Chapter 10:
Voting Strengths

Panos Louridas

Athens University of Economics and Business
Real World Algorithms
A Beginners Guide
The MIT Press

Outline

Candidates

The Human Resources department of a company invites the employees to vote for their preferred catering service:

- "MeatLovers", or M, offers a selection of mostly meat menus.
- "BitOfEverything", or E, offers a more varied menu.
- "VegForLife", or V, offers strictly vegetarian dishes.

Election Results

- M: 40%.
- E: 30%.
- V: 30%

That means that everybody will be eating meat, whether they like it or not.

Ranked Preferences

If, however, we take into account the ranked preferences of the employees, from most preferred to least preferred, we have

$$40\%: [M, E, V]$$
$$30\%: [V, E, M]$$
$$30\%: [E, V, M]$$

Pairwise Preferences: M and E

$$40\%: [M, E, V]$$
$$30\%: [V, E, M]$$
$$30\%: [E, V, M]$$

- 40% of voters prefer M to E.
- 30% of voters prefer E to M.
- 30% of voters prefer E to M.
- Therefore, between the two, 60% of voters prefer E and 40% prefer M.

Pairwise Preferences: M and V

$$40\%: [M, E, V]$$
$$30\%: [V, E, M]$$
$$30\%: [E, V, M]$$

- 40% of voters prefer M to V.
- 30% of voters prefer V to M.
- 30% of voters prefer V to M.
- Therefore, between the two, 60% of voters prefer V and 40% prefer M.

Pairwise Preferences: E and V

$$40\%: [M, E, V]$$
$$30\%: [V, E, M]$$
$$30\%: [E, V, M]$$

- 40% of voters prefer E to V.
- 30% of voters prefer V to E.
- 30% of voters prefer E to V.
- Therefore, between the two, 70% of voters prefer E and 30% prefer V.

Pairwise Preferences: Results

- E vs. M: 60% prefer E, 40% prefer M.
- V vs. M: 60% prefer V, 40% prefer M.
- E vs. V: 70% prefer E, 30% prefer M.

E wins two pairwise comparisons, V one pairwise comparison, whereas M does not win any pairwise comparison.
So E wins the election.

Outline

Definition

Definition

In *plurality voting* voters may record only one choice on the ballot. The candidate with the most ballots wins the election.

The Problem with Plurality Voting

- The problem with plurality voting is that voters cannot express all their preferences.
- So, a candidate that is liked by many voters but is not the top choice of a relative majority will lose to a candidate that is the top choice of a relative majority.

U.S. Presidential Elections 2000 (1)

- The race was between George W. Bush, Al Gore, and Ralph Nader.
- The president is elected by the Electoral College.
- The Electoral College is based on the election results of the particular U.S. states.
- After much drama, the 2000 election was decided by the results in the state of Florida.

U.S. Presidential Elections 2000 (2)

- George W. Bush got 2,912,790 votes, i.e., 48,847% of the total.
- Al Gore got 2,912,253 votes, i.e., 48,838% of the total.
- Ralph Nader got 97,421 votes, i.e., 1,634% of the total.

U.S. Presidential Elections 2000 (3)

- George W. Bush won by 537 votes, or 0,009% of the total in Florida.
- It is generally thought that most of Ralph Nader voters would prefer Al Gore to George W. Bush.
- If that is true, and the U.S. Presidential Elections were held with a voting system that allowed voters to express their second choices as well, the winner would be Al Gore.

French Presidential Elections 2002 (1)

- The contestants were Jacques Chirac, Lionel Jospin, and fourteen other politicians.
- The candidate that gets more than 50% of the votes is elected president.
- If this does not happen, then the first two candidates are selected to participate in a second election round.
- It came as a shock to the world that instead of Lionel Jospin, Jacques Chirac would face the far-right Jean-Marie Le Pen in the second round.

French Presidential Elections 2002 (2)

That does not mean that most French voters supported the far-right. The results of the first round were:

- Jacques Chirac got 5,666,440 votes, i.e., 19,88% of the total.
- Jean-Marie Le Pen got 4,805,307 votes, i.e., 16,86% of the total.
- Lionel Jospin got 4,610,749 votes, i.e., 16,18% of the total.

French Presidential Elections 2002 (3)

- In the second round, two weeks later, Jacques Chirac got more than 82% of the votes.
- Jean-Marie Le Pen got less than 18%.
- Chirac got almost all the votes of the supporters of those eliminated in the first round, while Le Pen barely moved from his performance on the first round.

French Presidential Elections 2002 (4)

- The problem resulted from the large number of candidates: 16.
- The votes were shared among the supporters of more moderate candidates, so that the most extreme candidate passed to the second round.

Outline

The Condorcet Criterion

Definition

The *Condorcet criterion* states that in an election the winner should be the candidate that when compared with any other candidate is preferred by most voters. The winner is called the *Condorcet candidate* or *Condorcet winner*.

The name comes from Marie Jean Antoine Nicolas Caritat, Marquis de Condorcet, a French 18th century philosopher and mathematician who described the situation in 1785 in his book *Essai sur l'application de l'analyse à la probabilité des décisions rendues à la pluralité des voix* (*Essay on the Application of Analysis to the Probability of Majority Decisions*).

Condorcet System

Definition

A voting system that declares as a winner of the elections the candidate that meets the Condorcet criterion, if possible, is called a *Condorcet system*.

Approval Voting

- In approval voting, the voters select as many candidates as they want.
- The winner is the candidate that gets most votes.

Approval Voting and the Condorcet Criterion

- Suppose we have three candidates A, B, C, and the following ballots:

$$60\%: A, B$$
$$40\%: C, B$$

- As 100% of voters voted for B, B will be the winner.
- Suppose that 60% of voters prefer A to B and B to C: $[A, B, C]$.
- Also suppose that 40% of the voters prefer C to B and B to A: $[C, B, A]$.
- Then, in pairwise comparisons, A prevails over B with 60% vs. 40%.
- Although most voters prefer A to B, A is not elected.

Therefore approval voting does not meet the Condorcet criterion.

Borda Count

It is named after another 18th century French, the mathematician and political scientist Jean-Charles de Borda, in1770.

- The voters give points to the candidates.
- If we have n candidates, the first choice in a ballot gets $n-1$ points, the second choice gets $n-2$ points, etc., until the last choice that gets zero points.

Borda Count and the Condorcet Criterion

- Suppose we have three candidates, A, B, C, and the following ballots:

$$60\%: [A, B, C]$$
$$40\%: [B, C, A]$$

- If we have $100m$ voters, candidate A gets $(60 \times 2)m = 120m$ points, candidate B gets $(60 + 2 \times 40)m = 140m$ points and candidate C gets $40m$ points.
- The winner is candidate B, even though most voters prefer A to B.

Therefore the Borda count does not meet the Condorcet criterion.

Existence of Condorcet Winner

A Condorcet winner might not necessarily exist.

- For example, let's say we have three candidates A, B and C, and the following ballots:

$$30 : [A, B, C]$$
$$30 : [B, C, A]$$
$$30 : [C, A, B]$$

- If we do the pairwise comparisons, then we find that A beats B by 60 to 30, B beats C by 60 to 30, and C beats A by 60 to 30.
- As no candidate beats more candidates than others, no overall winner emerges.

Outline

The Problem (1)

- The method we used to find the winner in the catering example is prone to ties.
- For example, suppose that the ballots were as follows:

$$10 \times [A, B, C]$$
$$5 \times [B, C, A]$$
$$5 \times [C, A, B]$$

- A beats B by 15 to 5.
- A and C are tied with 10 to 10.
- B beats C by 15 to 5.
- So, as both A and B have a pairwise win, we cannot declare a winner.

The Schulze Method

- A method that finds the Condorcet winner, if such a winner exists, is the Schulze method.
- It was proposed by Markus Schulze in 1997.
- The basic idea in the Schulze method is that we use the pairwise preferences of the voters to construct a graph.
- Then the preferences between candidates are found by tracing paths on that graph.

Step 1: Calculation of Pairwise Preferences

- The first step in the Schulze method is the calculation of the pairwise preferences.
- Suppose we have n ballots and m candidates.
- The ballots are $B = B_1, B_2, \ldots, B_n$ and the candidates are $C = c_1, c_2, \ldots, c_m$.
- We create an array P of size $m \times m$.
- We take each ballot B_i in turn. For each pair of candidates c_j and c_k where c_j is preferred over c_k in the ballot, we add one to the $P[c_j, c_k]$ element of array P.
- At the end, item $P[c_j, c_k]$ will show how many candidates prefer candidate c_j over candidate c_k.

Ballot Example (1)

- Suppose we have the ballot $b = [c_1, c_3, c_4, c_2, c_5]$.
- The voter prefers c_1 over all other candidates, c_3 over c_4, c_2, c_5, c_4 over c_2, c_5, and c_2 over c_5.
- As c_1 is preferred over c_3, c_4, c_2, and c_5, we add one to elements $P[c_1, c_3]$, $P[c_1, c_4]$, $P[c_1, c_2]$, and $P[c_1, c_5]$.
- As c_3 is preferred over c_4, c_2, and c_5, we add one to elements $P[c_3, c_4]$, $P[c_3, c_2]$, and $P[c_3, c_5]$, etc., until c_2 where we add one to $P[c_2, c_5]$.

Ballot Example (2)

$$b = [c_1, c_3, c_4, c_2, c_5]$$

$$\begin{array}{c} \\ c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \end{array} \begin{array}{c} \begin{array}{ccccc} c_1 & c_2 & c_3 & c_4 & c_5 \end{array} \\ \begin{bmatrix} - & +1 & +1 & +1 & +1 \\ - & - & - & - & +1 \\ - & +1 & - & +1 & +1 \\ - & +1 & - & - & +1 \\ - & - & - & - & - \end{bmatrix} \end{array}$$

Pairwise Preferences Calculation Algorithm

Algorithm: Calculate pairwise preferences.

`CalcPairwisePreferences`$(ballots, m) \rightarrow P$

Input: $ballots$, an array of ballots, where each ballot is an array of candidates
m, the number of candidates

Output: P, an array of size $m \times m$ with the pairwise preferences for candidates; $P[i, j]$ is the number of voters that prefer candidate i to candidate j

```
1  P ← CreateArray(m · m)
2  for i ← 0 to m do
3      for j ← 0 to m do
4          P[i, j] ← 0
5  for i ← 0 to |ballots| do
6      ballot ← ballots[i]
7      for j ← 0 to |ballot| do
8          c_j ← ballot[j]
9          for k ← j + 1 to |ballot| do
10             c_k ← ballot[k]
11             P[c_j, c_k] ← P[c_j, c_k] + 1
12 return P
```

Pairwise Preferences Calculation (1)

Let's say that in an election we had four candidates, A, B, C, and D. There were 21 candidates with the following ballots:

$$6 \times [A, C, D, B]$$
$$4 \times [B, A, D, C]$$
$$3 \times [C, D, B, A]$$
$$4 \times [D, B, A, C]$$
$$4 \times [D, C, B, A]$$

Pairwise Preferences Calculation (2)

The pairwise preferences array is calculated as follows:

$$\begin{array}{c} \\ A \\ B \\ C \\ D \end{array} \begin{array}{c} \begin{array}{cccc} A & B & C & D \end{array} \\ \begin{bmatrix} 0 & 6 & (6+4+4) & (6+4) \\ (4+3+4+4) & 0 & (4+4) & 4 \\ (3+4) & (6+3+4) & 0 & (6+3) \\ (3+4+4) & (6+3+4+5) & (4+4+4) & 0 \end{bmatrix} \end{array}$$

$$= \begin{array}{c} \\ A \\ B \\ C \\ D \end{array} \begin{array}{c} \begin{array}{cccc} A & B & C & D \end{array} \\ \begin{bmatrix} 0 & 6 & 14 & 10 \\ 15 & 0 & 8 & 4 \\ 7 & 13 & 0 & 9 \\ 11 & 17 & 12 & 0 \end{bmatrix} \end{array}$$

Step 2: Construction of Preferences (Election) Graph

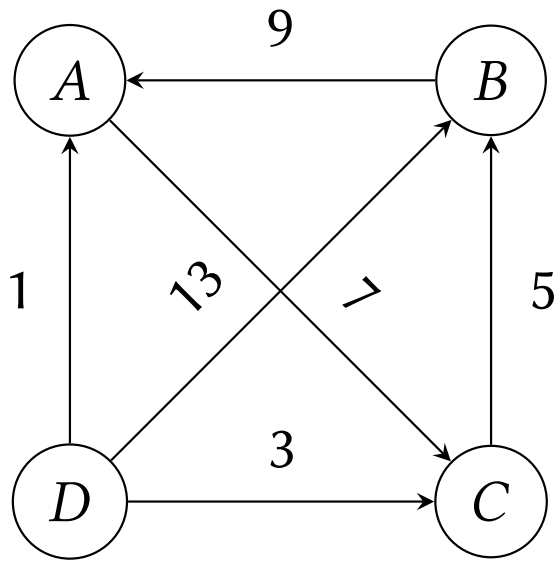
- In the second step we construct a graph where the candidates are the nodes and the positive differences between the preferences are the weight of the edges.
- If for two candidates c_i and c_j, the number $P[c_i, c_j]$ of voters that prefer c_i over c_j is greater than the number $P[c_j, c_i]$ of voters that prefer c_j over c_i, we add in the graph the link $c_i \longrightarrow c_j$ with weight $P[c_i, c_j] - P[c_j, c_i]$.
- Otherwise we use $-\infty$ to show that there is no link.

Graph Construction Example (1)

$$
\begin{array}{c} \\ A \\ B \\ C \\ D \end{array}
\begin{array}{c}
\begin{array}{cccc} A & B & C & D \end{array} \\
\begin{bmatrix} 0 & (6 < 15) & (14 - 7) & (10 < 11) \\ (15 - 6) = 9 & 0 & (8 < 13) & (4 < 17) \\ (7 < 14) & (13 - 8) & 0 & (9 < 12) \\ (11 - 10) = 1 & (17 - 4) & (12 - 9) & 0 \end{bmatrix}
\end{array}
$$

$$
=
\begin{array}{c} \\ A \\ B \\ C \\ D \end{array}
\begin{array}{c}
\begin{array}{cccc} A & B & C & D \end{array} \\
\begin{bmatrix} -\infty & -\infty & 7 & -\infty \\ 9 & -\infty & -\infty & -\infty \\ -\infty & 5 & -\infty & -\infty \\ 1 & 13 & 3 & -\infty \end{bmatrix}
\end{array}
$$

Graph Construction Example (2)



Step 3: Finding the Strongest Paths

Definition

The *strength*, or *width*, of a path is the minimum weight in the links that make up the path.

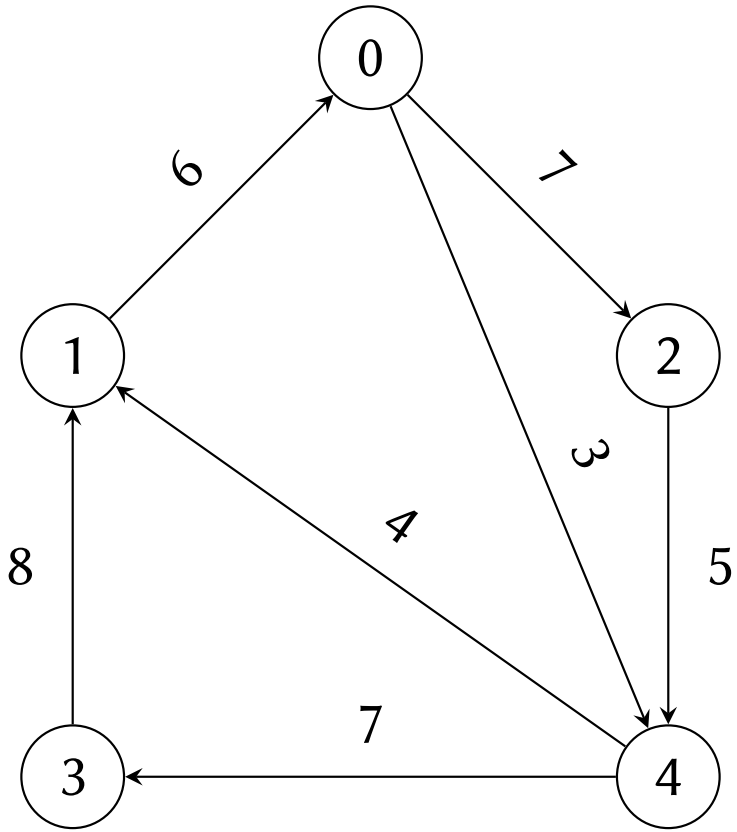
The definition corresponds to the weight that can be carried by a bridge, or the capacity of a link.

Strongest Path

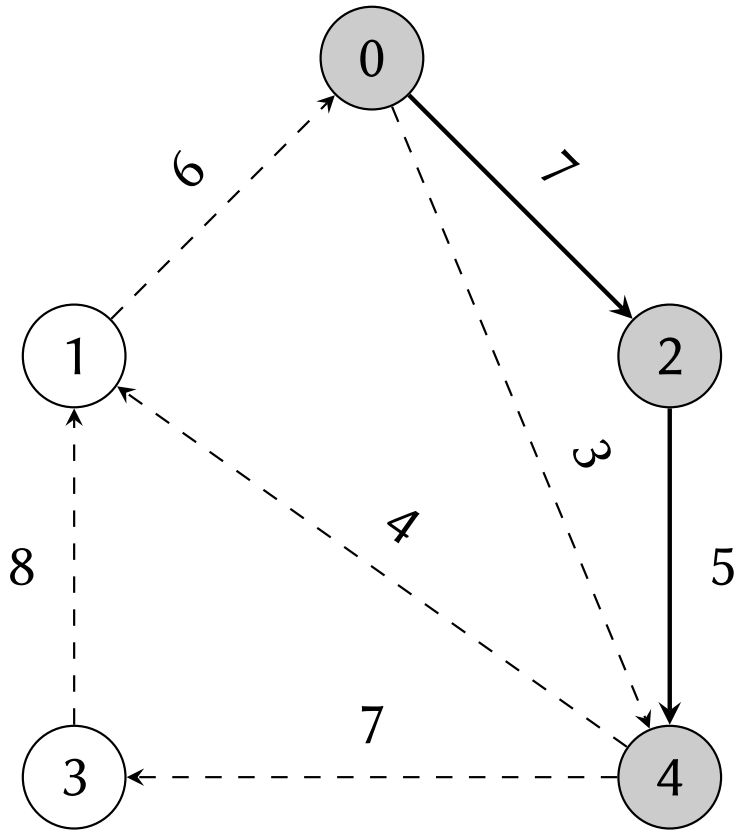
Definition

The *strongest path*, or *widest path* between two nodes is the path with the maximum strength, or width, among all paths between the two nodes.

Strongest Path Example (1)

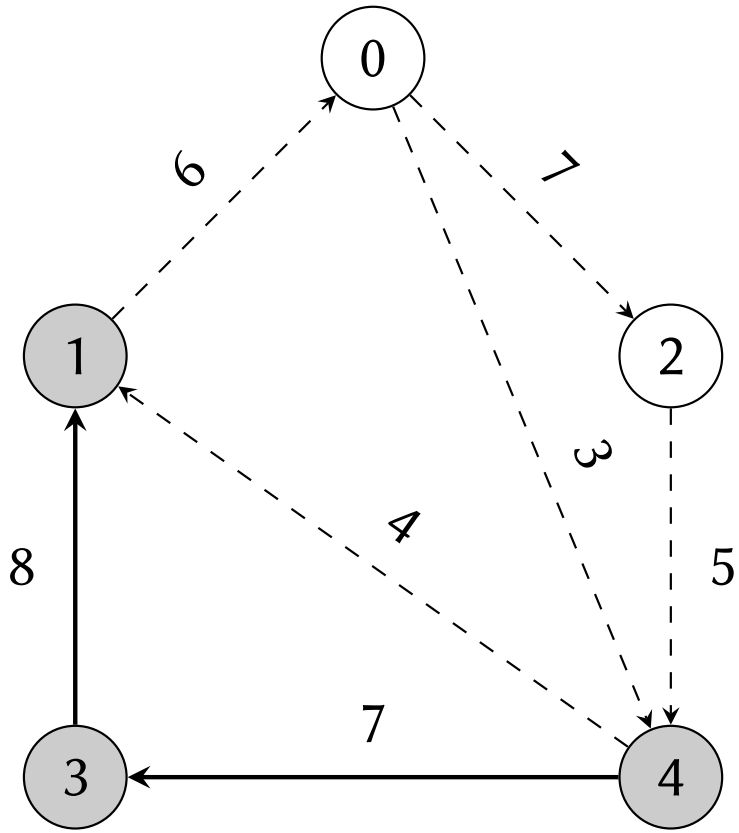


Strongest Path Example (2)



The strongest path between nodes 0 and 4 is $0 \longrightarrow 2 \longrightarrow 4$ and has strength 5.

Strongest Path Example (3)



The strongest path between nodes 4 and 1 is $4 \longrightarrow 3 \longrightarrow 1$ and has strength 7.

Strongest Paths Algorithm (1)

- We put all nodes in an order $c_1, c_2, \ldots, c_n$.
- We find the strongest path between every pair c_i and c_j using no intermediate nodes from $c_1, c_2, \ldots, c_n$.
- The strongest path between two ndoes without using intermediate nodes is the direct link between the two nodes, if it exists.

Strongest Paths Algorithm (2)

- We look for the strongest path between every pair c_i and c_j using the first node, c_1, as an intermediate node.
- If in the previous step we had found a path between c_i and c_j, then if there exist paths $c_i \longrightarrow c_1$ and $c_1 \longrightarrow c_j$ we compare the strength of the path $c_i \longrightarrow c_j$ with the strength of the path $c_i \longrightarrow c_1 \longrightarrow c_j$, and we take the strongest of the two as the new strongest path between c_i and c_j.

Strongest Paths Algorithm (3)

- Suppose we have found the strongest path between every pair c_i and c_j using the first k nodes as intermediate nodes.
- We look for the strongest path between c_i and c_j using the first $k+1$ nodes as intermediate nodes.
- If we find such a path, it will consist of two parts. The first will be from c_i to c_{k+1} using the first k nodes as intermediate nodes and the second will be from c_{k+1} to c_j again using the first k nodes as intermediate nodes.
- We have already found the strength of these paths, so, if $s_{i,j}(k)$ is the strength of the path that may use the first k nodes, we have:

$$s_{i,j}(k+1) = max\Big(s_{i,j}(k), min\big(s_{i,k+1}(k), s_{k+1,j}(k)\big)\Big)$$

Strongest Paths Algorithm (4)

Algorithm: Calculate strongest paths.

`CalcStrongestPaths`$(W, n) \rightarrow (S, pred)$

Input: W, an array of size $n \times n$ representing the adjacency matrix of a graph; $W[i, j]$ is the weight of the edge between nodes i and j
n, the size of each dimension of W

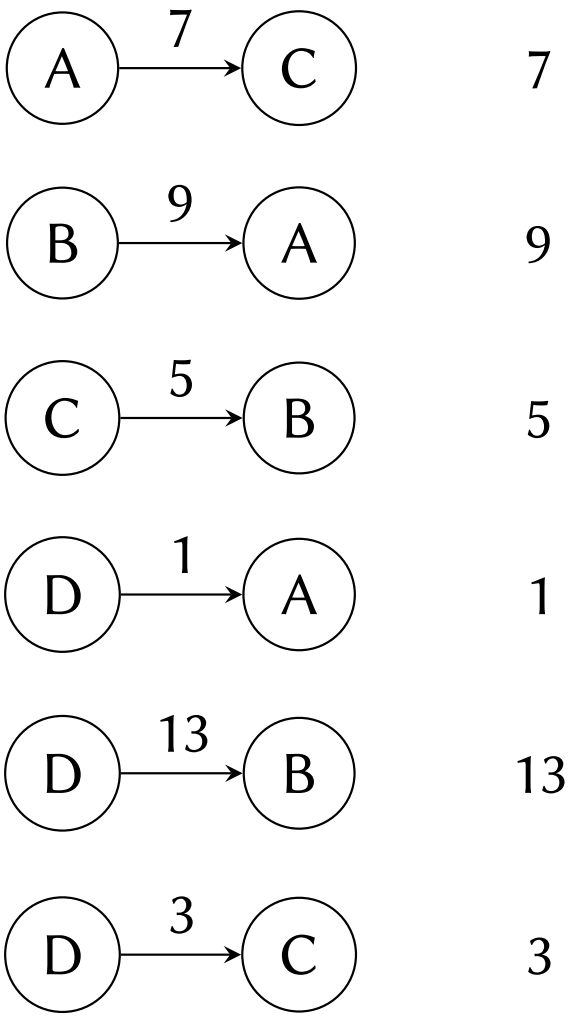
Output: S, an array of size $n \times n$ such that $S[i, j]$ is the strongest path between nodes i and j
$pred$, an array of size $n \times n$ such that $pred[i, j]$ is the predecessor of node i in the strongest path to node j

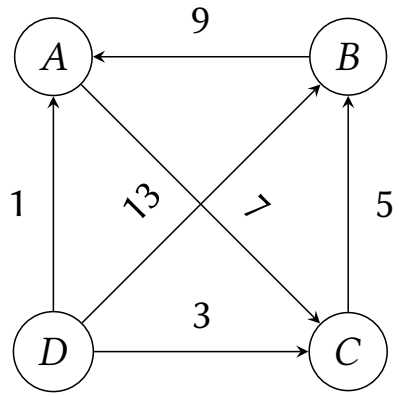
```
1  S ← CreateArray(n · n)
2  pred ← CreateArray(n · n)
3  for i ← 0 to n do
4      for j ← 0 to n do
5          if W[i, j] > W[j, i] then
6              S[i, j] ← W[i, j] − W[j, i]
7              pred[i, j] ← i
8          else
9              S[i, j] ← −∞
10             pred[i, j] ← −1
11 for k ← 0 to n do
12     for i ← 0 to n do
13         if i ≠ k then
14             for j ← 0 to n do
15                 if j ≠ i then
16                     if S[i, j] < Min(S[i, k], S[k, j]) then
17                         S[i, j] ← Min(S[i, k], S[k, j])
18                         pred[i, j] ← pred[k, j]
19 return (S, pred)
```

Strongest Paths Algorithm (5)

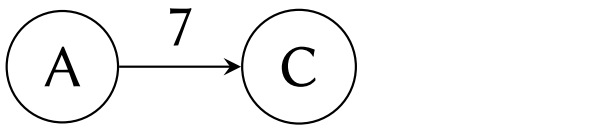
- The loop in lines 2–10 is executed n^2 times.
- The loop in lines 11–18 is executed n^3 times.
- As we use an adjacency matrix, accessing its elements requires constant time, so the time required by the algorithm is $\Theta(n^3)$.

Strongest Paths Example (1)

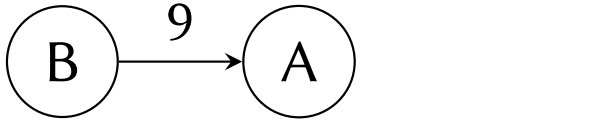




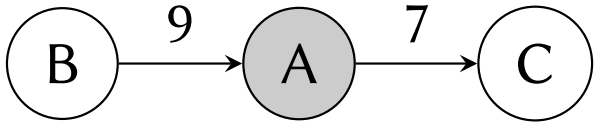
Strongest Paths Example (2)



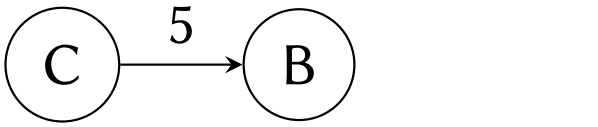
7



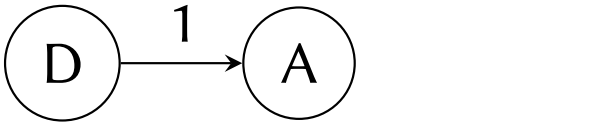
9



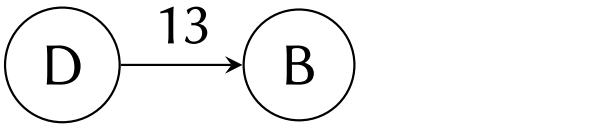
7



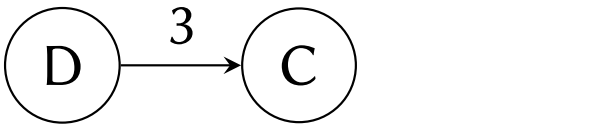
5



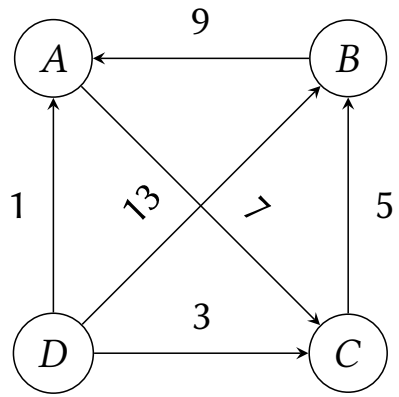
1



13



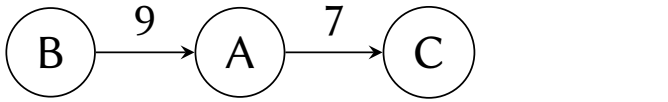
3



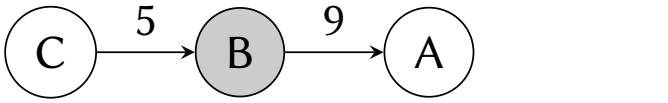
Strongest Paths Example (3)

7

9

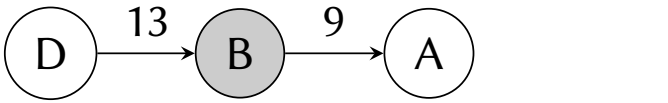


7



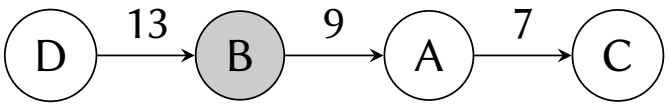
5

5

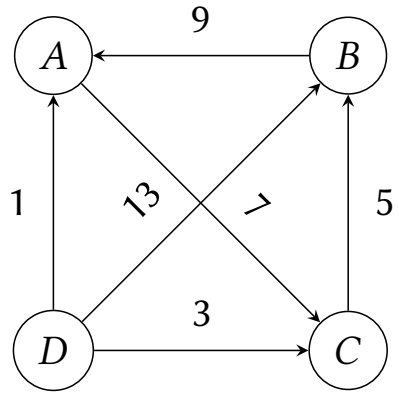


9

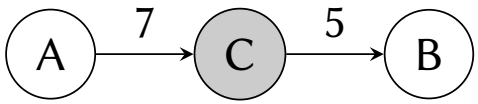
13



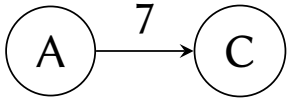
7



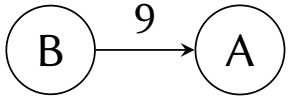
Strongest Paths Example (4)



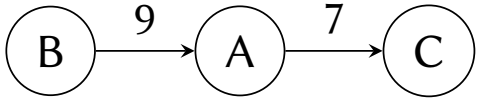
5



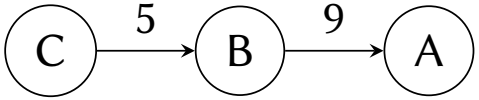
7



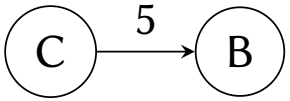
9



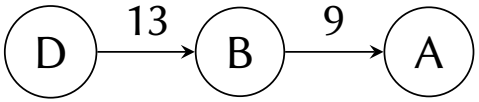
7



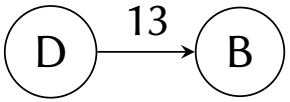
5



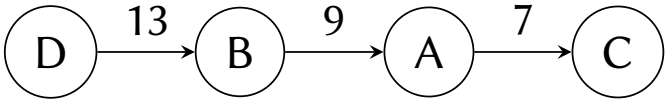
5



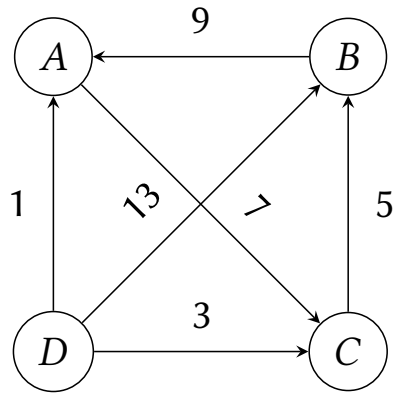
9



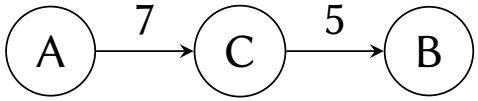
13



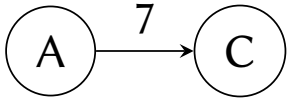
7



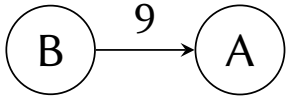
Strongest Paths Example (5)



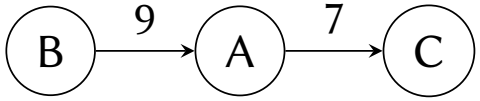
5



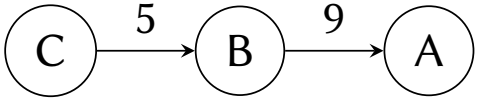
7



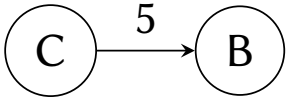
9



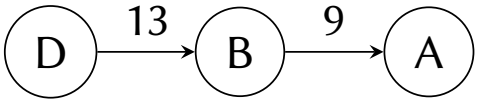
7



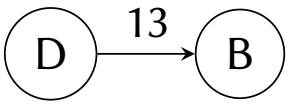
5



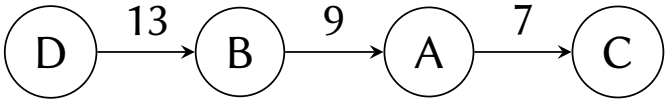
5



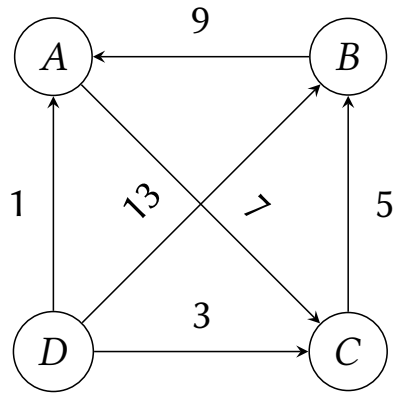
9



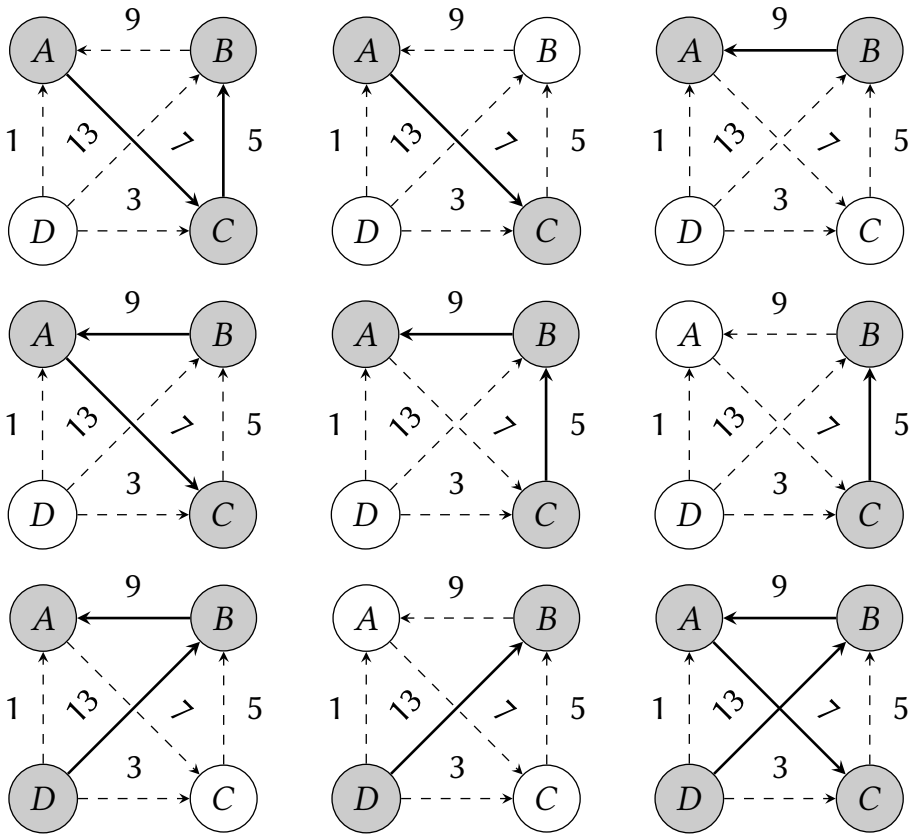
13



7



The Strongest Paths in the Graph



Step 3: Candidates Comparison

- Having found the strongest paths, we want to find, for each pair of candidates c_i and c_j, the amount of support for c_i against c_j and the amount of support for c_j against c_i.
- The support of a candidate against another candidate is the strength of the path between the two candidates.
- If the strength of the path from c_i to c_j is greater than the strength of the path from c_j to c_i, we say that c_i wins over c_j.
- Then we want to find out, for each candidate c_i, over how many candidates the c_i candidate wins.
- To do that we go over the path strengths we have found and we add the number of times c_i is preferred over c_j.

Shortest Path Array

$$\begin{array}{c} \\ A \\ B \\ C \\ D \end{array} \begin{array}{c} \begin{array}{cccc} A & B & C & D \end{array} \\ \begin{bmatrix} -\infty & 5 & 7 & -\infty \\ 9 & -\infty & 7 & -\infty \\ 5 & 5 & -\infty & -\infty \\ 9 & 13 & 7 & -\infty \end{bmatrix} \end{array}$$

Results Calculation

Algorithm: Calculate results.

`CalcResults`$(S, n) \rightarrow wins$

Input: S, an array of size $n \times n$ with the strengths of the strongest paths between nodes; $s[i, j]$ is the strength of the strongest path between nodes i and j
n, the size of each dimension of S

Output: $wins$, an array of size n; item i of $wins$ is a list containing m integer items $j_1, j_2, \ldots, j_m$ for which $S[i, j_k] > S[j_k, i]$

```
1  wins ← CreateArray(n)
2  for i ← 0 to n do
3      list ← CreateList()
4      wins[i] ← list
5      for j ← 0 to n do
6          if i ≠ j then
7              if S[i, j] > S[j, i] then
8                  InsertInList(list, NULL, j)
9  return wins
```

Example Election Results (1)

From:

$$\begin{array}{c} \\ A \\ B \\ C \\ D \end{array} \begin{array}{c} \begin{array}{cccc} A & B & C & D \end{array} \\ \begin{bmatrix} -\infty & 5 & 7 & -\infty \\ 9 & -\infty & 7 & -\infty \\ 5 & 5 & -\infty & -\infty \\ 9 & 13 & 7 & -\infty \end{bmatrix} \end{array}$$

We find:

$$wins = [[2], [2, 0], [\quad], [2, 1, 0]]$$

That is, the wins of the candidates over other candidates are:

$$[[C], [C, A], [\quad], [C, B, A]]$$

Example Election Results (2)

- So, A wins over C, B wins over A and C, C does not win anybody, and D wins over A, B, C.
- Therefore the election results are: $A = 1$, $B = 2$, $C = 0$, and $D = 3$.
- D is the candidate that is preferred by most voters.

Return to the Tied Example (1)

We saw earlier the following example, where by simple pairwise preferences we could not find a winner:

$$\begin{aligned} 10 \times [A, B, C] \\ 5 \times [B, C, A] \\ 5 \times [C, A, B] \end{aligned}$$

- A beats B by 15 to 5.
- A and C are tied with 10 to 10.
- B beats C by 15 to 5.
- So, as both A and B have a pairwise win, we cannot declare a winner.

Return to the Tied Example (2)

The preference array is:

$$\begin{array}{c} \\ A \\ B \\ C \end{array} \begin{array}{c} \begin{array}{ccc} A & B & C \end{array} \\ \begin{bmatrix} 0 & 15 & 10 \\ 5 & 0 & 15 \\ 10 & 5 & 0 \end{bmatrix} \end{array}$$

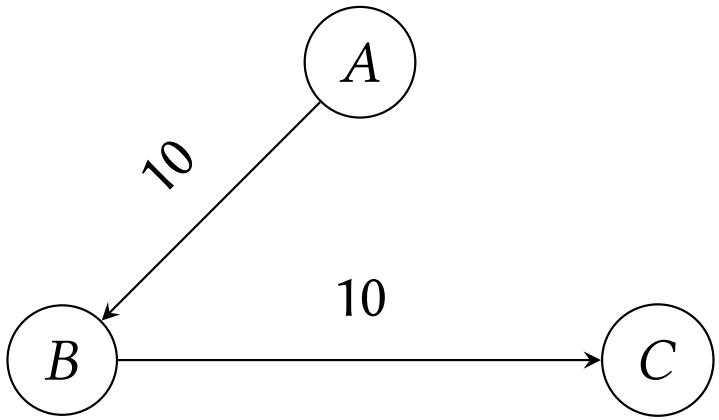
Return to the Tied Example (3)

The adjacency matrix for the election graph is:

$$\begin{array}{c} \\ A \\ B \\ C \end{array} \begin{array}{c} \begin{array}{ccc} A & B & C \end{array} \\ \begin{bmatrix} -\infty & 10 & -\infty \\ -\infty & -\infty & 10 \\ -\infty & -\infty & -\infty \end{bmatrix} \end{array}$$

Return to the Tied Example (4)

The election graph is:



Return to the Tied Example (5)

The Schulze method will give the following result:

- A wins over B and C.
- B wins over C.
- C does not win anybody.

Therefore the winner is A; we no longer have a tie.

Some Remarks for the Schulze Method

- The Schulze method does not declare only the first candidate, but also ranks all candidates, therefore it can be used to select the first k out of n candidates.
- If no Condorcet winner exists, of course it will not report any.
- In general, we may find candidates with the same ranking, but it is rare not to have a winner.

Outline

Relationship with the Floyd-Warshall Algorithm

- The algorithm for finding the strongest paths is a variation of an algorithm for finding the shortest paths.
- That algorithm is the Floyd-Warshall algorithm.
- It was published by Robert Floyd in 1962, but similar algorithms had also been published by Bernard Roy in 1959 and Stephen Warshall in 1962.
- The Floyd-Warshall algorithm requires time $\Theta(n^3)$. It is slower than Dijkstra's algorithm, but still runs fast. Also, it is easy to implement and works with negative weights.

The Floyd-Warshall Algorithm

Algorithm: Floyd-Warshall all pairs shortest paths.

`FloydWarshall`$(W, n) \rightarrow (dist, pred)$

Input: W, an array of size $n \times n$ representing the adjacency matrix of a graph; $W[i, j]$ is the weight of the edge between nodes i and j
n, the size of each dimension of W

Output: $dist$, an array of size $n \times n$ such that $dist[i, j]$ is the shortest path between nodes i and j
$pred$, an array of size $n \times n$ such that $pred[i, j]$ is the predecessor of node i in the shortest path to node j

```
1  dist ← CreateArray(n · n)
2  pred ← CreateArray(n · n)
3  for i ← 0 to n do
4      for j ← 0 to n do
5          if W[i, j] ≠ 0 then
6              dist[i, j] ← W[i, j]
7              pred[i, j] ← i
8          else
9              dist[i, j] ← +∞
10             pred[i, j] ← −1
11 for k ← 0 to n do
12     for i ← 0 to n do
13         if i ≠ k then
14             for j ← 0 to n do
15                 if j ≠ i then
16                     if dist[i, j] > dist[i, k] + dist[k, j] then
17                         dist[i, j] ← dist[i, k] + dist[k, j]
18                         pred[i, j] ← pred[k, j]
19 return (dist, pred)
```