Optimisation and Operations Research

Lecture 16: Graph Problems and Dijkstra's algorithm

Matthew Roughan
<matthew.roughan@adelaide.edu.au>
http://www.maths.adelaide.edu.au/matthew.roughan/notes/OORII/

School of Mathematical Sciences,
University of Adelaide

August 13, 2019

Graph terminology

- A directed graph is a *tree* if it is connected and acyclic.
- A directed graph (N', L') is a *subgraph* of (N, L) if $N' \subseteq N$ and $L' \subset L$.
- A subgraph (N', L') is a *spanning tree* if it is a tree and $N' = N$.

Graph Terminology

Definition (A Path)

A **path** in a directed network $G(N, L)$ (of node set N and link set L) is a list of nodes,$i_1, i_2, i_3 ... i_{r-1}, i_r$, where

(i) $i_j \in N$ for all $j = 1...r$

(ii) for each successive pair of nodes, $(i_k, i_{k+1}) \in L$, and

(iii) with no repetition of nodes i.e $i_k \neq i_j$ when $k \neq j$.

Definition (A cycle)

A **cycle** in a directed network $G(N, L)$ is a list of nodes, $i_1, i_2, i_3 ... i_{r-1}, i_r, i_1$, where $i_1, i_2, i_3 ... i_{r-1}, i_r$ is a path and $(i_r, i_1) \in L$ (*i.e.*, the link from the last node of a path to the first is included).

Graph Terminology

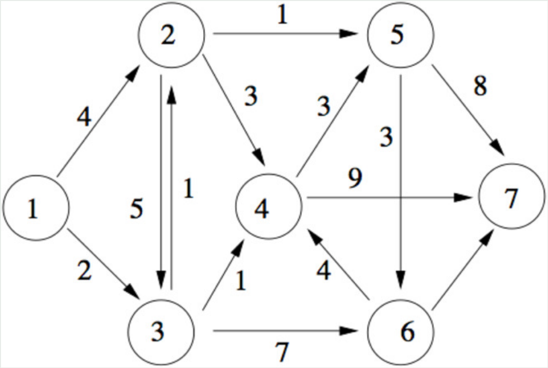
Example

Consider a path from node 1 to node 7

$1, 2, 4, 7,$
is one path,
while another is
$1, 3, 2, 4, 5, 6, 7.$

Note that $1, 2, 4, 6, 7$
is not a path,
because $(4, 6) \notin L$.

A (directed) cycle is
$4, 5, 6, 4.$



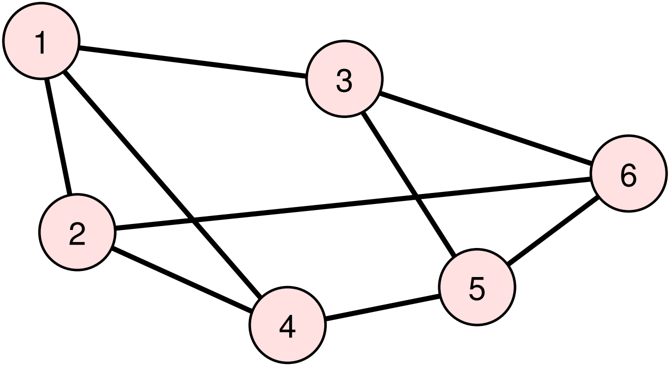
All Paths

- Origin-Destination (O-D) pair $(p, q) \in N \times N$
- Let K be the set of all O-D pairs, with $K = \{[p, q] : p, q \in N\}$.
- The set of **paths** joining an O-D pair (p, q) is denoted P_{pq}.
- The set of all paths in $G(N, L)$ is denoted P.

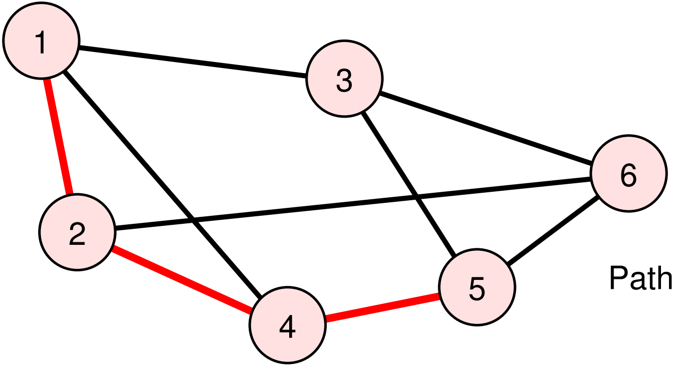
$$P = \cup_{[p,q]\in K} P_{pq}$$

- There can be exponentially many possible paths in a network

Network Paths

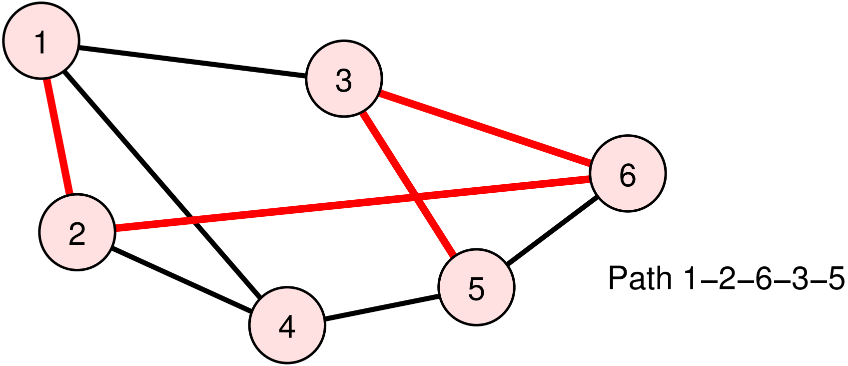


Network Paths



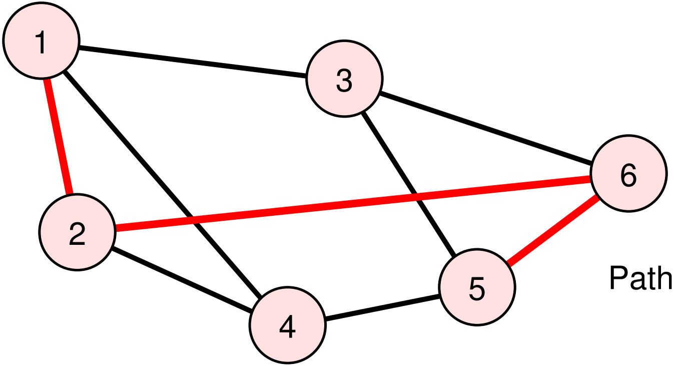
Paths P_{15}: 1-2-4-5

Network Paths



Paths P_{15}: 1-2-4-5, 1-2-6-3-5

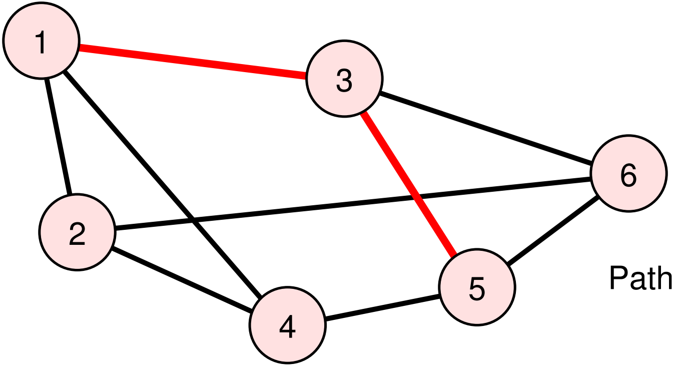
Network Paths



Path 1–2–6–5

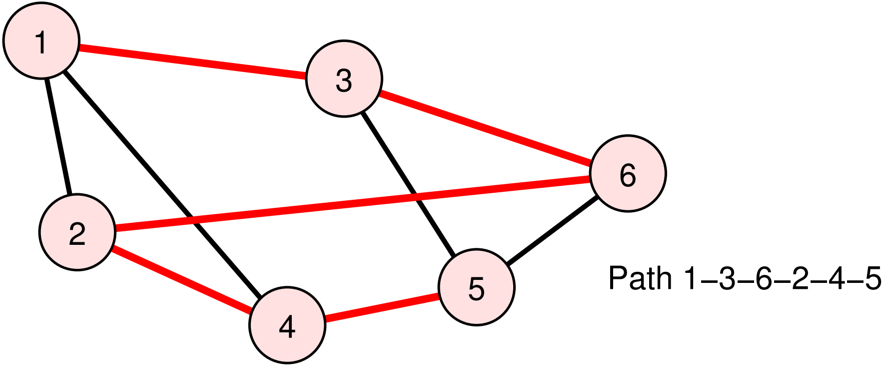
Paths P_{15}: 1-2-4-5, 1-2-6-3-5, 1-2-6-5

Network Paths



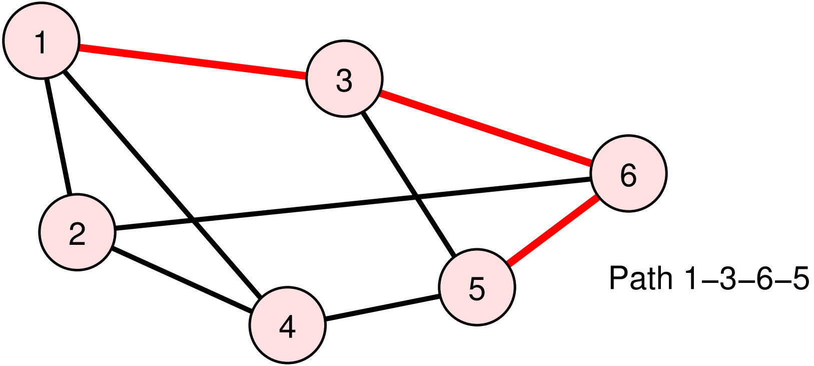
Paths P_{15}: 1-2-4-5, 1-2-6-3-5, 1-2-6-5, 1-3-5

Network Paths



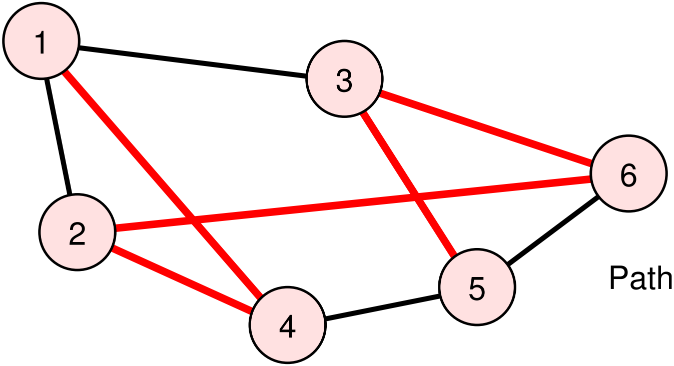
Paths P_{15}: 1-2-4-5, 1-2-6-3-5, 1-2-6-5, 1-3-5, 1-3-6-2-4-5

Network Paths



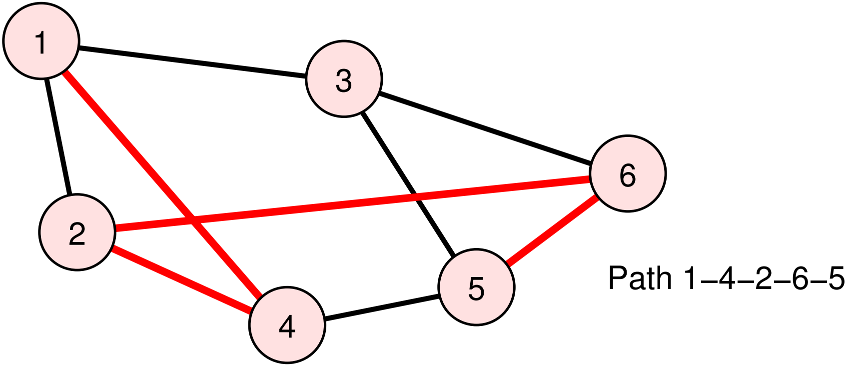
Paths P_{15}: 1-2-4-5, 1-2-6-3-5, 1-2-6-5, 1-3-5, 1-3-6-2-4-5, 1-3-6-5

Network Paths



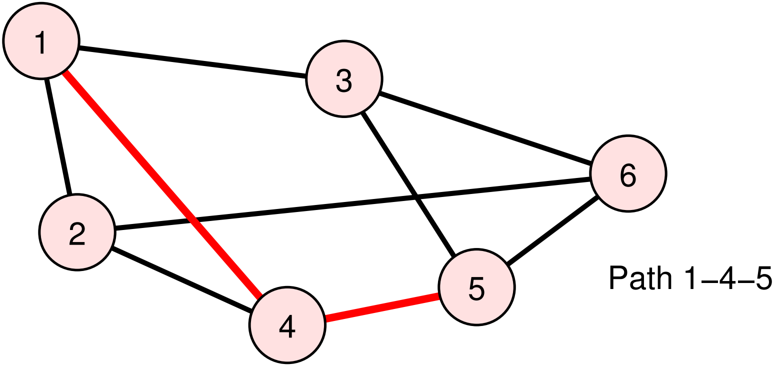
Paths P_{15}: 1-2-4-5, 1-2-6-3-5, 1-2-6-5, 1-3-5, 1-3-6-2-4-5, 1-3-6-5,
1-4-2-6-3-5

Network Paths



Paths P_{15}: 1-2-4-5, 1-2-6-3-5, 1-2-6-5, 1-3-5, 1-3-6-2-4-5, 1-3-6-5, 1-4-2-6-3-5, 1-4-2-6-5

Network Paths



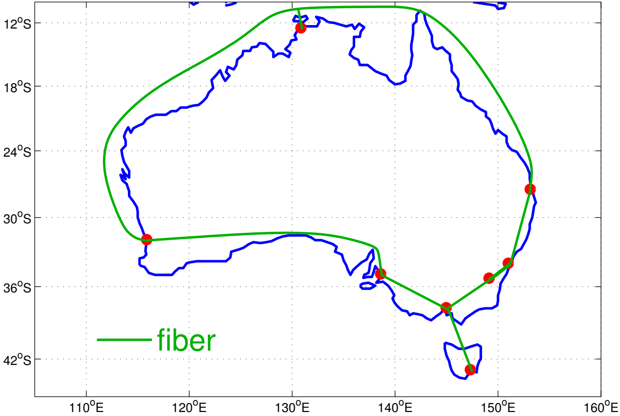
Paths P_{15}: 1-2-4-5, 1-2-6-3-5, 1-2-6-5, 1-3-5, 1-3-6-2-4-5, 1-3-6-5, 1-4-2-6-3-5, 1-4-2-6-5, 1-4-5

Section 1

Shortest Paths

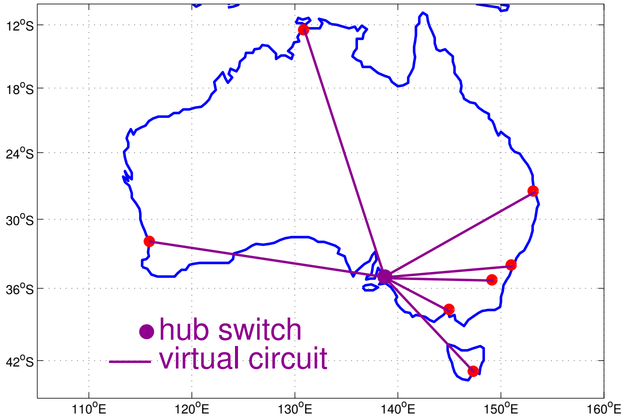
Logical vs. Physical Network

Imagine a physical network



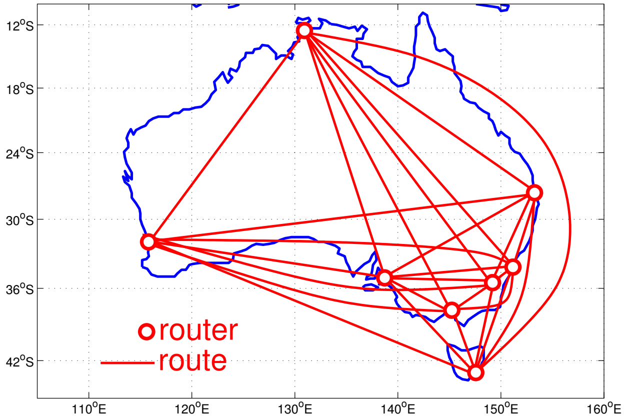
Logical vs. Physical Network

But a logical network that looks like



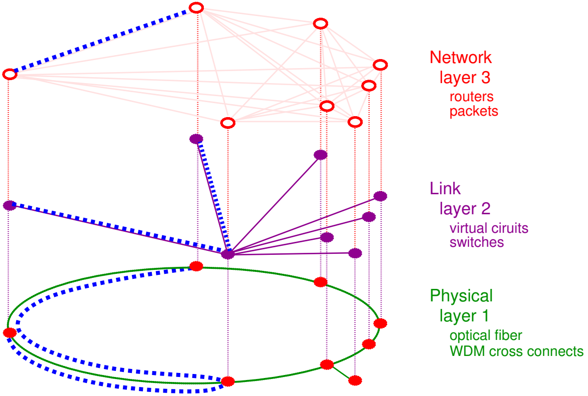
Logical vs. Physical Network

And potential pairs of network locations that want to communicate



Mapping the logical to the physical

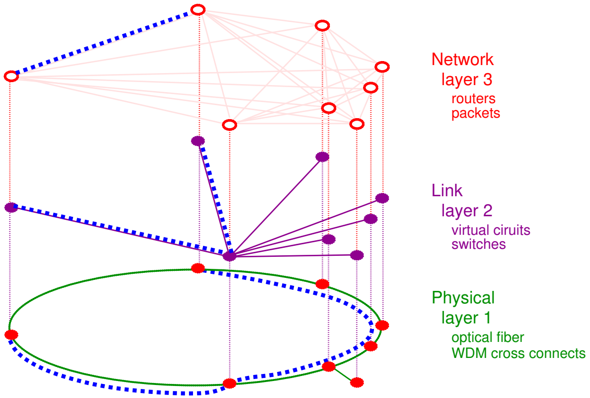
We need to map from one layer to another



That is, the logical links must be *routed* across physical links.

Mapping the logical to the physical

We need to map from one layer to another



That is, the logical links must be *routed* across physical links.

Routing

We need a method to map packet *routes* to *links*

- called a routing protocol
- several types exist
- we consider here shortest path protocols

Shortest paths

- We have a problem of working out what path packets should take from origin to destination
- Often, links in networks have lengths associated with them
  - shortest paths would get packets to their destinations fastest
  - shortest paths also use the least resources
  - shortest paths come up in lots of other applications
- We'll look an algorithm (Dijkstra's) for computing shortest paths
  - it's a greedy algorithm
  - but it guarantees to find the optimal path

Shortest path problem assumptions

For a network $G(N, L)$,

1. The length of link (i, j) is $c_{ij} \geq 0$
   - some shortest path algorithms can work with negative distances, but we then need to assume there are no negative cycles
2. There exists a path from s to all other $i \in N$.
   - if not, add a dummy link from s to i with very large cost M
3. What problem?
   - APSP = All Pairs Shortest Paths
   - SSSP = Single Source Shortest Paths $\Leftarrow$ we'll start with this

Section 2

Dijkstra's algorithm

Routing

- in essence, routing maps
  - end-to-end traffic from p to q, *i.e.*, t_{pq}
  - to end-to-end paths in P_{pq}
  - to links in E
- there are very many paths
  - can't search them all
  - have to be clever about choice of paths
- can use multiple paths
  - load-balancing — spreads load over paths

Dijkstra's algorithm

- fast method to find shortest paths is *Dijkstra's algorithm* [Dij59]
  - Edsger Dijkstra (1930-2002)
    - Dutch computer scientist
    - Turing prize winner 1972.
    - "Goto Statement Considered Harmful" paper
- find distance of all nodes from one start point
- works by finding paths in order of shortest first
  - longer paths are built up of shorter paths

Dijkstra's algorithm

Input

- graph (N, E)
- link *weights* α_e, define link distances

$$d_{ij} = \begin{cases} 0 & \text{if } i = j \\ \alpha_e & \text{where } (i,j) = e \in E \\ \infty & \text{where } (i,j) = e \notin E \end{cases}$$

- a start node, WLOG assume it is node 1

Output

- distances D_j of each node $j \in N$ from start node 1.
- a predecessor node for each node (gives path)

Dijkstra's algorithm

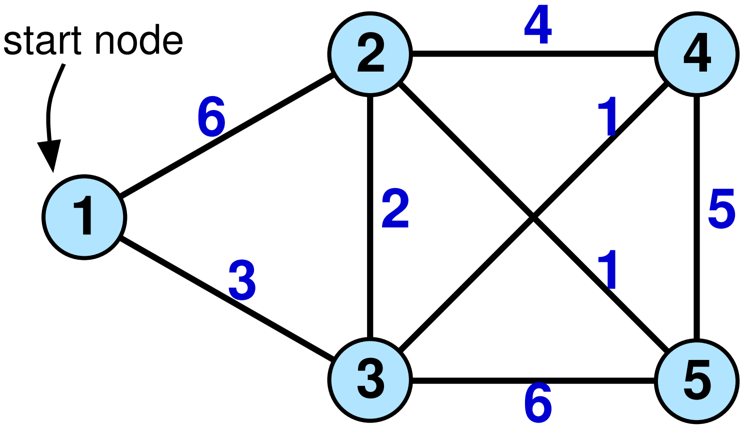
Let S be the set of *labelled* nodes.

Initialise: $S = \{1\}$,
$D_1 = 0$,
$D_j = d_{1j}, \ \forall j \notin S$, *i.e.*, $j \neq 1$.

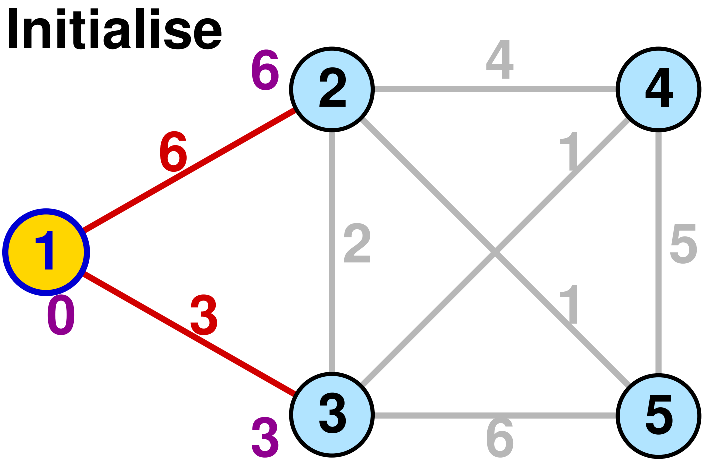
Step 1: *Find the next closest node*
Find $i \notin S$ such that $D_i = \min\{D_j : j \notin S\}$
Set $S = S \cup \{i\}$.
If $S = N$, **stop**

Step 2: *Find new distances*
For all $j \notin S$, set
$D_j = \min\{D_j, D_i + d_{ij}\}$
Goto Step 1.

Dijkstra Example



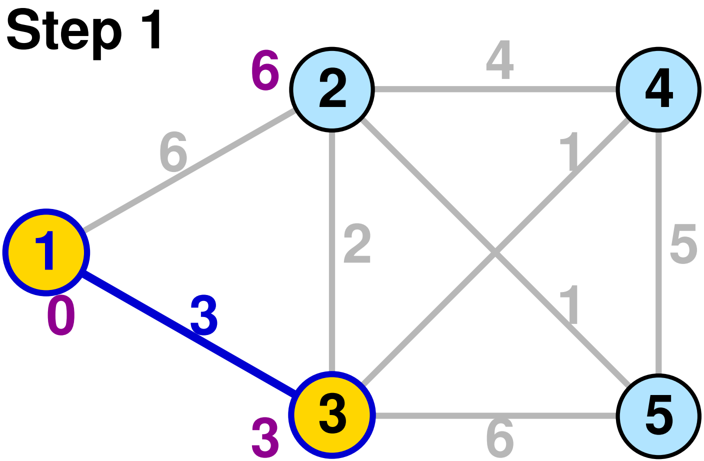
Dijkstra Example



S={1}

D=(0,6,3, ,)

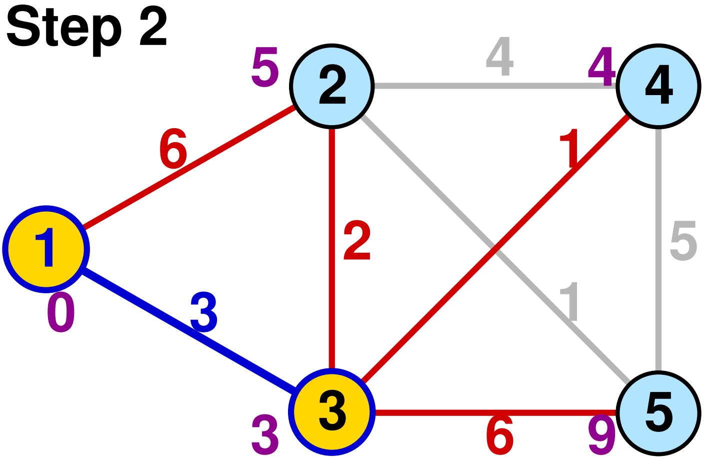
Dijkstra Example



S={1,3}

D=(0,6,3, ,)

Dijkstra Example

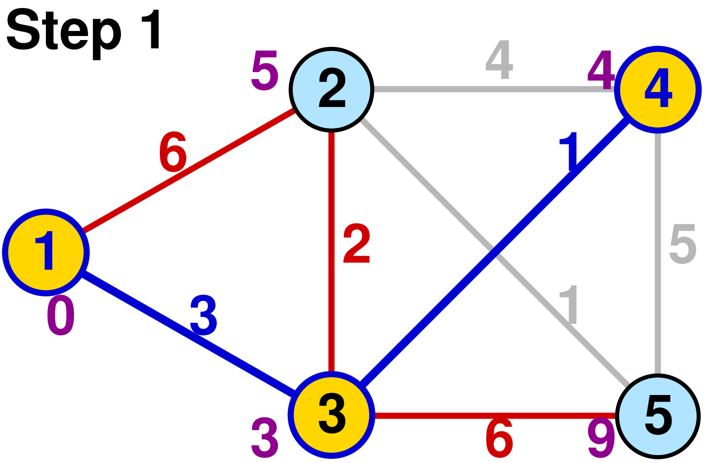


S={1,3}

D=(0,5,3,4,9)

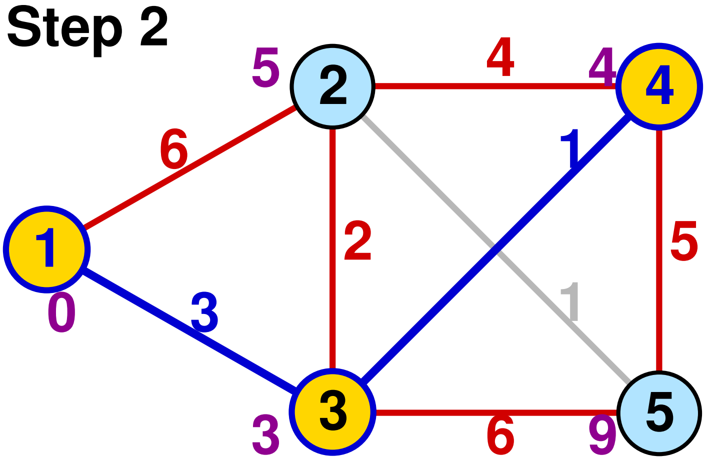
changed

Dijkstra Example



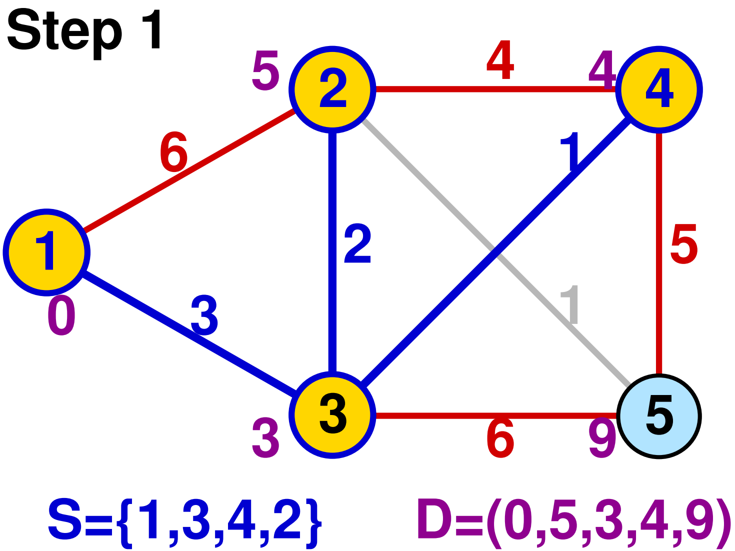
S={1,3,4} D=(0,5,3,4,9)

Dijkstra Example

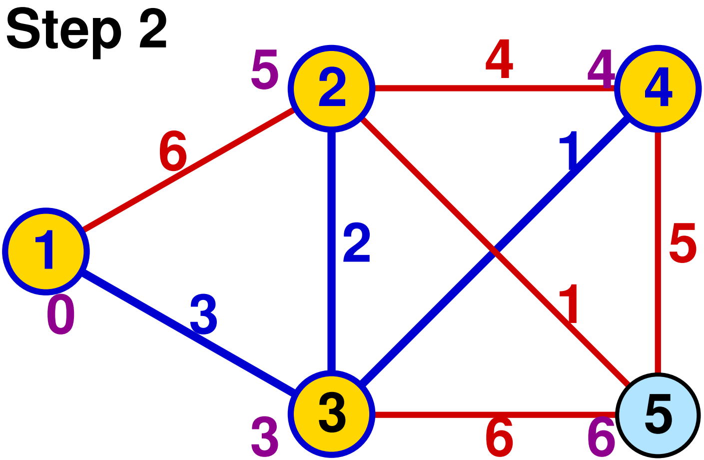


S={1,3,4} **D=(0,5,3,4,9)**

Dijkstra Example



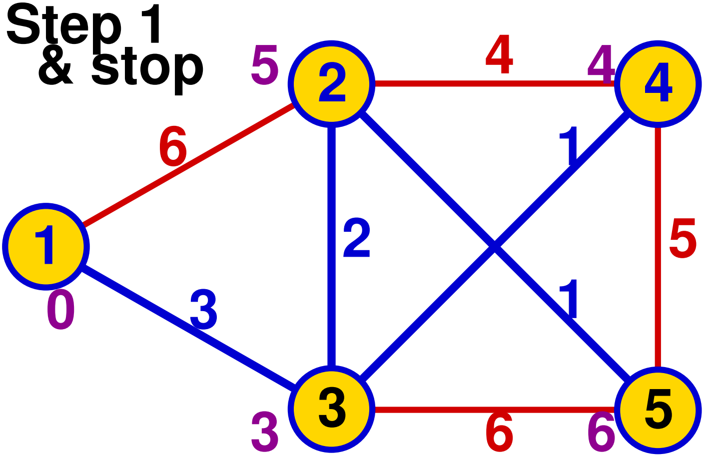
Dijkstra Example



S={1,3,4,2}   D=(0,5,3,4,6)

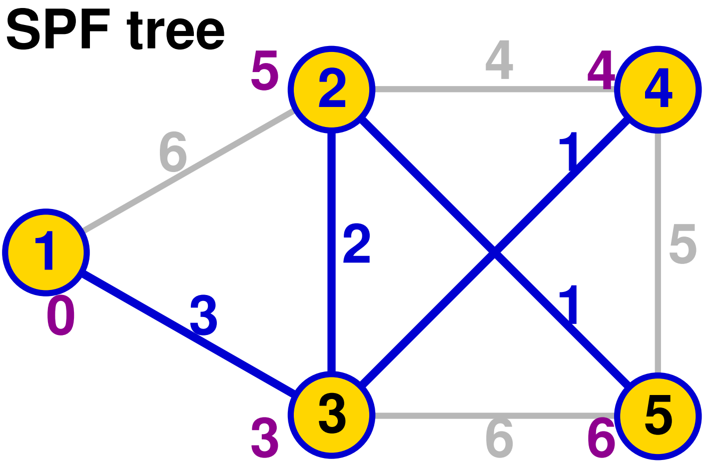
changed

Dijkstra Example



S={1,3,4,2,5} **D=(0,5,3,4,6)**

Dijkstra Result



S={1,3,4,2,5} D=(0,5,3,4,6)

Dijkstra intuition

- build a (Shortest-Path First) *SPF tree*
- let it grow
- grow by adding shortest paths onto it
- solution must look like a tree
  - to get paths, we only need to keep track of *predecessors*, *e.g.*, previous example

node	predecessor
1	-
2	3
3	1
4	3
6	2

Dijkstra issues

- Dijkstra's algorithm solves *single-source all-destinations problem*
- easily extended to a directed graph
  - can only join up in the direction of a link
- link-distances (weights) must be non-negative
  - there are other algorithms to deal with negative weights

Dijkstra complexity

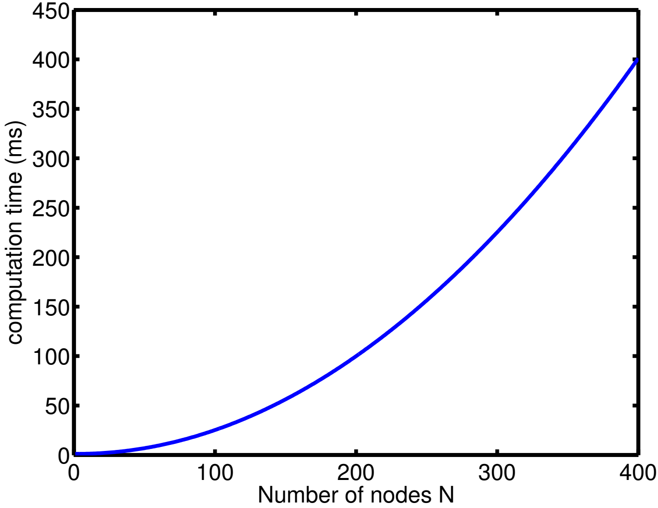
- Instance size given by number of nodes $|N|$ and edges $|E|$ in the graph
- Simple implementation complexity $O(|N|^2)$
- Cisco's implementation of Dijkstra tested in [SG01]

$$\text{comp.time} = 2.53N^2 - 12.5N + 1200 \text{ microseconds}$$

- Complexity (assuming smart data structures, *i.e.*, Fibonacci heap) is $O(|E| + |N| \log |N|)$,
  - $|E|$ = number of edges
  - $|N|$ = number of nodes
- To compute paths for all pairs, we can perform Dijkstra for each starting point, with complexity $O(|N||E| + |N|^2 \log |N|)$,

Dijkstra complexity

Empirical Cisco 7500 and 12000 (GSR) computation times for Dijkstra [SG01]



$2.53N^2 - 12.5N + 1200\mu s$

Sketch of proof of Dijkstra

Theorem

Dijkstra's algorithm solves the single-source shortest-paths problem in networks that have nonnegative weights.

Proof: Call the source node s the root, then we need to show that the paths from s to each node x corresponds to a shortest path in the graph from s to x. Note that this set of paths forms a tree out of a subset of edges of the graph.

The proof uses induction. We assume that the subtree formed at some point along the algorithm has the property (of shortest paths). Clearly the starting point satisfies this assumption, so we need only prove that adding a new node x adds a shortest path to that node. All other paths to x must begin with a path from the current subtree (because these are shortest paths) followed by an edge to a node not on the tree. By construction, all such paths are longer than the one from s to x that is produced by Dijkstra.

Takeaways

- Shortest paths
  - common optimisation problem
  - Dijkstra is a good solution
  - but not the only one: there are better approaches
    - some deal with more general cases
    - some are distributed
    - some are slightly faster
- Greed is good
  - greedy algorithms can be optimal
  - there are lots of similar algorithms
    - *e.g.*, Prim's algorithm for finding minimum spanning trees

Further reading I

E.W. Dijkstra, *A note in two problems in connexion with graphs*, Numerische Mathematik **1** (1959), 269–271.

Aman Shaikh and Albert Greenberg, *Experience in black-box OSPF measurement*, Proc. ACM SIGCOMM Internet Measurement Workshop, 2001, pp. 113–125.