The Bellman-Ford algorithm
COMS20010 2020, Video 11-3

John Lapinskas, University of Bristol

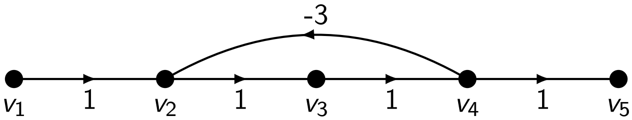
Shortest paths with negative-weight edges

The **length** of a path/walk $P = x_1 \dots x_t$ is the total weight $\sum_{i=1}^{t-1} w(x_i, x_{i+1})$ of P's edges.

The **distance** from x to y is the shortest length of any path/walk from x to y, or ∞ if they are in different components.

We touched on negative-weight edges when we covered Dijkstra's algorithm in week 4, but now we can actually solve the problem.

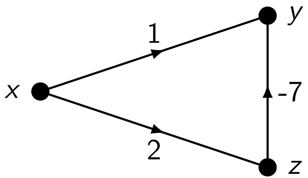
We assume every cycle in the graph has non-negative total weight — this guarantees that a shortest walk from one vertex to another exists, and is a path. Otherwise, it often doesn't exist!



Here there is no shortest walk from v_1 to v_5, since we can keep repeating the cycle $v_2 v_3 v_4$ to send the length of the walk off to $-\infty$...

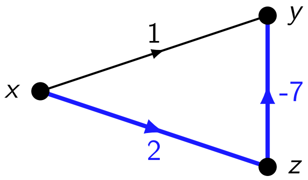
What goes wrong with Dijkstra?

Dijkstra's algorithm relies on the assumption that the best route out of a set X of vertices is determined by the graph's structure in and near X. With negative weights, this fails.



What goes wrong with Dijkstra?

Dijkstra's algorithm relies on the assumption that the best route out of a set X of vertices is determined by the graph's structure in and near X. With negative weights, this fails.



Since (x, y) has lower weight than (x, z), Dijkstra's algorithm run from x finalises $d(x, y) = 1$ as its first step even though $d(x, y) = -5$. It can't "see" the weight-(-7) edge when it's finalising the distance of y.

A dynamic programming approach

Step 1: Find a slow algorithm by reducing the problem to itself.

Original problem: Given a weighted digraph G with no negative-weight cycles and vertices $s, t \in V(G)$, find a shortest path from s to t.

Remember, when a solution is composed of lots of separate choices, a good way of going about this is often to consider the results of each choice.

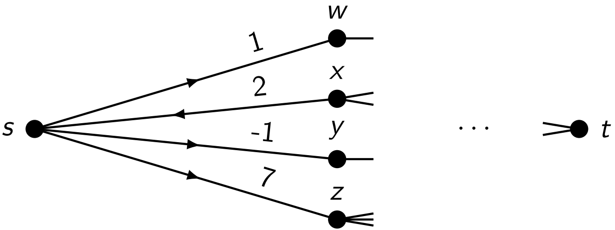
A dynamic programming approach

Step 1: Find a slow algorithm by reducing the problem to itself.

Original problem: Given a weighted digraph G with no negative-weight cycles and vertices $s, t \in V(G)$, find a shortest path from s to t.

Remember, when a solution is composed of lots of separate choices, a good way of going about this is often to consider the results of each choice.

Here, a good first choice is: which edge do we take out of s?



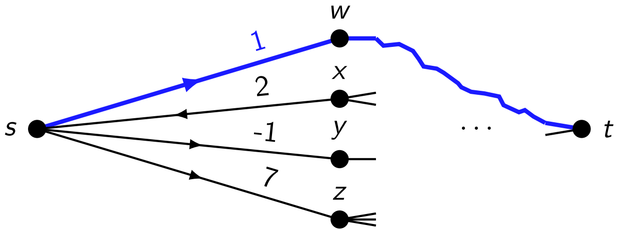
A dynamic programming approach

Step 1: Find a slow algorithm by reducing the problem to itself.

Original problem: Given a weighted digraph G with no negative-weight cycles and vertices $s, t \in V(G)$, find a shortest path from s to t.

Remember, when a solution is composed of lots of separate choices, a good way of going about this is often to consider the results of each choice.

Here, a good first choice is: which edge do we take out of s?



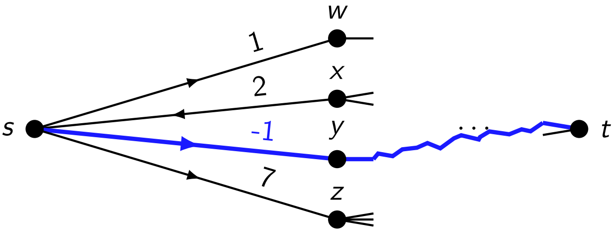
A dynamic programming approach

Step 1: Find a slow algorithm by reducing the problem to itself.

Original problem: Given a weighted digraph G with no negative-weight cycles and vertices $s, t \in V(G)$, find a shortest path from s to t.

Remember, when a solution is composed of lots of separate choices, a good way of going about this is often to consider the results of each choice.

Here, a good first choice is: which edge do we take out of s?



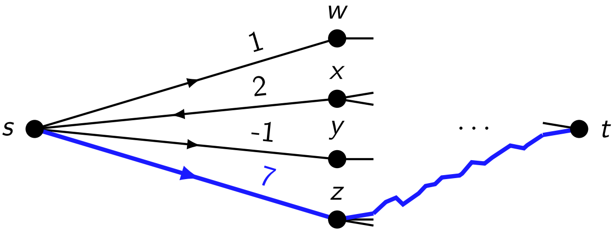
A dynamic programming approach

Step 1: Find a slow algorithm by reducing the problem to itself.

Original problem: Given a weighted digraph G with no negative-weight cycles and vertices $s, t \in V(G)$, find a shortest path from s to t.

Remember, when a solution is composed of lots of separate choices, a good way of going about this is often to consider the results of each choice.

Here, a good first choice is: which edge do we take out of s?



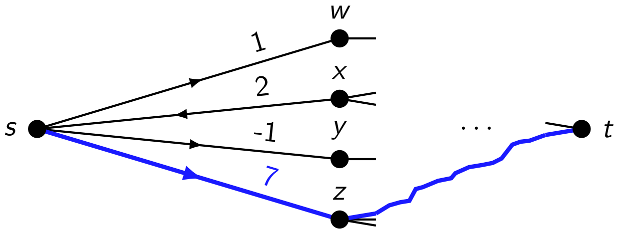
A dynamic programming approach

Step 1: Find a slow algorithm by reducing the problem to itself.

Original problem: Given a weighted digraph G with no negative-weight cycles and vertices $s, t \in V(G)$, find a shortest path from s to t.

Remember, when a solution is composed of lots of separate choices, a good way of going about this is often to consider the results of each choice.

Here, a good first choice is: which edge do we take out of s?



Any shortest path must be an edge from s to some $v \in N^+(s)$, followed by a shortest path from v to t in $G - s$.

The slow recursive algorithm

Algorithm: BadPath

Input : A weighted digraph $G = ((V, E), w)$ with no negative-weight cycles, and two vertices $s, t \in V(G)$.

Output : A shortest path from s to t in G, or `None` if none exists.

begin
  if $s = t$ **then**
    Return the empty path.
  if $d^+(s) = 0$ **then**
    Return `None`.
  Write $N^+(s) = \{v_1, \dots, v_d\}$, where $d \geq 1$.
  Let $P_i \leftarrow \textsc{BadPath}(G - s, v_i, t)$ for all $i \in [d]$.
  if $P_i = \texttt{None}$ *for all* $i \in [d]$ **then**
    Return `None`.
  Return whichever path is shortest in $\{sv_iP_i : i \in [d], P_i \neq \texttt{None}\}$.

How many possible calls are there to BadPath?

The slow recursive algorithm

Algorithm: BadPath

Input : A weighted digraph $G = ((V, E), w)$ with no negative-weight cycles, and two vertices $s, t \in V(G)$.

Output : A shortest path from s to t in G, or `None` if none exists.

begin

    if $s = t$ **then**

        Return the empty path.

    if $d^+(s) = 0$ **then**

        Return `None`.

    Write $N^+(s) = \{v_1, \dots, v_d\}$, where $d \geq 1$.

    Let $P_i \leftarrow \textsc{BadPath}(G - s, v_i, t)$ for all $i \in [d]$.

    if $P_i =$ `None` *for all* $i \in [d]$ **then**

        Return `None`.

    Return whichever path is shortest in $\{sv_iP_i : i \in [d], P_i \neq \texttt{None}\}$.

How many possible calls are there to BadPath? If the input graph is a clique, there are $\Theta(|V|2^{|V|})$ — G could be any of the $2^{|V|}$ induced subgraphs, and s could be any of the $|V|$ vertices!

So we can't just memoise this — we need to consolidate the calls.

The hard part: consolidating calls!

We can get around this by using two common tricks in dynamic programming: **reframing the problem** and **adding a parameter**.

Instead of asking for a shortest **path** from s to t in G, we will ask for a shortest **walk** from s to t in G **with at most** $|\boldsymbol{V(G)}| - \mathbf{1}$ **edges**.

The hard part: consolidating calls!

We can get around this by using two common tricks in dynamic programming: **reframing the problem** and **adding a parameter**.

Instead of asking for a shortest **path** from s to t in G, we will ask for a shortest **walk** from s to t in G **with at most** $\boldsymbol{|V(G)| - 1}$ **edges**.

Remember, when there are no negative-weight cycles, the shortest walk will be a path, and all paths have length at most $|V(G)| - 1$! So we're still asking for the same thing.

But the new formulation gives a much better recursive algorithm.

The hard part: consolidating calls!

We can get around this by using two common tricks in dynamic programming: **reframing the problem** and **adding a parameter**.

Instead of asking for a shortest **path** from s to t in G, we will ask for a shortest **walk** from s to t in G **with at most $|V(G)| - 1$ edges**.

Remember, when there are no negative-weight cycles, the shortest walk will be a path, and all paths have length at most $|V(G)| - 1$! So we're still asking for the same thing.

But the new formulation gives a much better recursive algorithm.

Most of dynamic programming is "cookie-cutter". It's not easy to learn, but once you know how, it's the same method for every problem. This is the part that can be arbitrarily difficult and only comes with practice.

A decent algorithm

Algorithm: GoodPath

Input : A weighted digraph $G = ((V, E), w)$ with no negative-weight cycles, two vertices $s, t \in V(G)$, and an integer $k \geq 0$.

Output : A shortest walk from s to t in G with at most k edges, or `None` if none exists.

begin

    if $k = 0$ **then**

        Return the empty walk if $s = t$, and `None` otherwise.

    Write $N^+(s) = \{v_1, \dots, v_d\}$, where $d \geq 1$.

    Let $P_i \leftarrow \textsc{GoodPath}(G, v_i, t, k-1)$ for all $i \in [d]$.

    if $P_i =$ `None` *for all* $i \in [d]$ **then**

        Return `None`.

    Return whichever walk is shortest in $\{sv_iP_i \colon i \in [d], P_i \neq \texttt{None}\}$.

How many distinct calls are there in $\textsc{GoodPath}(G, s, t, |V| - 1)$?

A decent algorithm

Algorithm: GoodPath

Input : A weighted digraph $G = ((V, E), w)$ with no negative-weight cycles, two vertices $s, t \in V(G)$, and an integer $k \geq 0$.

Output : A shortest walk from s to t in G with at most k edges, or `None` if none exists.

```
begin
    if k = 0 then
        Return the empty walk if s = t, and None otherwise.
    Write N⁺(s) = {v_1, ..., v_d}, where d ≥ 1.
    Let P_i ← GoodPath(G, v_i, t, k − 1) for all i ∈ [d].
    if P_i = None for all i ∈ [d] then
        Return None.
    Return whichever walk is shortest in {s v_i P_i : i ∈ [d], P_i ≠ None}.
```

How many distinct calls are there in $\textsc{GoodPath}(G, s, t, |V| - 1)$?

Only $|V|^2$! (One for each possible (k, s) pair, since G and t stay the same between calls.)

Each call takes $O(|V|)$ time, so if we memoise, the algorithm runs in total time $O(|V|^3)$. And as a bonus, we can get $d(v, t)$ for all $v \in V$ for free.