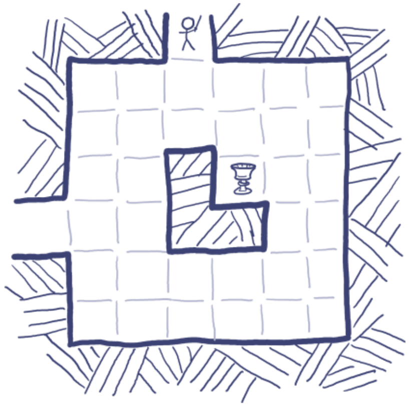
SECTION 5.6

Bellman-Ford

How can we find minimum-cost paths in graphs where some edge ~~costs~~ weights may be negative?

Dijkstra	Bellman-Ford
Can get stuck in ∞ loop if some weights are –ve	always terminates
$O(E + V \log V)$ if all weights ≥0	$O(V\ E)$



SECTION 5.7

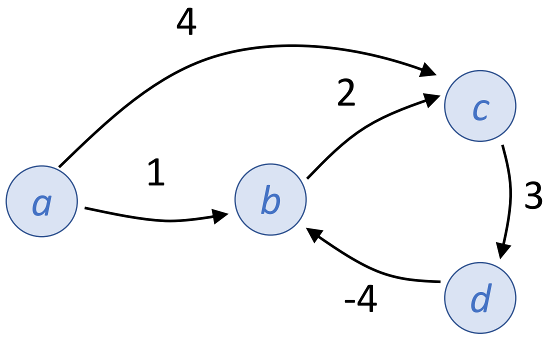
Dynamic programming


60
OXYGEN


60
OXYGEN
value function

Value function

Let $F(v)$ be the expected future reward that can be gained starting from state v



Bellman equation

$$F(v) = \max_{a} \left\{ \text{reward}_{v,a} + F(\text{nextstate}_{v,a}) \right\}$$

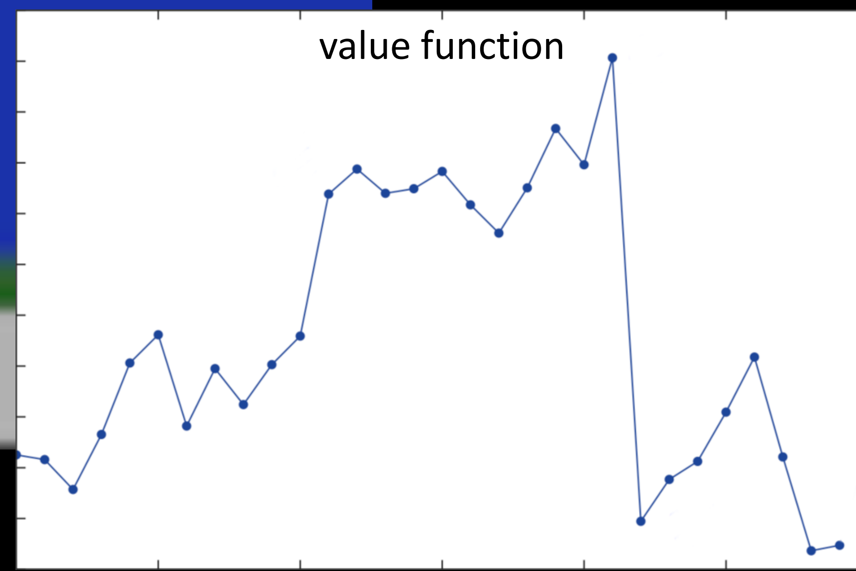
Theorem

Let g be a directed graph where each edge is labelled with a weight.
Assume g has no –ve weight cycles.

Then, $F_{d,V-1}(v)$ is the minimum weight from v to d
(over paths of *any* length).

Algorithm

Solve the Bellman recurrence equation.
[There's a nifty matrix trick for solving it for all pairs of vertices in $O(V^3 \log V)$.]



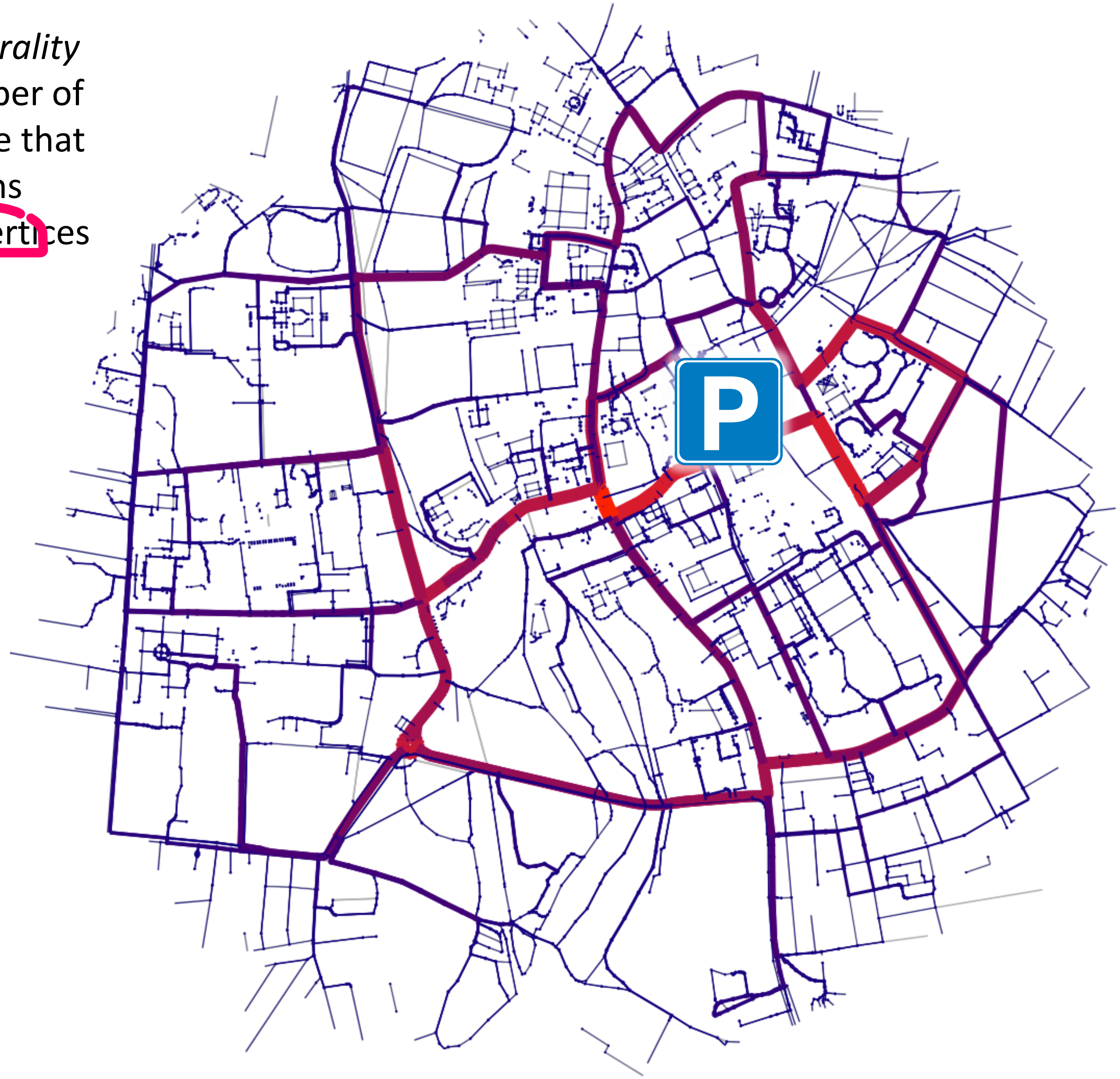
Playing Atari with Deep Reinforcement Learning, Silver et al., 2013. For interesting games it's impractical to solve $F(\cdot)$ exactly using Bellman's recursion; instead DeepMind used a neural network to approximate $F(\cdot)$.

SECTION 5.8

Johnson’s algorithm

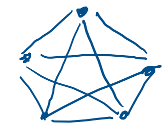
Definition

The *betweenness centrality* of an edge is the number of shortest paths that use that edge, considering paths between all pairs of vertices in the graph



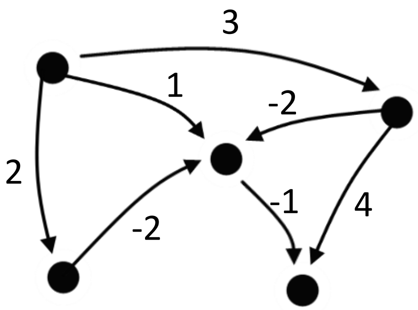
What's the cost of finding all-to-all minimum weights?

tree
$E=V-1$



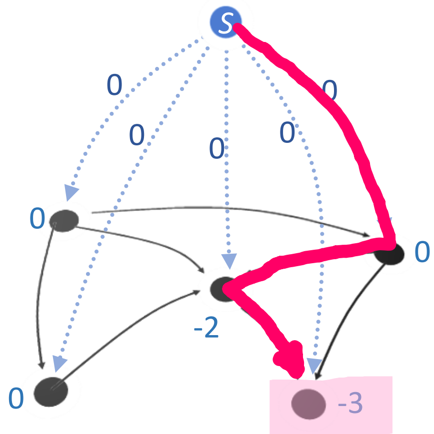
Fully connected
$E=V(V-1)$

		tree	Fully connected
$V \times$ Dijkstra	$V \times O(E + V \log V)$	$O(V^2 \log V)$	$O(V^3)$
$V \times$ Bellman-Ford	$V \times O(V\ E)$	$O(V^3)$	$O(V^4)$
Dynamic programming (with cunning matrix trick)	$O(V^3 \log V)$	$O(V^3 \log V)$	$O(V^3 \log V)$
Johnson	same as Dijkstra, but works with –ve edge weights		



0. The graph where we want all-to-all minweights

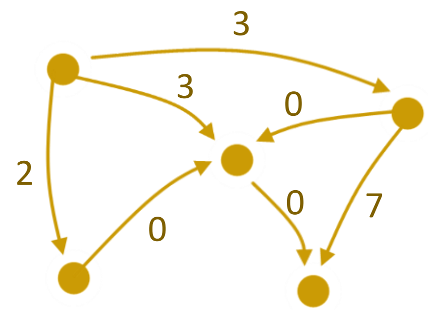
Let the edge weights be $w(u \to v)$



1. The augmented graph

Add a new vertex s, and run Bellman-Ford to compute minimum weights from s,

$$d_v = \text{minweight}(s \text{ to } v)$$



2. The helper graph

Define a new graph with modified edge weights

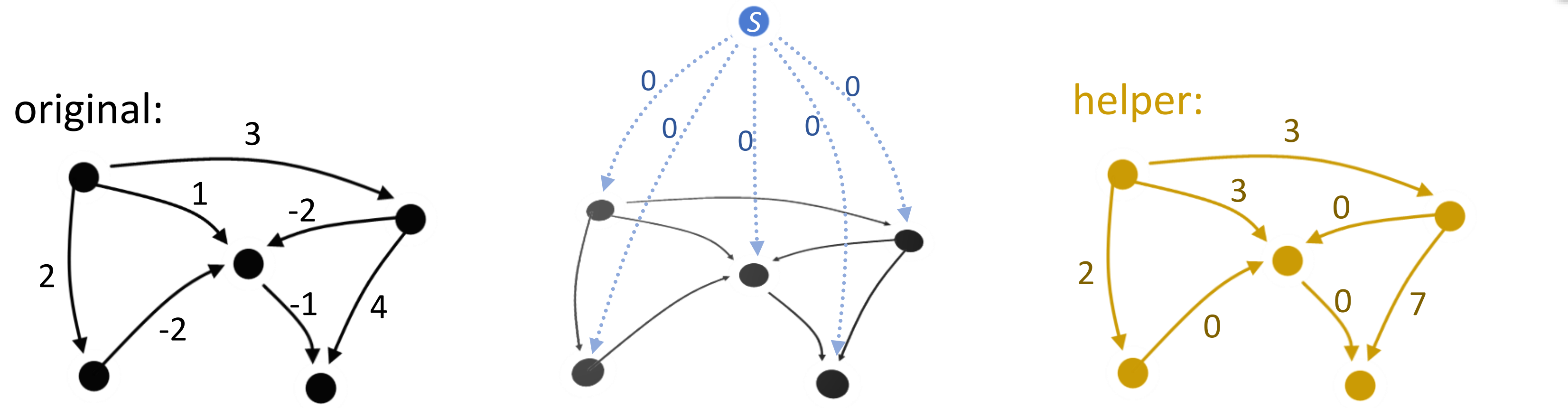
$$w'(u \to v) = d_u + w(u \to v) - d_v$$

3. Run Dijkstra to get all-to-all distances in the helper graph, $\text{distance}'(u \text{ to } v)$

4. Translation

$$\text{minweight}(p \text{ to } q) = \text{distance}'(p \text{ to } q) - d_p + d_q$$

Lemma. The edge weights in the helper graph are all ≥ 0



edge weights $w(u \to v)$

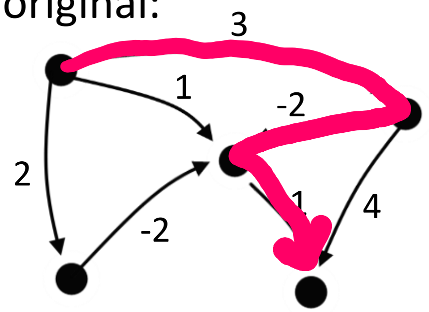
$d_v = \text{minweight}(s \text{ to } v)$

$w'(u \to v) = d_u + w(u \to v) - d_v$

Lemma. The translation step computes correct minweights:

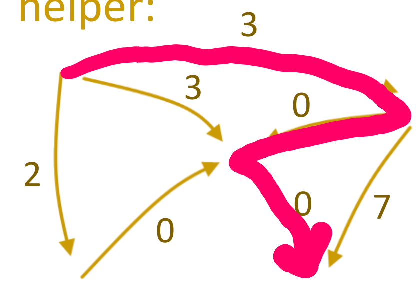
$$\text{minweight}(p \text{ to } q) = \text{distance}'(p \text{ to } q) - d_p + d_q$$

original:



edge weights $w(u \to v)$

helper:



$w'(u \to v) = d_u + w(u \to v) - d_v$