Voting Strengths

A company wishes to select a catering service that will provide daily meals to its employees. After some market research, the company has found that there are three services that meet quality and price requirements. However, there is a catch. The first catering service, called "MeatLovers", or M for short, offers a selection of mostly meat menus, with a sprinkling of pasta dishes on the side. The second catering service, called "BitOfEverything", E for short, offers a more varied menu, ranging from meat to vegetarian dishes. The third catering service, called "VegForLife", or V, offers strictly vegetarian dishes.

The Human Resources department of the company decides to ask the employees about their preferred catering service. They put the matter to a vote. The results of the vote are as follows. 40% of the employees prefer M. 30% of the employees prefer E and 30% of the employees prefer V. The contract is awarded to M, as all due process has been followed.

Can you spot a problem? It is very unlikely that the vegetarians in the company would like to abide by such a decision, and yet they must; or rather they will have to bring their own food everyday.

If you pay attention, you may notice that the way the election was framed created the problem. Although 30% of the employees chose V, a relative minority, it is very probable that they prefer B to M. The M, on their part most likely prefer E to V. Finally, those who chose E may prefer V over M. Let's tabulate these preferences:

$$\begin{aligned} 40\%\colon & [M, E, V] \\ 30\%\colon & [V, E, M] \\ 30\%\colon & [E, V, M] \end{aligned}$$

Panos Louridas, Athens University of Economics and Business.

Now let's calculate the *pairwise preferences* among the choices, that is, for every pair of choices, how many of voters prefer one to the other?

We starting by examining M and E for the three groups of voters above. 40% of the voters prefer M to E, having M as their first choice. At the same time, 30% of the voters prefer E to M, with their first choice being V, and another 30% of the voters prefer E to M, with E being their first choice. That means that 60% of the voters prefer E to M. So E beats M by 60% to 40%.

We proceed with M and V. We reason in the same way. 40% of the voters prefer M to V, having M as their first choice. However, 30% of the voters prefer V to everything else, and the 30% of the voters that have E as their first choice also prefer V to M. So V beats M by 60% to 40%.

Finally, to compare E with V, 40% of the voters listed M as their first choice and prefer E to V. 30% of the voters listed V as their first choice and also prefer V to E. The 30% of the voters that listed E as their first choice prefer, obviously, E over V. Therefore E beats V by 70% to 30%.

Taking it all together, E beats both the other two contestants in their pairwise comparisons, M by 60% to 40% and V by 70% to 30%, while V beats one contestant, and M beats none. We therefore declare E the winner of the election process.

The example showed a problem that arises with a very well known and very popular election method, *plurality voting*. In plurality voting voters record on their ballot their first preference. The candidate with the most ballots wins the election. The problem with plurality voting is that the voters do not record their full set of preferences. They only record their top one. So, it is possible that one candidate may preferred to any other candidate, but that candidate will not win the elections if it is not the top choice of the majority of the voters.

The requirement that in an election the winner should be the candidate that when compared with any other candidate is preferred by most voters is called the *Condorcet criterion*. The winner is called the *Condorcet candidate*, or *Condorcet winner*. The name comes from Marie Jean Antoine Nicolas Caritat, the Marquis de Condorcet, an 18th century French mathematician and philosopher who described the situation in his 1785 book *Essai sur l'application de l'analyse á la probabilité des décisions rendues à la pluralité des voix* (*Essay on the Application of Analysis to the Probability of Majority Decisions*).

Lest you think that the Condorcet criterion is an arcane concept embedded in culinary discussions or 18th century France, consider two more recent examples.

In the 2000 U.S. Presidential Election the voters were called to select among George W. Bush, Al Gore, and Ralph Nader. In the U.S. the president is elected by an Electoral College. The Electoral College is composed 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. The final results gave to the candidates the following:

- George W. Bush received 2,912,790 votes, equal to 48.847% of voters.
- Al Gore received 2,912,253 votes, equal to 48.838% of voters.
- Ralph Nader received 97,421 votes, or 1.634% percent of voters.

George W. Bush won with a wafer-thin 537 vote margin, or 0.009% of the votes in Florida. It is generally thought, however, that most of the Ralph Nader voters preferred Al Gore to George W. Bush. If that is true, and the U.S. elections used a voting method that allowed voters to express their second choices as well, then Al Gore would be the winner.

Changing continents and moving to France, in the 2002 French Presidential Election on April 21, Jacques Chirac, Lionel Jospin, and fourteen other candidates were competing. To be elected president in France a candidate has to secure more than 50% of the vote. 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.

That does not mean that most of French voters support the far-right. The count of the first round on April 21, 2002 was:

- Jacques Chirac received 5,666,440 votes, equal to 19.88% of the vote.
- Jean-Marie Le Pen received 4,805,307 votes, equal to 16.86% of the vote.
- Lionel Jospin received 4,610,749 votes, equal to 16.18% percent of the vote.

In the second round of the elections, which took place two weeks later on May 5, 2002, Jacques Chirac got more than 82% of the vote, while Jean-Marie Le Pen got less than 18% of the vote. It seems that while Chirac got pretty much the votes of all the other fourteen eliminated candidates, Le Pen barely moved from his performance on the first round.

The problem arose because initially there were sixteen candidates, from which two would be chosen to go to the second round. With sixteen candidates, it is not difficult to find a situation where support among them is spread so much that an extreme candidate, with no support beyond its own supporters, will manage to get enough votes from voters with hardcore views to prevail over a more moderate candidate. More generally, a candidate that is fairly liked by most people, but is not necessarily their first choice, will fail against a candidate that is hated by most people, by is the first choice of some sizable minority.

Plurality voting is not the only voting method that fails the Condorcet criterion, but due to its popularity it is the one that can be easily picked upon by critics.

In *approval voting*, voters can select any number of candidates, not just one, on the ballot. The winner is the candidate with the most votes. Suppose in an election we have three candidates A, B, and C, and we obtained the following count (we do not use square brackets around the ballots to emphasise that the sequence does not matter):

$$\begin{aligned} 60\%&\colon A, B \\ 40\%&\colon C, B \end{aligned}$$

Since B was selected by all 100% of the voters, B carries the day. Suppose that the 60% of the voters favour A over B over C, and the 40% of the voters favour C over B over A, although they are not able to express that on the ballot. In other words, if they were, the ballots would be:

$$\begin{aligned} 60\%&\colon [A, B, C] \\ 40\%&\colon [C, B, A] \end{aligned}$$

Then A beats B by 60% to 40%; so although most voters prefer A to B, A is not elected.

Another voting method is the *Borda count*, named after another 18th century Frenchman, the mathematician and political scientist Jean-Charles de Borda, who described it in 1770. In Borda count, voters award points to the candidates. If there are n candidates, their first choice gets $(n-1)$ points, their second choice $(n-2)$ points, down to the last choice that will get zero points. The winner is the candidate that collects most points. Consider an election in which three candidates, A, B, and C compete and the ballots are as follows:

$$\begin{aligned} 60\%&: [A, B, C] \\ 40\%&: [B, C, A] \end{aligned}$$

If there are n voters, candidate A gets $(60 \times 2)n = 120n$ points, candidate B gets $(60 + 2 \times 40)n = 140n$ points and candidate C gets $40n$ points. The winner is candidate B. Yet, most candidates prefer candidate A to candidate B.

Returning to the Condorcet criterion itself, the problem is to find a method that will pick the Condorcet winner of an election, if such a winner exists. A Condorcet winner might not necessarily exist. For example, say we have three candidates A, B, and C, for which we received the following ballots:

$$\begin{aligned} 30&: [A, B, C] \\ 30&: [B, C, A] \\ 30&: [C, A, B] \end{aligned}$$

If we do the pairwise comparisons, we find that A beats B by 60 to 40; B beats C by 60 to 40; C beats A by 60 to 40. Therefore each candidate beats some other candidate, and no candidate beats more candidates than others, so no overall winner emerges.

Now see another election involving three candidates, where the ballots were cast in a different way:

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

A is preferred over B by 15 to 5; B is preferred over C by 15 to 5; and C and A tie up with 10 each. Since both A and B have a pairwise win, we cannot declare a winner.

That's a bit strange, because it is not a case where all ballots take an equal share of the voters, as the previous one. It is clear that more voters cast the first ballot, but somehow this does not make its way to the results. We would therefore like to have a method that respects the Condorcet criterion but is less prone to ties than the simple method we have been using till now.

A method that finds the Condorcet winner of an election, if such a winner exists, is the Schulze method, developed in 1997 by Markus Schulze. It is a method used very widely among technological organisations, and is not prone to ties. 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.

The first step in the Schulze method is exactly finding the pairwise preferences for the candidates. Suppose we have n ballots and m candidates. The ballots are $B_s = b_1, b_2, \ldots, b_n$ and the candidates are $C_s = c_1, c_2, \ldots, c_m$. For each pair of candidates c_i, c_j we want to find how many voters favour c_i over c_j. We take each ballot in turn. Each ballot contains a series of candidates, in decreasing preference order, so a candidate that precedes other candidates in a ballot is preferred by the voter that cast that ballot to any of the candidates that follow in the ballot.

To find the pairwise preferences for the candidates we use an array p of size $m \times m$. Each element of p, p_{c_i,c_j} shows how many voters prefer candidate c_i to candidate c_j. To calculate the contents of the array, we first initialise it to all zero values. Then it suffices to take each and every ballot in turn. We read the contents of the ballot. For every pair c_i and c_j of candidates in the ballot such that c_i precedes c_j we add one to element $p[c_i, c_j]$ of array p.

For example, a ballot might be $b = [c_1, c_3, c_4, c_2, c_5]$, meaning that the voter prefers candidate c_1 over all other candidates, candidate c_3 over candidates c_4, c_2, c_5, candidate c_4 over candidates c_2, c_5 and candidate c_2 over candidate c_5. In the ballot $b = [c_1, c_3, c_4, c_2, c_5]$ candidate c_1 is before candidates c_3, c_4, c_2, and c_5, so we add one to elements $p[c_1, c_3]$, $p[c_1, c_4]$, $p[c1_,c_2]$, and $p[c_1, c_5]$. Candidate c_3 is before candidates c_4, c_2, and c_5, so we add one to elements $p[c_3, c_4]$, $p[c_3, c_2]$, and $p[c_3, c_5]$, and so on, up to candidate c_2 where we add one to element $p[c_2, c_5]$. Algorithm 1 does that.

Algorithm 1: Calculate pairwise preferences.

```
Input: ballots, n, m: ballots is an array of ballots, of size n. Each
       ballot is an array of candidates. m is 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 $j$.

for $i = 0$ to $m$ do
    for $j = 0$ to $m$ do
        $p[i][j] \leftarrow 0$

for $i = 0$ to $n$ do
    $b \leftarrow ballots[i]$
    for $j = 0$ to Size($b$) do
        $f \leftarrow b[j]$
        for $k = j + 1$ to Size($b$) do
            $s \leftarrow b[k]$
            $p[f, s] \leftarrow p[f, s] + 1$

return $p$
```

The algorithm initialises the pairwise preferences for all candidates to 0 in lines 1–3. This requires $\Theta(|B_s|^2)$ time. Then it takes each of the $|B_s|$ ballots. In each ballot, it starts with the first candidate. If the candidate is c_i, then for every other candidate c_j that follows c_i in the ballot it adds up one to the element $p[c_i, c_j]$. It does the same thing starting with all the other candidates in the ballot in sequence. If the ballot contains all $|C_s|$ candidates, it will update the preferences array $(|C_s| - 1) + (|C_s| - 2) + \ldots + 1 = |C|(|C| - 1)/2$ times. In the worst case all ballots contain all $|C_s|$ candidates, so the time required for each ballot is $O(|C_s|(|C_s| - 1)/2) = O(|C_s|^2)$ and the time for all ballots is $O(|B_s||C_s|^2)$. In total, Algorithm 1 runs in $O(|C_s|^2 + |B_s|^2)$ time.

As an example, take an election with four candidates, A, B, C, and D. In the election took part 21 voters. After counting the ballots we found that the following had been cast:

$$\begin{array}{c}6 \times [A, C, D, B] \\ 4 \times [B, A, D, C] \\ 3 \times [C, D, B, A] \\ 4 \times [D, B, A, C] \\ 4 \times [D, C, B, A]\end{array}$$

That is, six ballots $[A, C, D, B]$, four ballots $[B, A, D, C]$, and so on. In the first six ballots the voters preferred candidate A to candidate C, candidate C to candidate D, and candidate D to candidate B.

To calculate the preference array we find that candidate A is preferred to B in only the first set of ballots, so the entry in the array for the pairwise preferences between A and B will be six. Similarly, we find that candidate A is preferred to C in the first, second, and fifth set of ballots, so the pairwise preferences between A and C will be fourteen. Continuing in this way we find that the preference array for the candidates of our election is:

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

that is:

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

We then construct a graph, where the candidates are the nodes and the margins of the preferences of one candidate over another are the weights of the links. If for two candidates c_i and c_j the number $p[i, j]$ of voters that prefer c_i over c_j is greater than the number of voters $p[j, i]$ that prefer c_j over c_i, we add the link $c_i \rightarrow c_j$ and we assign the number $(p[i, j] - p[j, i])$ as the weight of the link $c_i \rightarrow c_j$. We use $-\infty$ for the other pairs to show that the corresponding link does not exist. To do that we start by doing the comparisons and the operations as necessary:

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

and then substituting $-\infty$ for negative and zero entries:

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

You can see the corresponding graph in Figure 1.1. As explained, the graph has the candidates as its nodes, and the positive margins of the differences in preferences between each pair of candidates as its edges.

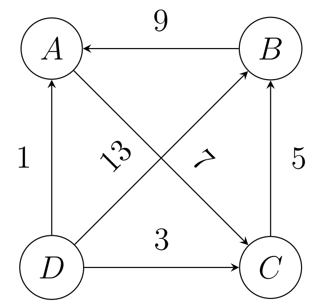


Figure 1.1: An election graph.

The next step is the calculation of the *strongest paths* between all nodes in the preference graph. We define the *strength of a path*, or equivalently the *width of a path* as the minimum weight in the links that make up the path. If you imagine a path as a sequence of bridges between nodes, the path is only as strong as its weakest link, or bridge. There may be multiple paths between two nodes in the preference graph, each with a difference strength. The path with the greatest strength among them is the strongest path. Returning to the bridges metaphor, it is the path that allows us to transport the heaviest vehicle between the two nodes. Figure 1.2 shows a graph and two strongest paths. The strongest path between node 0 and node 4 passes through node 2, and has strength 5; similarly, the strongest path between node 4 and node 1 passes through node 3 and has strength 7.

Finding the strongest paths is a problem that arises in other areas as well. In computer networks, it is equivalent to finding the maximum bandwidth between two computers in the internet when between any two computers or routers in the network the link has a limited bandwidth capacity. It is also called *maximum capacity path problem*, because it boils down to finding the maximum capacity of a path in a graph. The maximum capacity of the path is constrained by its weakest link; the maximum capacity path between two nodes, is the maximum capacity among the capacities of the paths between the two nodes.

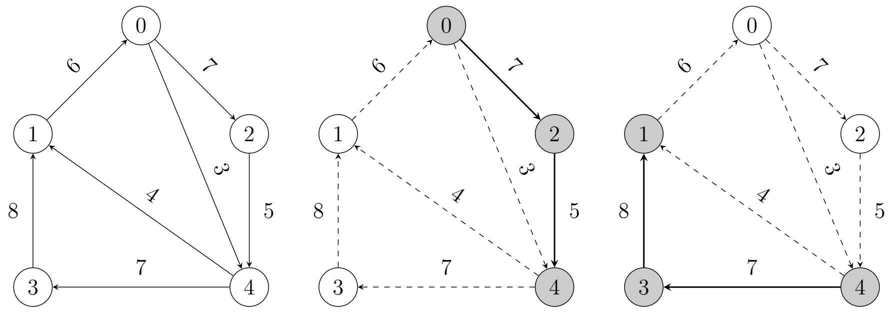


Figure 1.2: Strongest path examples.

To find the strongest path between all pairs of nodes in the graph, we reason as follows. We take all nodes of the graph in sequence, $c_1, c_2, \ldots, c_n$. We find the strongest path between all pairs of nodes, c_i and c_j in the graph using zero intermediate nodes from the sequence $c_1, c_2, \ldots, c_n$. The strongest path between the two nodes using no intermediate nodes exists if there is a direct link between c_i and c_j; otherwise it does not exist at all.

We proceed to find again the strongest path between all pairs of nodes c_i and c_j using the first of the nodes in the sequence, node c_1, as an intermediate node. If we have already found a strongest path between c_i and c_j in the previous step, then if there exist paths $c_i \to c_1$ and $c_1 \to c_j$ we compare the strength of the path $c_i \to c_j$ with the strength of the path $c_i \to c_1 \to c_j$, and we take the strongest of the two as our new estimate for the strongest path between c_i and c_j.

We continue the same procedure until we have used all n nodes in the sequence. Suppose we have found the strongest path between all pairs of nodes in the graph, c_i and c_j, using the first k nodes of the sequence as intermediate nodes. We now try to find the strongest path between the two nodes c_i and c_j using the first $(k+1)$ nodes in the sequence. If we find a path from c_i to c_j using the $(k+1)$th node, it will consist of two parts. The first part will be a path from c_i to c_{k+1} using the first k nodes of the sequence as intermediate nodes, and the second part will be from c_{k+1} to c_j using again the first k nodes of the sequence as intermediate nodes. We have already found the strengths of these two paths in the previous steps of the procedure. If we call $s_{i,j}(k)$ the strength of the path from c_i to c_j using the first k nodes, the two paths using node $(k+1)$ are $s_{i,k+1}(k)$ and $s_{k+1,j}(k)$. The strength of the path going from c_i to c_j through c_{k+1} is by definition the minimum of the strength of the paths $s_{i,k+1}(k)$ and $s_{k+1,j}(k)$ and we have already found both of them previously. So it is:

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

At the end, after using all n nodes in the sequence, we will have found all the strongest paths between any two pairs in the graph. Algorithm 2 shows the procedure in detail.

Lines 1–8 of the algorithm initialise the strengths of the path between two nodes to their direct links between them, if they exist. They correspond to finding the strongest paths using zero intermediate nodes. Then, lines 9–16 calculate strongest paths using more and more intermediate nodes. The outer loop, over variable k, corresponds to the intermediate node we are adding to our set of intermediate nodes. Each time we increase k we check all pairs of nodes, given by i and j, and adjust their strongest path estimates, if needed, in lines 14–16. We also keep track of the paths themselves using an array *pred* that gives, at each position (i, j) in the array, the predecessor node along the strongest path from node i to node j.

The algorithm is efficient. The first loop in lines 1–8 executes n^2 times and the second loop in lines 9–16 executes n^3 times, where n is the number of vertices in the graph. Since the graph is represented as an adjacency matrix, all graph operations take constant time, so in total it takes $\Theta(n^3)$ time. If we take the discussion to elections, then n is the number of candidates. In the notation we have adopted, the time required is $\Theta(|C_s|^3)$.

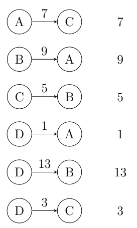
Algorithm 2: Calculate strongest paths.

Input: w, n: w is 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.

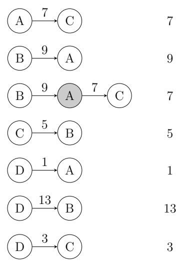
Output: $(s, pred)$: s is an array of size $n \times n$ such that $s[i, j]$ is the strongest path between nodes i and j. $pred$ is an array of size $n \times n$ such that $pred[i, j]$ is the predecessor of node i in the strongest path to node j.

```
1  for $i = 0$ to $n$ do
2      for $j = 0$ to $n$ do
3          if $w[i,j] > w[j,i]$ then
4              $s[i][j] \leftarrow w[i,j] - w[j,i]$
5              $pred[i,j] \leftarrow i$
6          else
7              $s[i][j] \leftarrow -\infty$
8              $pred[i,j] \leftarrow -1$

9  for $k = 0$ to $n$ do
10     for $i = 0$ to $n$ do
11         if $i \neq k$ then
12             for $j = 0$ to $n$ do
13                 if $j \neq i$ then
14                     if $s[i,j] < min(s[i,k], s[k,j])$ then
15                         $s[i,j] \leftarrow min(s[i,k], s[k,j])$
16                         $pred[i,j] \leftarrow pred[k,j]$;

17 return $(s, pred)$
```

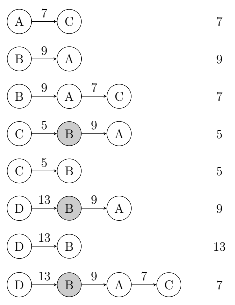
You can see a trace of the execution of the algorithm in our example in Figure 1.3. At each sub-figure we indicate with a gray fill the intermediate node that is used to form a new path at each step. Note that in the last step, when we add node D in our set of intermediate nodes, we find no path stronger from those already found. This may happen, but we cannot know it in advance. It is also possible to happen in one of the previous steps, even though it did not happen in this case. Be it as it may, we have to carry out the execution of the algorithm to the end, examining all nodes of the graph in sequence as intermediate nodes.



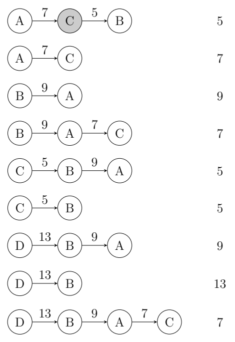
(a) No intermediate nodes.



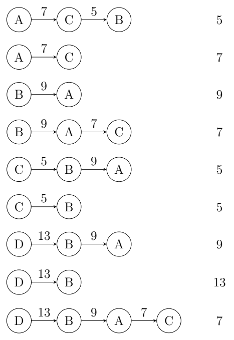
(b) *A* as intermediate node.



(c) *A*, *B* as intermediate nodes.



(d) *A*, *B*, *C* as intermediate nodes.



(e) *A*, *B*, *C*, *D* as intermediate nodes.

Figure 1.3: Calculation of strongest paths.

The paths are returned from Algorithm 2 via array *pred*. If you want to see the paths on the graph itself, you can check them in Figure 1.4. Verify that the strongest path from node D to node C is through nodes B and A, and not the direct path from D to C; at the third sub-figure the earlier estimate gets overwritten by the better estimate when we use node B as an intermediate node. By contrast, when we add A as an intermediate node we can obtain the path $D \rightarrow A \rightarrow B$ with strength 1; but we already have path $D \rightarrow A$ with strength 1, so there is no need to update the strongest path between A and D. By the same token, we can obtain the path $D \rightarrow A \rightarrow C$ with strength 1; but we already have path $D \rightarrow C$ with strength 3, so the strongest path between D and C remains unchanged.

What the algorithm really does is to calculate parts of the solution to our problem and combine them incrementally to arrive at the overall solution. It finds shorter strongest paths that it combines, if possible, producing longer strongest paths. This strategy, of solving part of the problem and combining the solved parts to produce the final solution, is called *dynamic programming* and lies behind many interesting algorithms.

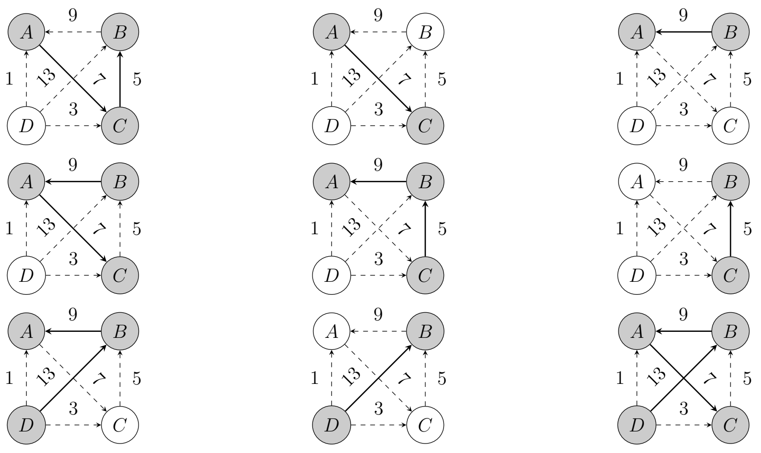


Figure 1.4: Strongest paths for the election graph.

In our example Algorithm 2 produces an array s with the strengths of the strongest paths between any pair of nodes:

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

With these available, we have established, for any two candidates c_i and c_j how much support there is for candidate c_i to c_j as well as how much support there is for candidate c_j to c_i. They are the strengths of the paths from c_i to c_j and from c_j to c_i respectively. 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, then we say that candidate c_i wins over candidate c_j. We want then to find, for each candidate c_i, over how many other candidates c_i wins. That is easy to do, by just going over the array with the path strengths that we calculated using Algorithm 2 and adding up the times c_i is preferred over c_j. We do this in Algorithm 3, which returns a list *wins* such that item i of the list *wins* contains a list with the candidates over which candidate i prevails. We only need a function `Add(`*l, i*`)` that adds item i to list l; in line 2 of the algorithm we use [] to denote the empty list.

Algorithm 3: Calculate results.

```
Input: $s$, $n$: an array of size $n \times n$ with the strongest paths between
       nodes; $s[i, j]$ is the strongest path between nodes $i$ and $j$.
Output: $wins$: a list 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 for $i = 0$ to $n$ do
2     $listi \leftarrow []$
3     Add($wins$, $listi$)
4     for $j = 0$ to $n$ do
5         if $i \neq j$ then
6             if $s[i, j] > s[j, i]$ then
7                 Add($listi$, $j$)
8 return $wins$
```

In our example we find that A beats C, B beats A and C, C is beaten by all, and D beats A, B, and C. In particular, $wins = [[2], [2, 0], [], [0, 1, 2]]$. Since the number of times a candidate wins other candidates is $A = 1$, $B = 2$, $C = 0$ and $D = 3$, D is the preferred candidate. Algorithm 3 requires $O(n^2)$ time, where n is the number of candidates, or $|C_s|$ in the notation that we have been using, so the time required is $O(|C_s|^2)$. Since Algorithm 1 runs in $O(|C_s|^2 + |B_s|^2)$ time, Algorithm 2 runs in $\Theta(|C_s|^3)$ time, and Algorithm 3 runs in $O(|C_s|^2)$ time, the whole Schulze method requires polynomial time and is efficient.

Note that we did not just get a winner for the election; we got an ordering of the candidates. That means that the Schulze method can be used also when we want to pick the first k out of a total n candidates. We just select the first k in the ordering it produces.

In our example we did not have ties. Candidates were ordered in a clear way. That may not always happen. The Schulze method will produce a Condorcet winner, when there is one, but of course it cannot invent one if there is none. Also, there may may be ties lower down in the order, with an overall winner and two candidates tied up in second place. For example, there could be an election scenario in which D would beat B, C, and D, while A would beat C, B would beat D, and C would beat nobody. D would be the winner and A and B would both get second place.

Now let's return to the example that got us started on the Schulze method. Recall that we had the following ballots for three candidates, A, B, and C:

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

We found that with simple comparisons, without using the Schulze method, A is preferred over B by 15 to 5, B is preferred over C by 15 to 5 and C and A tie up with 10 each. Since both A and B have a pairwise win, we had ended up with a tie. How does the Schulze method fare? Is it worth the trouble? The preference array for the candidates is:

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

From that we get the adjacency matrix for the election graph:

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

The graph itself is in Figure 1.5. It's easy to see that there are two (strongest) paths starting from A, the path $A \to B$ and the path $A \to B \to C$, while there is only one strongest path from B, $B \to C$. As there are no reverse paths to compare, the Schulze method will give as a result that A wins over both B and C, B wins over C, and C beats nobody. It will therefore declare A as a winner, resolving the earlier tie. The Schulze method will in general produce less ties than a simple pairwise comparison. It can be proven that it meets the criterion of *resolvability*, which means that there is a low possibility of tied results.

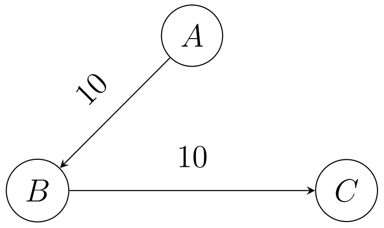


Figure 1.5: Another election graph.

Caveat emptor. We have presented the Condorcet criterion and showed that it is a reasonable requirement for an election. We have also presented the Schulze method, which is a reasonable voting method that meets the Condorcet criterion. That does not mean that the Schulze method is the one best voting method, or even that the Condorcet criterion is the one criterion that should guide a polity's decisions as it looks to decide how to run its elections. It can be proven, as it was by the Nobel prize winner Kenneth Joseph Arrow in his PhD thesis, that there exists no perfect voting system where voters express their preferences in the ballots. That is called Arrow's Impossibility Theorem. What is important is that the choice of a voting method should be an informed one, and not made unthinkingly, or perhaps guided by custom. Voters must decide how to vote wisely. That said, the Schulze method does provide a nice example to introduce a nice graph algorithm and that is the real reason why we chose it here.

Algorithm 2 is a variant of a venerable algorithm for calculating the shortest paths between all pairs in a graph, the Floyd-Warshall algorithm. It was published by Robert Floyd in 1962, but similar algorithms have been published by Bernard Roy in 1959 and Stephen Warshall in 1962. This is Algorithm 4. Like Algorithm 2, this also runs in time $\Theta(n^3)$. That makes it in in general slower than the Dijkstra algorithm, but it performs well in dense graphs. It is also very simple to implement, with no requirements for any special data structures, and it works in graphs with negative weights as well. Attention is needed in line 14 to watch for overflows in computer languages that have no notion of infinity and you use big numbers to stand in for them. In these settings you need to check whether $dist[i,k]$ and $dist[k,j]$ are not equal to what you use to represent ∞. If they are, there is no point in adding them and the shortest path from i to j will not go through them anyway.

Algorithm 4: Floyd-Warshall all pairs shortest paths.

Input: w, n: w is 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.

Output: $(dist, pred)$: $dist$ is an array of size $n \times n$ such that $dist[i, j]$ is the shortest path between nodes i and j. $pred$ is 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  for $i = 0$ to $n$ do
2      for $j = 0$ to $n$ do
3          if $w[i, j] \neq -\infty$ then
4              $dist[i, j] \leftarrow w[i, j]$
5              $pred[i, j] \leftarrow i$
6          else
7              $dist[i, j] \leftarrow +\infty$
8              $pred[i, j] \leftarrow -1$
9  for $k = 0$ to $n$ do
10     for $i = 0$ to $n$ do
11         if $i \neq k$ then
12             for $j = 0$ to $n$ do
13                 if $j \neq i$ then
14                     if $dist[i, j] > dist[i, k] + dist[k, j]$ then
15                         $dist[i, j] \leftarrow dist[i, k] + dist[k, j]$
16                         $pred[i, j] \leftarrow pred[k, j]$;
17 return $(dist, pred)$
```