(\sum_{1 \leq k \leq t} \min(n - i, n - \pi_k^{-1}(q)) \right)_{1 \leq i \leq n, 1 \leq q \leq n}$$

and use the Hungarian method to compute a permutation ρ such that

$$(v_\pi(\mu) \leq) \sum_{1 \leq i \leq n} \sigma_{i,\rho(i)}$$

is maximal.

Content

- 1 The Ranking Problem
- 2 The Condorcet Problem
- 3 Lehmer Code
- 4 Cusanus and Hungarian Upper Bounds
- 5 Implementation**
- 6 Outlook: Schulze's paths

Implementations in AXIOM / Fricas

- All algorithms were implemented in AXIOM, descendant Fricas, FriCAS-homepage:

`http://fricas.sourceforge.net`

- `Domain SymmetricGroup(n:`
`NonNegativeInteger)` is based on `Permutation`
`NonNegativeInteger`

- Code will be provided at `www.grabmeier.net`.

Implementations in AXIOM / Fricas

- All algorithms were implemented in AXIOM, descendant Fricas, FriCAS-homepage:

`http://fricas.sourceforge.net`

- `Domain SymmetricGroup(n:`
`NonNegativeInteger)` **is based on** `Permutation`
`NonNegativeInteger`

- Code will be provided at `www.grabmeier.net`.

Implementations in AXIOM / Fricas

- All algorithms were implemented in AXIOM, descendant Fricas, FriCAS-homepage:
`http://fricas.sourceforge.net`
- `Domain SymmetricGroup(n:
NonNegativeInteger)` **is based on** `Permutation
NonNegativeInteger`
- Code will be provided at `www.grabmeier.net`.

Content

- 1 The Ranking Problem
- 2 The Condorcet Problem
- 3 Lehmer Code
- 4 Cusanus and Hungarian Upper Bounds
- 5 Implementation
- 6 Outlook: Schulze's paths**

Schulze's beatpath method

- The method of **Schulze** is currently used in more and more organisations like the Debian developers for their elections of project leaders etc.
- Starting point are the pairing votes of Condorcet.
- Then it considers other paths from candidate i to candidate j than the direct one.
- This uses the idea of transitivity and instead of taking the direct comparison it takes the minimum votes along the paths as the new **Schulze vote** of a pairing: **Schulze's strongest paths**.
- This is used to find a **Schulze winner**, beating all the other candidates by Schulze votes.

Schulze's beatpath method

- The method of **Schulze** is currently used in more and more organisations like the Debian developers for their elections of project leaders etc.
- Starting point are the pairing votes of Condorcet.
- Then it considers other paths from candidate i to candidate j than the direct one.
- This uses the idea of transitivity and instead of taking the direct comparison it takes the minimum votes along the paths as the new **Schulze vote** of a pairing: **Schulze's strongest paths**.
- This is used to find a **Schulze winner**, beating all the other candidates by Schulze votes.

Schulze's beatpath method

- The method of **Schulze** is currently used in more and more organisations like the Debian developers for their elections of project leaders etc.
- Starting point are the pairing votes of Condorcet.
- Then it considers other paths from candidate i to candidate j than the direct one.
- This uses the idea of transitivity and instead of taking the direct comparison it takes the minimum votes along the paths as the new **Schulze vote** of a pairing: **Schulze's strongest paths**.
- This is used to find a **Schulze winner**, beating all the other candidates by Schulze votes.

Schulze's beatpath method

- The method of **Schulze** is currently used in more and more organisations like the Debian developers for their elections of project leaders etc.
- Starting point are the pairing votes of Condorcet.
- Then it considers other paths from candidate i to candidate j than the direct one.
- This uses the idea of transitivity and instead of taking the direct comparison it takes the minimum votes along the paths as the new **Schulze vote** of a pairing: **Schulze's strongest paths**.
- This is used to find a **Schulze winner**, beating all the other candidates by Schulze votes.

Schulze's beatpath method

- The method of **Schulze** is currently used in more and more organisations like the Debian developers for their elections of project leaders etc.
- Starting point are the pairing votes of Condorcet.
- Then it considers other paths from candidate i to candidate j than the direct one.
- This uses the idea of transitivity and instead of taking the direct comparison it takes the minimum votes along the paths as the new **Schulze vote** of a pairing: **Schulze's strongest paths**.
- This is used to find a **Schulze winner**, beating all the other candidates by Schulze votes.

Formal definition of Schulze

Definition

A **path** of strength z from a candidate p to a candidate q is called of **strength z** if there exists a sequence of $s \geq 2$ candidates $c := (p = c_1, \dots, c_s = q)$ with

$$v(c_\nu, c_{\nu+1}) := \sum_{1 \leq k \leq t} [\pi_k^{-1}(c_\nu) < \pi_k^{-1}(c_{\nu+1})]$$

and

$$z := \min_{1 \leq \nu < s} v(c_\nu, c_{\nu+1})$$

for all $1 \leq \nu < s$.

We define an $n \times n$ -matrix s by and $s(p, q) := 0$ if $p = q$ or if there is no path from p to q and $s(p, q) := z$, if there is a path with strength z and z is maximal.

Floyd and Warshall compute the matrix of strongest paths I

- Paths of maximal strengths can be found by using a modification of the $O(n^3)$ -algorithm of **Floyd and Warshall** for the **'All Pairs Shortest Path Problem'**.
- Note, that a path of maximal strength from candidate p to candidate q via candidate k , contains the paths of maximal strengths from p to k and from k to q .
- Hence, we recursively can assume that all paths of maximal strengths between candidates were found among the candidates via a candidate from $1 \dots k - 1$.
- Then to find all paths from p to q with $p \neq q$ of maximal strengths via candidate up to k , then there are two cases:

Floyd and Warshall compute the matrix of strongest paths I

- Paths of maximal strengths can be found by using a modification of the $O(n^3)$ -algorithm of **Floyd and Warshall** for the **'All Pairs Shortest Path Problem'**.
- Note, that a path of maximal strength from candidate p to candidate q via candidate k , contains the paths of maximal strengths from p to k and from k to q .
- Hence, we recursively can assume that all paths of maximal strengths between candidates were found among the candidates via a candidate from $1 \dots k - 1$.
- Then to find all paths from p to q with $p \neq q$ of maximal strengths via candidate up to k , then there are two cases:

Floyd and Warshall compute the matrix of strongest paths I

- Paths of maximal strengths can be found by using a modification of the $O(n^3)$ -algorithm of **Floyd and Warshall** for the **'All Pairs Shortest Path Problem'**.
- Note, that a path of maximal strength from candidate p to candidate q via candidate k , contains the paths of maximal strengths from p to k and from k to q .
- Hence, we recursively can assume that all paths of maximal strengths between candidates were found among the candidates via a candidate from $1 \dots k - 1$.
- Then to find all paths from p to q with $p \neq q$ of maximal strengths via candidate up to k , then there are two cases:

Floyd and Warshall compute the matrix of strongest paths I

- Paths of maximal strengths can be found by using a modification of the $O(n^3)$ -algorithm of **Floyd and Warshall** for the **'All Pairs Shortest Path Problem'**.
- Note, that a path of maximal strength from candidate p to candidate q via candidate k , contains the paths of maximal strengths from p to k and from k to q .
- Hence, we recursively can assume that all paths of maximal strengths between candidates were found among the candidates via a candidate from $1 \dots k - 1$.
- Then to find all paths from p to q with $p \neq q$ of maximal strengths via candidate up to k , then there are two cases:

Floyd and Warshall compute the matrix of strongest paths II

- First case is, that there are paths via a candidate from 1 to $k - 1$. Then their paths with maximal strengths are known and this strength is already known by the current value of $s(p, q)$.
- If the path is via k , we glue together the paths from $p \rightarrow k$ and $k \rightarrow q$ with new strength $\min(s(p, k), s(k, q))$ and compare it with $s(p, q)$. Choose the one of these two with larger strength.

Floyd and Warshall compute the matrix of strongest paths II

- First case is, that there are paths via a candidate from 1 to $k - 1$. Then their paths with maximal strengths are known and this strength is already known by the current value of $s(p, q)$.
- If the path is via k , we glue together the paths from $p \rightarrow k$ and $k \rightarrow q$ with new strength $\min(s(p, k), s(k, q))$ and compare it with $s(p, q)$. Choose the one of these two with larger strength.

Schulze winner

- A **Schulze winner**, i.e. the rank 1 candidate $\sigma(1)$ is the one, if existent and unique, for whom we have

$$s(\sigma(1), q) \geq s(q, \sigma(1))$$

for all $1 \leq q \leq n$. If not existent, we need some extra decision procedure to choose one first ranked candidate.

- We have decided to take the candidate with maximum number of wins w.r.to s .
- If there is more than one, we break the tie by adding up all supporting votes, i.e. the row sums of s . We choose the one with maximal votes, if two or more have the same number of votes, we take the first one in the given order of the candidates.
- Recursively, we apply this method after cancelling the first ranked candidate's row and column of the matrix s .

Schulze winner

- A **Schulze winner**, i.e. the rank 1 candidate $\sigma(1)$ is the one, if existent and unique, for whom we have

$$s(\sigma(1), q) \geq s(q, \sigma(1))$$

for all $1 \leq q \leq n$. If not existent, we need some extra decision procedure to choose one first ranked candidate.

- We have decided to take the candidate with maximum number of wins w.r.to s .
- If there is more than one, we break the tie by adding up all supporting votes, i.e. the row sums of s . We choose the one with maximal votes, if two or more have the same number of votes, we take the first one in the given order of the candidates.
- Recursively, we apply this method after cancelling the first ranked candidate's row and column of the matrix s .

Schulze winner

- A **Schulze winner**, i.e. the rank 1 candidate $\sigma(1)$ is the one, if existent and unique, for whom we have

$$s(\sigma(1), q) \geq s(q, \sigma(1))$$

for all $1 \leq q \leq n$. If not existent, we need some extra decision procedure to choose one first ranked candidate.

- We have decided to take the candidate with maximum number of wins w.r.to s .
- If there is more than one, we break the tie by adding up all supporting votes, i.e. the row sums of s . We choose the one with maximal votes, if two or more have the same number of votes, we take the first one in the given order of the candidates.
- Recursively, we apply this method after cancelling the first ranked candidate's row and column of the matrix s .

Schulze winner

- A **Schulze winner**, i.e. the rank 1 candidate $\sigma(1)$ is the one, if existent and unique, for whom we have

$$s(\sigma(1), q) \geq s(q, \sigma(1))$$

for all $1 \leq q \leq n$. If not existent, we need some extra decision procedure to choose one first ranked candidate.

- We have decided to take the candidate with maximum number of wins w.r.to s .
- If there is more than one, we break the tie by adding up all supporting votes, i.e. the row sums of s . We choose the one with maximal votes, if two or more have the same number of votes, we take the first one in the given order of the candidates.
- Recursively, we apply this method after cancelling the first ranked candidate's row and column of the matrix s .

Schulze votes optimization

In our setting we would be more interested in

$$w : S_n \rightarrow \mathbb{N}, \pi \mapsto w(\pi) = \sum_{1 \leq i < j \leq n} s(i, j)$$

the number of **Schulze beatpath votes** and in finding optimal Schulze rankings.

Outlook

- **Compare** Condorcet ranking with Schulze ranking.
- In **social choice** theory, a subject with contributions from many fields, one is interested in finding a **single winner** of an election and not so much in a **full ranking**. Many quality criteria are raised and the attempt is to design methods, which satisfies them in order to avoid strategic voting in the awareness of a shortcome.
- **Relate** quality criteria from social choice theory to our setting.

Outlook

- **Compare** Condorcet ranking with Schulze ranking.
- In **social choice** theory, a subject with contributions from many fields, one is interested in finding a **single winner** of an election and not so much in a **full ranking**. Many quality criteria are raised and the attempt is to design methods, which satisfies them in order to avoid strategic voting in the awareness of a shortcome.
- **Relate** quality criteria from social choice theory to our setting.

Outlook

- **Compare** Condorcet ranking with Schulze ranking.
- In **social choice** theory, a subject with contributions from many fields, one is interested in finding a **single winner** of an election and not so much in a **full ranking**. Many quality criteria are raised and the attempt is to design methods, which satisfies them in order to avoid strategic voting in the awareness of a shortcome.
- **Relate** quality criteria from social choice theory to our setting.