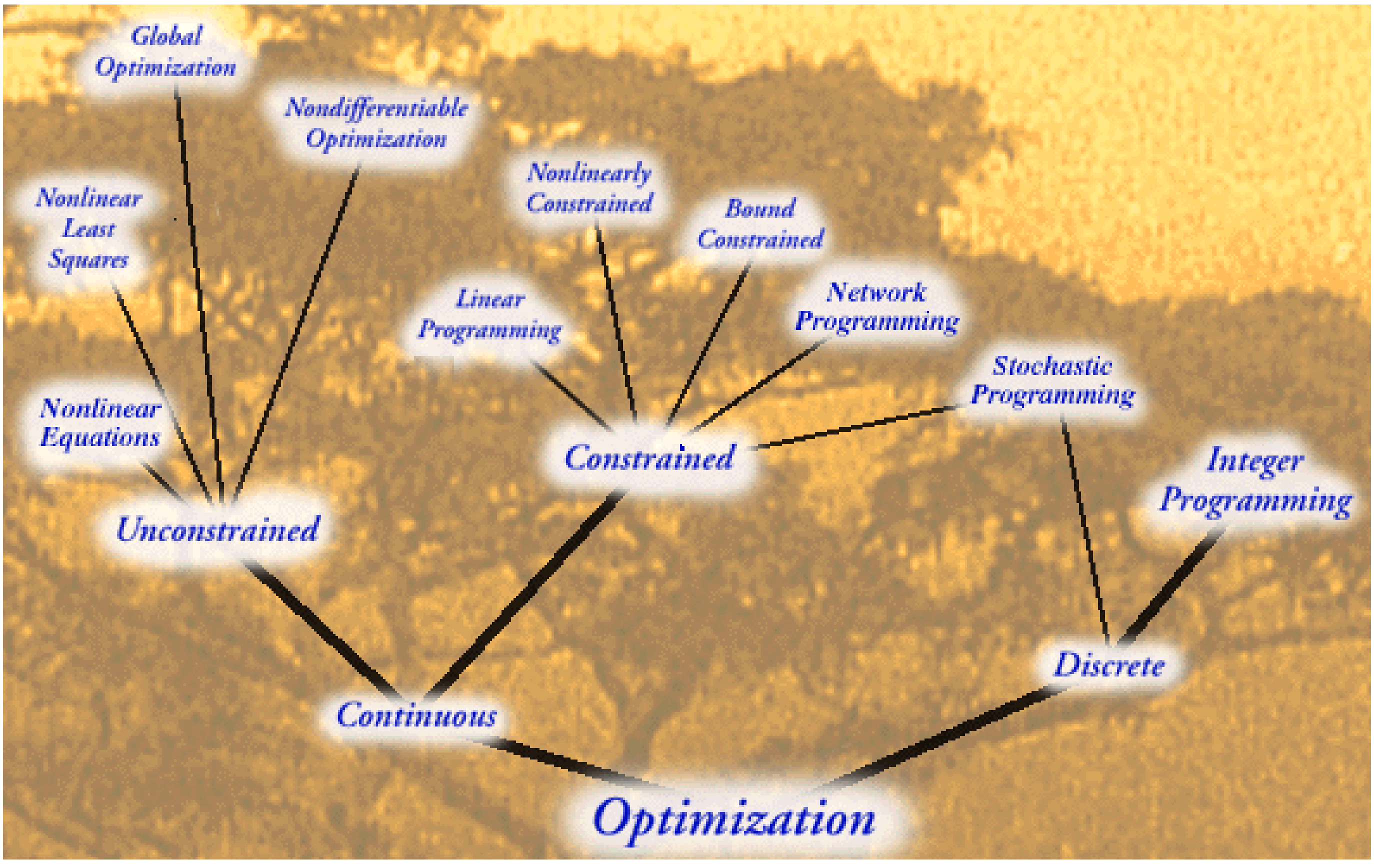
Lecture 3

C25 Optimization Hilary 2013 A. Zisserman

Dynamic Programming on graphs

- Terminology: chains, stars, trees, loops
- Application
- Message passing
- Shortest path - Dijkstra's algorithm

The Optimization Tree



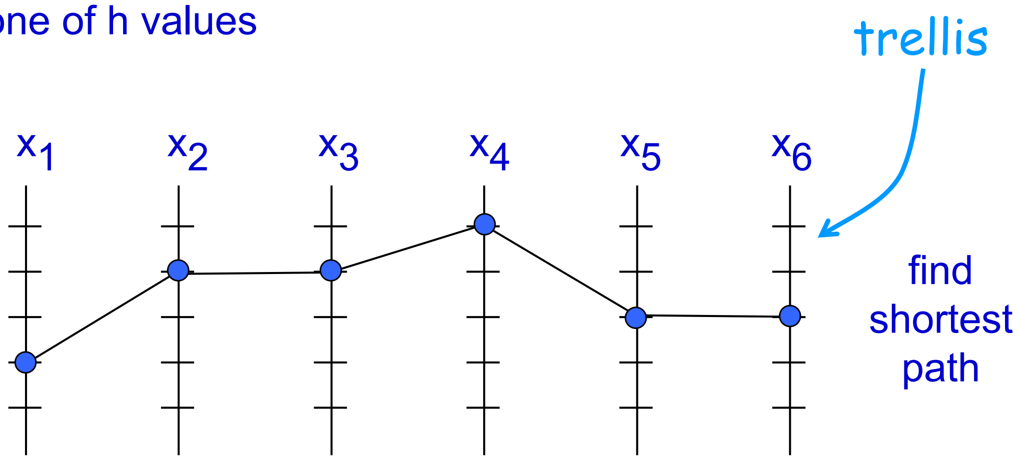
Consider a cost function $f(\mathbf{x}) : \mathbb{R}^n \to \mathbb{R}$ of the form

RECAP

$$f(\mathbf{x}) = \sum_{i=1}^{n} m_i(x_i) + \sum_{i=2}^{n} \phi_i(x_{i-1}, x_i)$$

where x_i can take one of h values

e.g. h=5, n=6



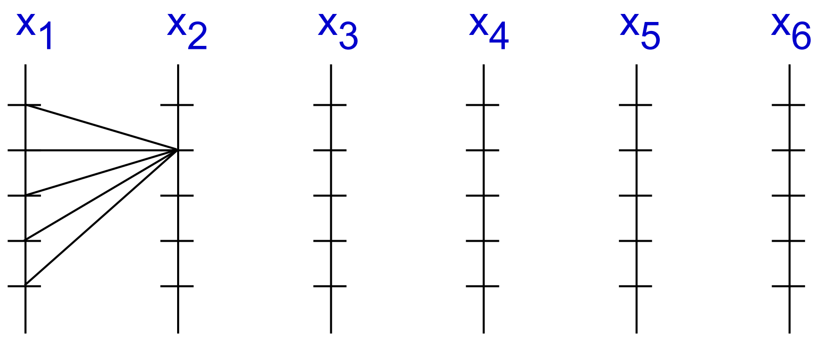
$$f(\mathbf{x}) = \begin{cases} m_1(x_1) + m_2(x_2) + m_3(x_3) + m_4(x_4) + m_5(x_5) + m_6(x_6) \\ \phi(x_1, x_2) + \phi(x_2, x_3) + \phi(x_3, x_4) + \phi(x_4, x_5) + \phi(x_5, x_6) \end{cases}$$

Complexity of minimization:

- exhaustive search $O(h^n)$
- dynamic programming $O(nh^2)$

RECAP

$$f(\mathbf{x}) = \sum_{i=1}^{n} m_i(x_i) + \sum_{i=2}^{n} \phi(x_{i-1}, x_i)$$



Key idea: the optimization can be broken down into n sub-optimizations

Step 1: For each value of x_2 determine the best value of x_1

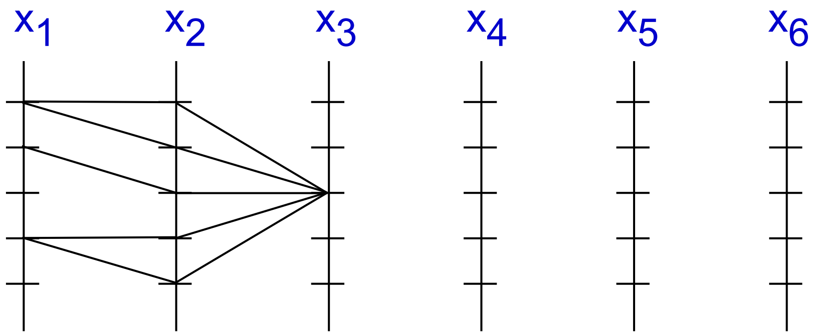
- Compute

$$\begin{aligned} S_2(x_2) &= \min_{x_1}\{m_2(x_2) + m_1(x_1) + \phi(x_1, x_2)\} \\ &= m_2(x_2) + \min_{x_1}\{m_1(x_1) + \phi(x_1, x_2)\} \end{aligned}$$

- Record the value of x_1 for which $S_2(x_2)$ is a minimum

To compute this minimum for all x_2 involves $O(h^2)$ operations

RECAP



Step 2: For each value of x_3 determine the best value of x_2 **and** x_1

- Compute

$$S_3(x_3) \;=\; m_3(x_3) + \min_{x_2}\{S_2(x_2) + \phi(x_2, x_3)\}$$

- Record the value of x_2 for which $S_3(x_3)$ is a minimum

Again, to compute this minimum for all x_3 involves $O(h^2)$ operations
Note $S_k(x_k)$ encodes the lowest cost partial sum for all nodes up to k which have the value x_k at node k, i.e.

$$S_k(x_k) = \min_{x_1, x_2, \ldots, x_{k-1}} \sum_{i=1}^{k} m_i(x_i) + \sum_{i=2}^{k} \phi(x_{i-1}, x_i)$$

RECAP

Viterbi Algorithm

- Initialize $S_1(x_1) = m_1(x_1)$
- For $k = 2 : n$

$$\begin{aligned} S_k(x_k) &= m_k(x_k) + \min_{x_{k-1}}\{S_{k-1}(x_{k-1}) + \phi(x_{k-1}, x_k)\} \\ b_k(x_k) &= \arg\min_{x_{k-1}}\{S_{k-1}(x_{k-1}) + \phi(x_{k-1}, x_k)\} \end{aligned}$$

- Terminate

$$x_n^* = \arg\min_{x_n} S_n(x_n)$$

- Backtrack

$$x_{i-1} = b_i(x_i)$$

Complexity O(nh^2)

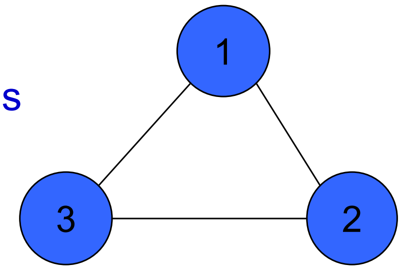
Dynamic Programming on graphs

- Graph (V, E)
- Vertices v_i for $i = 1, \ldots, n$
- Edges e_{ij} connect v_i to other vertices v_j

$$f(\mathbf{x}) = \sum_{v_i \in V} m_i(v_i) + \sum_{e_{ij} \in E} \phi(v_i, v_j)$$

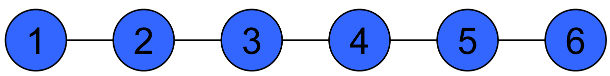
e.g.

- 3 vertices, each can take one of h values
- 3 edges

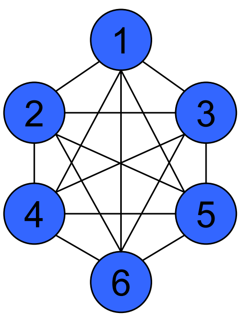


Dynamic Programming on graphs

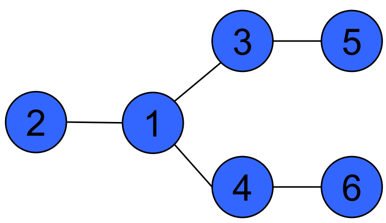
So far have considered chains



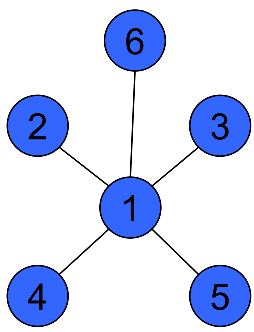
Can dynamic programming be applied to these configurations?
(i.e. to reduce the optimization complexity from $O(h^n)$ to $O(nh^2)$)



Fully connected

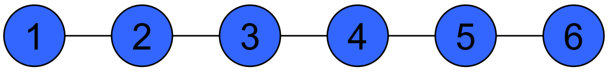


Tree structure



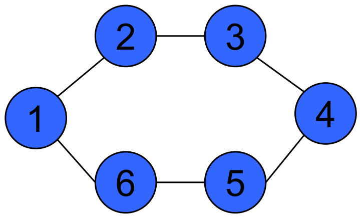
Star structure

Terminlogy



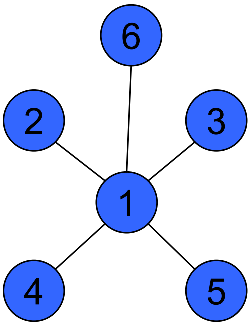
simple (or open) chain

Graphs with loops

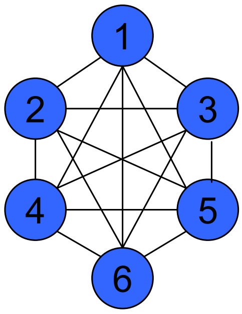


closed chain

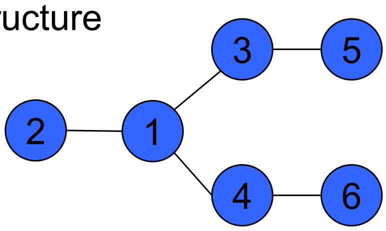
Star structure

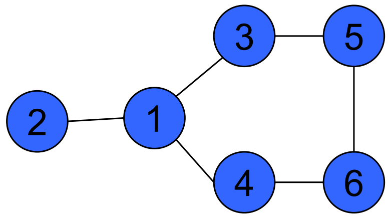


Fully connected

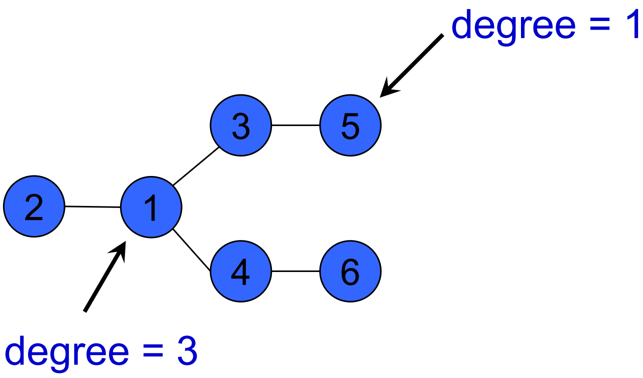


Tree structure





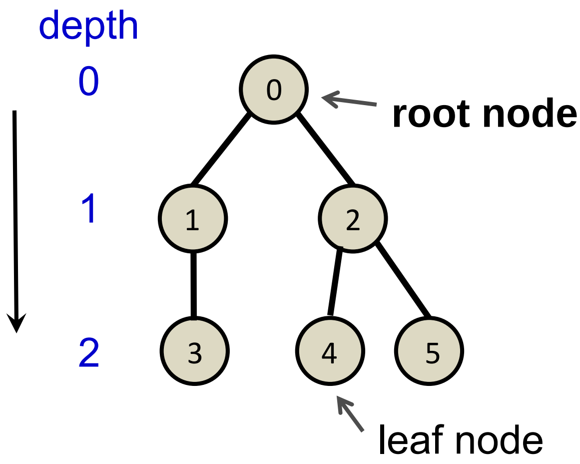
Terminlogy



degree (or valency) of a vertex

- is the number of edges incident to the vertex

Terminlogy for trees



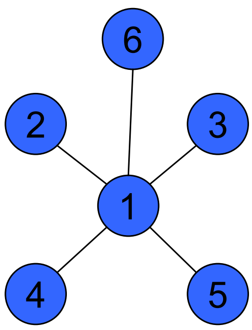
depth of node

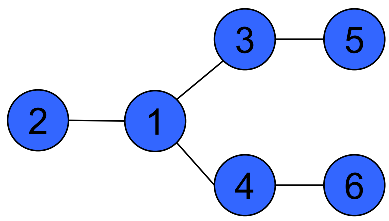
- number of edges between node and root

children of node i

- are neighbouring vertices of depth $d_i + 1$

Dynamic programming on stars and trees

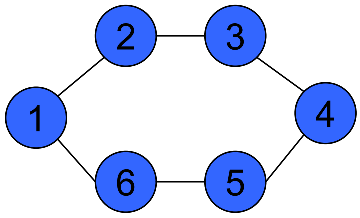




- for a value of the central node determine the best value of each of the other nodes in turn O(nh)
- repeat for all values of the central node O(h)
- final complexity $O(nh^2)$

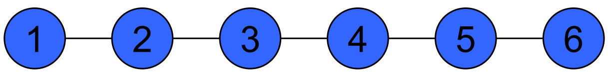
- order the nodes by their depth from the root, where d is max depth
- start with nodes at depth d-1, and compute best value for all (child) nodes at depth d, $O(h^2)$
- decrease depth and repeat, O(n)
- final complexity $O(nh^2)$

Dynamic programming on graphs with loops

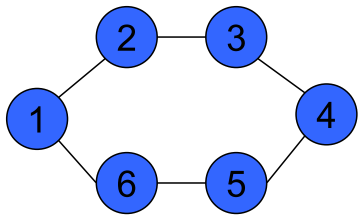


- select one node and choose its value
- for the other nodes, the graph is then equivalent to an open chain and can be optimized with $O(nh^2)$ complexity
- repeat for all values of the selected node and choose lowest overall cost from these
- final complexity $O(nh^3)$

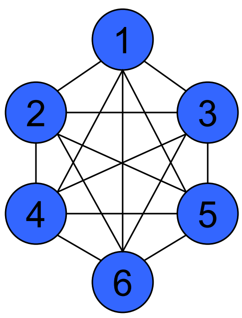
Summary of different graph structures



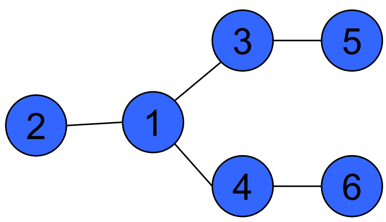
Open chain $O(nh^2)$



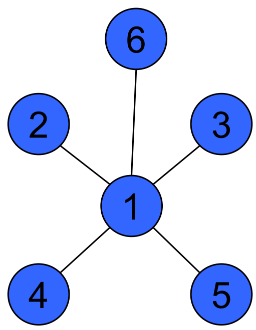
Closed chain $O(nh^3)$



Fully connected



Tree structure

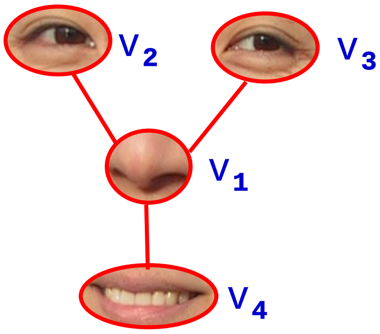


Star structure

$O(h^n)$ $O(nh^2)$ $O(nh^2)$

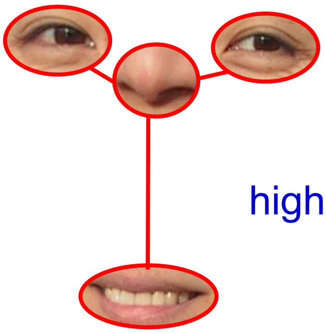
Application: facial feature detection in images

Model



- Parts V= {v_1, … v_n}
- Connected by springs in star configuration to nose
- Quadratic cost for spring

$$
\begin{aligned}
f(\mathbf{x}) &= \sum_{v_i \in V} m_i(v_i) + \sum_{e_{ij} \in E} \phi(v_i, v_j) \\
&= \sum_{v_i \in V} m_i(v_i) + \sum_{j} \phi(v_1, v_j)
\end{aligned}
$$



high spring cost

1 - NCC with appearance template

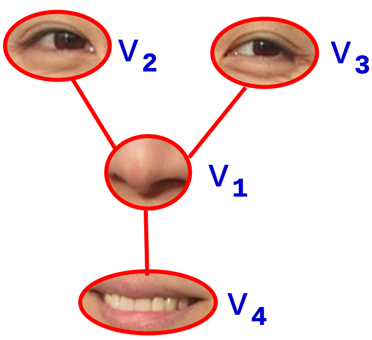
Spring extension from v_1 to v_j

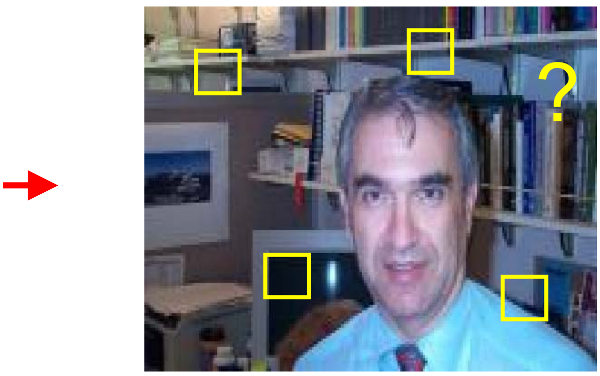
Fitting the model to an image

Find the configuration with the lowest energy

$$
E(\mathbf{x}) = \sum_{v_i \in V} m_i(v_i) + \sum_{j} \phi_{1,j}(v_1, v_j)
$$

Model





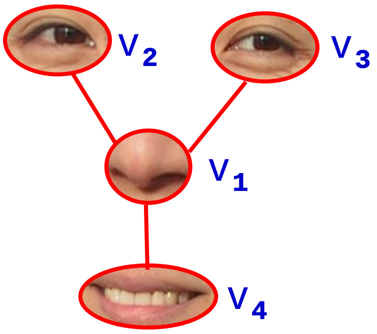
h = 10^6 n = 4

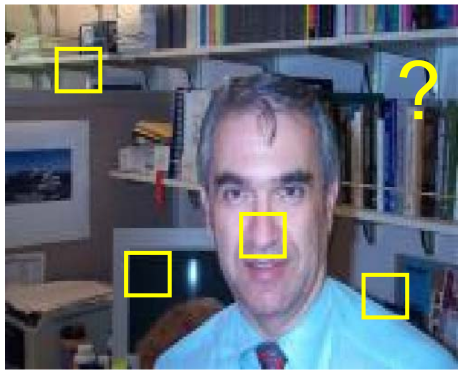
Fitting the model to an image

Find the configuration with the lowest energy

$$E(\mathbf{x}) = \sum_{v_i \in V} m_i(v_i) + \sum_j \phi_{1,j}(v_1, v_j)$$

Model





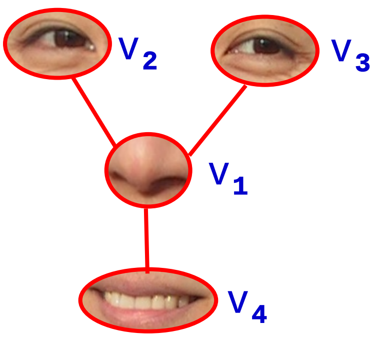
h = 10^6 n = 4

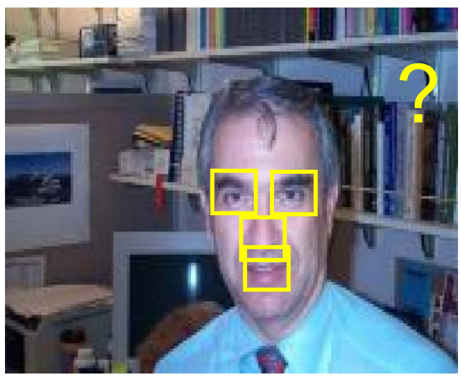
Fitting the model to an image

Find the configuration with the lowest energy

$$E(\mathbf{x}) = \sum_{v_i \in V} m_i(v_i) + \sum_j \phi_{1,j}(v_1, v_j)$$

Model





h = 10^6 n = 4

$$O(h^n) \quad vs \quad O(nh^2)$$

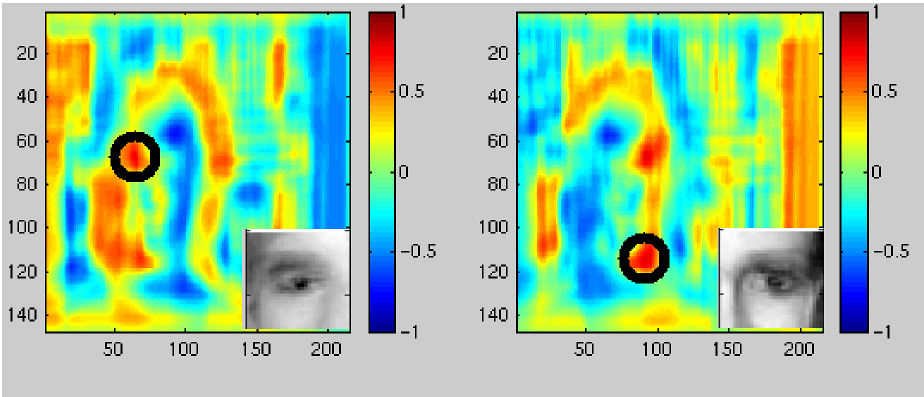
Appearance templates and springs

$$f(\mathbf{x}) = \sum_{v_i \in V} m_i(v_i) + \sum_j \phi(v_1, v_j)$$

$$\mathbf{x} = (x_1, y_1, \ldots, x_4, y_4)^\top$$

Each ($\mathbf{x}_i$, y_i) ranges over h (x,y) positions in the image

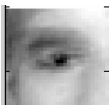
$$NCC = 1 - m_i$$

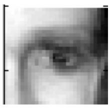


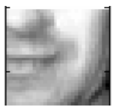
Requires pair wise terms for correct detection

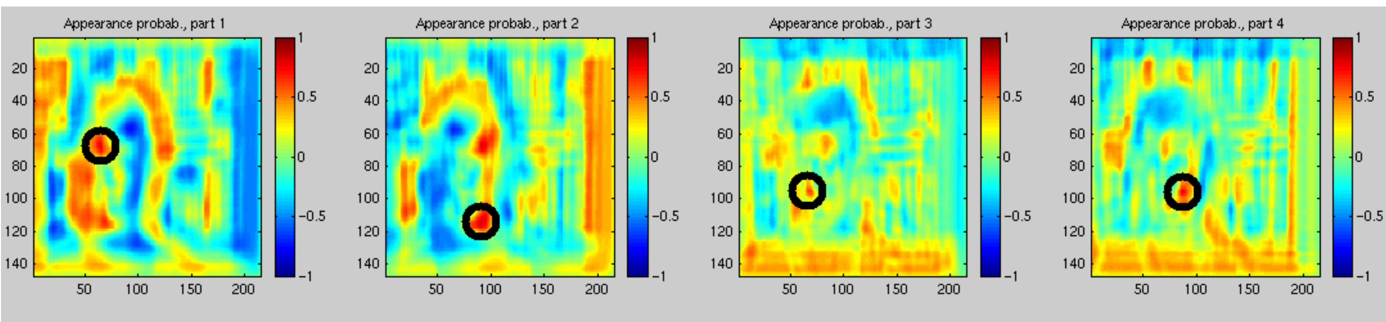
Example

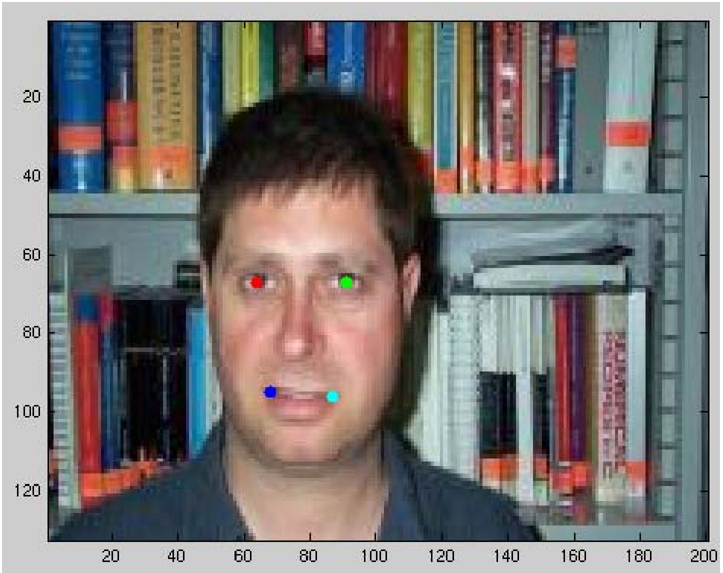
$1 - m_i$







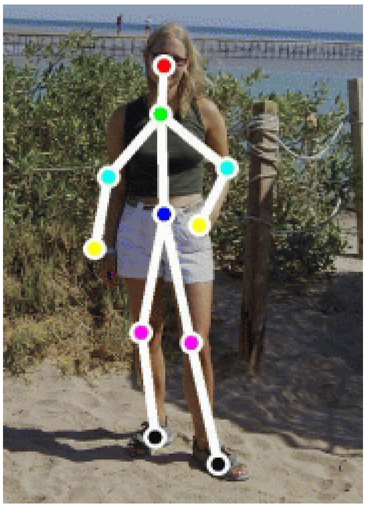


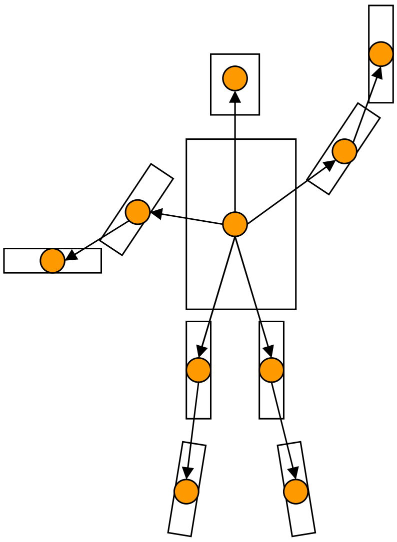


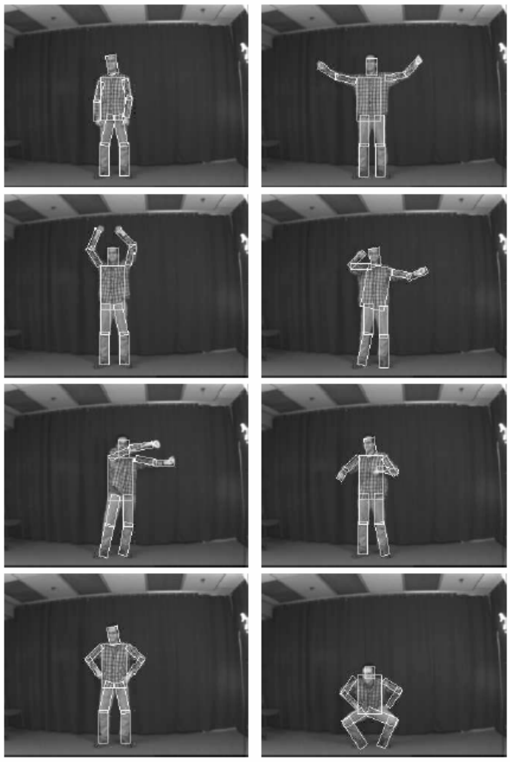
Example

Application of trees – computer vision example

Detect person layout

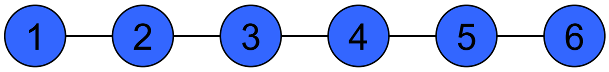






Message Passing

Example: counting people in a queue



How to count the number of people in a queue?

Method 1:

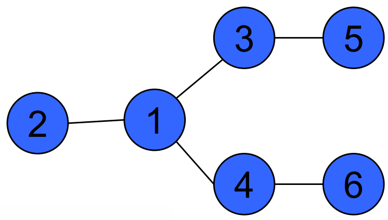
- ask them all to shout out their names and count these

Method 2:

- person at either end sends “1” to their neighbour
- if a person receives a number from their neighbour, add one, and pass total to neighbour on other side
- when anyone receives a message from both sides, they can compute the length of the queue by adding both messages and one (for themselves)

Counting vertices on a tree

How would the algorithm have to be modified to count vertices on a tree?



1. Count your number of neighbours, N.
2. Keep count of the number of messages you have received from your neighbours, m, and of the values v_1, v_2, $\ldots, v_N$ of each of those messages. Let V be the running total of the messages you have received.
3. If the number of messages you have received, m, is equal to $N-1$, then identify the neighbour who has not sent you a message and tell them the number $V+1$.
4. If the number of messages you have received is equal to N, then:

   (a) the number $V+1$ is the required total.

   (b) for each neighbour n {

   say to neighbour n the number $V+1-v_n$.

   }

From David Mackay’s book

RECAP

Viterbi Algorithm

- Initialize $S_1(x_1) = m_1(x_1)$
- For $k = 2 : n$

$$
\begin{aligned}
S_k(x_k) &= m_k(x_k) + \min_{x_{k-1}}\{S_{k-1}(x_{k-1}) + \phi(x_{k-1}, x_k)\} \\
b_k(x_k) &= \arg\min_{x_{k-1}}\{S_{k-1}(x_{k-1}) + \phi(x_{k-1}, x_k)\}
\end{aligned}
$$

- Terminate

$$x_n^* = \arg\min_{x_n} S_n(x_n)$$

- Backtrack

$$x_{i-1} = b_i(x_i)$$

Complexity $O(nh^2)$

Message passing for Dynamic Programming

- nodes send messages to their neighbours
- messages are vectors of dimension h
- notation: $m^t{}_{p\to q}$ is message sent from p to q at time t

$m^t_{p\to q}(v_i)$, i.e. the ith component of the vector, is low if vertex p "believes" that i is a good (low cost) solution for vertex q

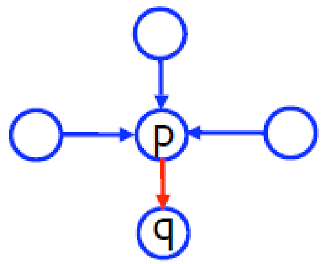
- also known as Belief Propagation

Non Examinable

Viterbi Algorithm using Message Passing

$$\min_{\mathbf{x}} f(\mathbf{x}) = \sum_{v_p \in V} \psi_p(v_p) + \sum_{e_{pq} \in E} \phi(v_p, v_q)$$

Algorithm



- Initialize $m^t_{p\to q}(v_q) = \mathbf{0}$
- Repeat for each vertex

$$m^t_{p\to q}(v_q) = \min_{v_p}\{\psi_p(v_p) + \phi(v_p, v_q) + \sum_{s\in\mathcal{N}(p)\backslash q} m^{t-1}_{s\to p}(v_p)\}$$

Non Examinable

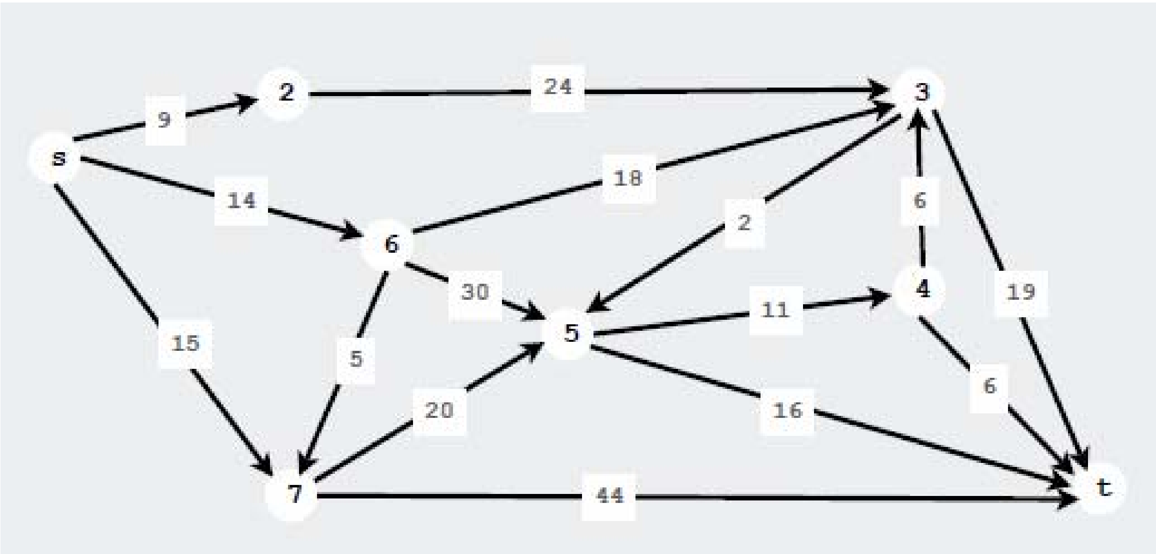
Shortest Path Algorithms

The Shortest Path Problem

Given: a weighted directed graph, with a single source s

Goal: Find the shortest path from s to t

Length of path = Σ Length of arcs



Application

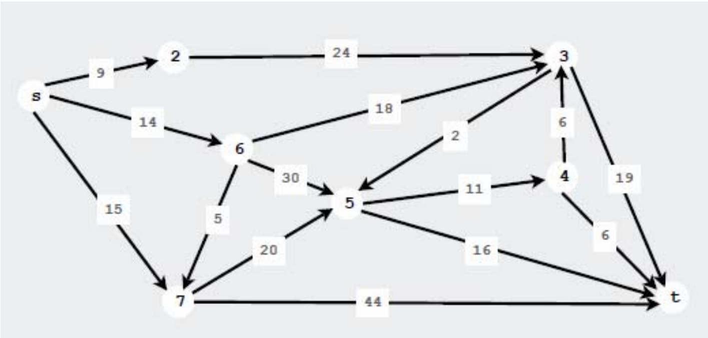
Minimize number of stops (lengths = 1)

Minimize amount of time (positive lengths)

Dijkstra's Algorithm (1959)

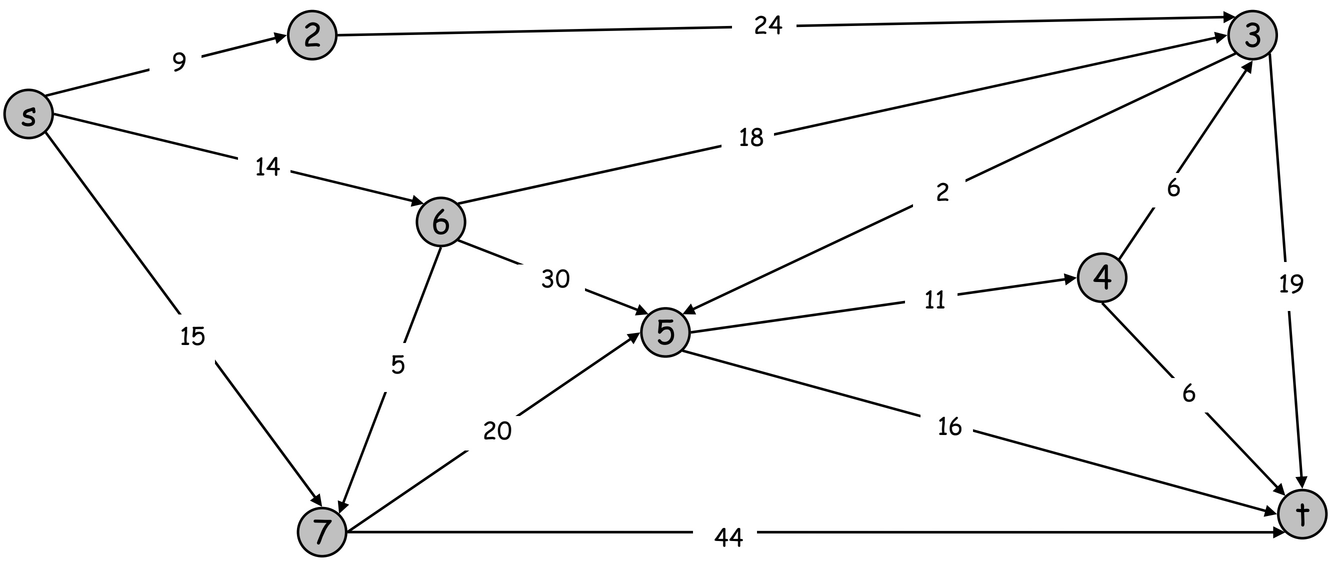
Given: a directed graph with non-negative edge costs,
Find: the shortest path from s to every other vertex

- pick the unvisited vertex with the lowest distance to s
- calculate the distance through it to each unvisited neighbour, and update the neighbour's distance if smaller
- mark vertex as visited once all neighbours explored



Example: Dijkstra's Shortest Path Algorithm

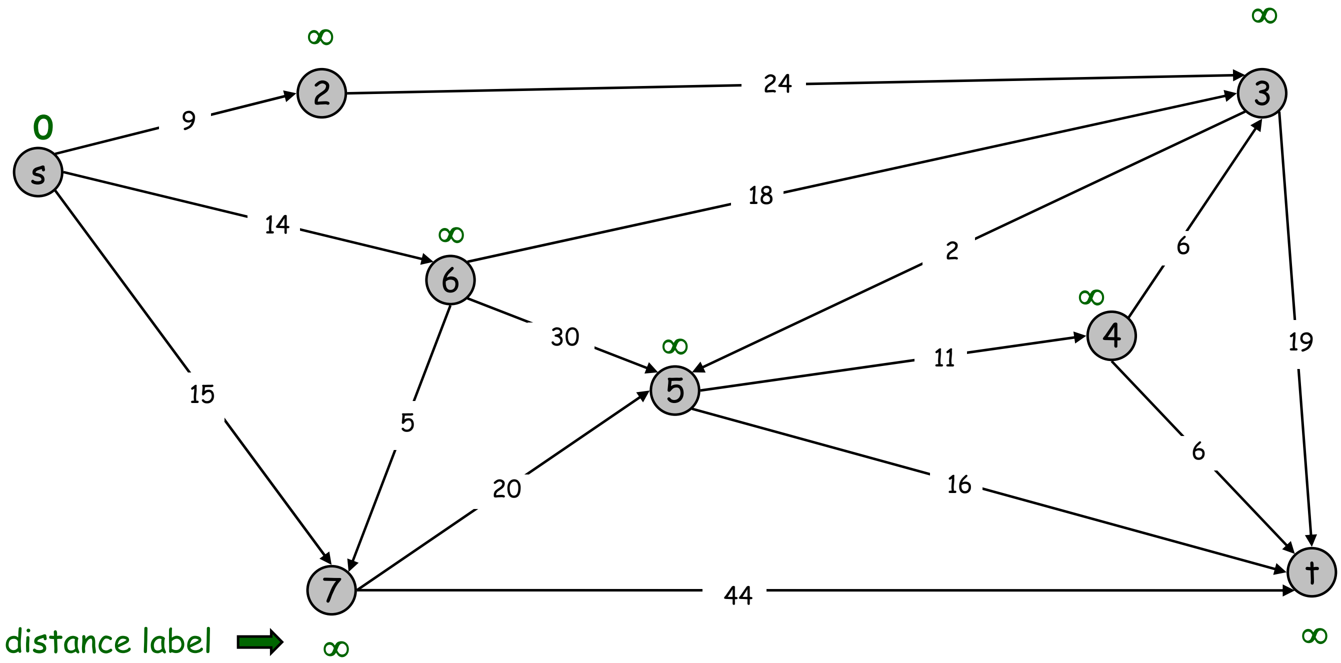
Find shortest path from s to t.



Slide: Robert Sedgewick and Kevin Wayne

Dijkstra's Shortest Path Algorithm

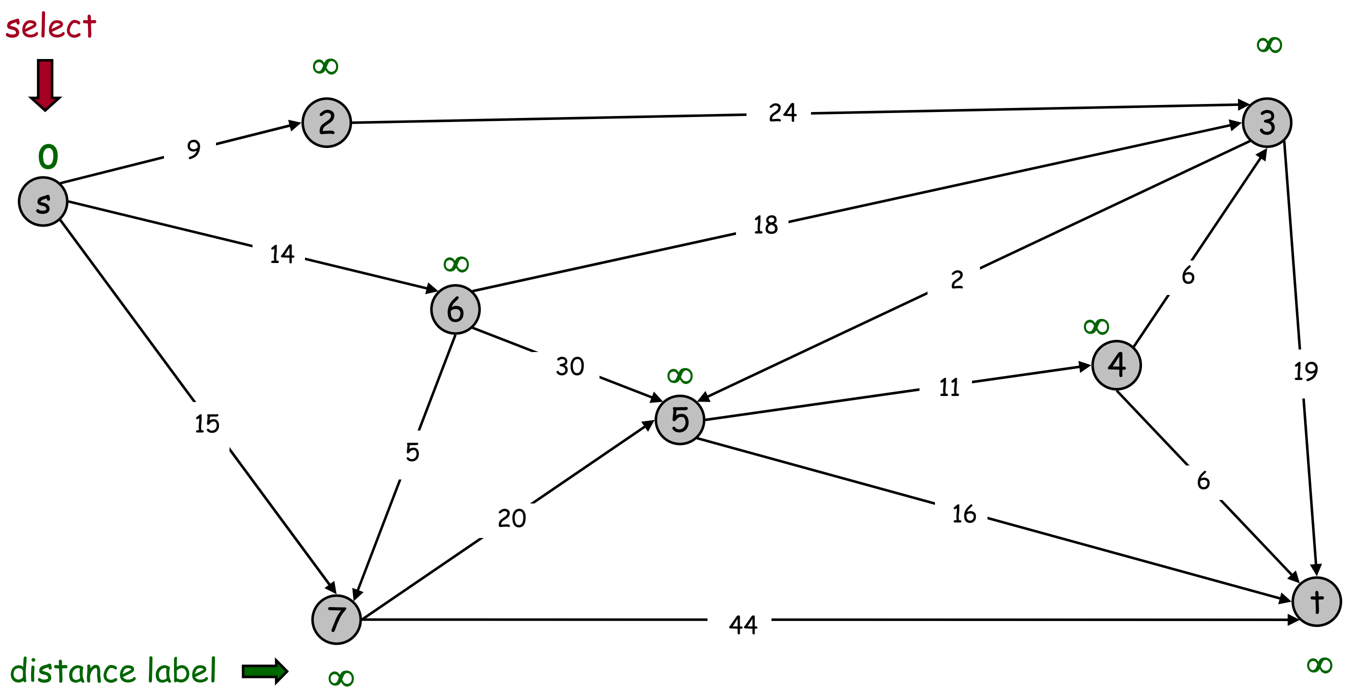
S = { }
PQ = { s, 2, 3, 4, 5, 6, 7, t }



Slide: Robert Sedgewick and Kevin Wayne

Dijkstra's Shortest Path Algorithm

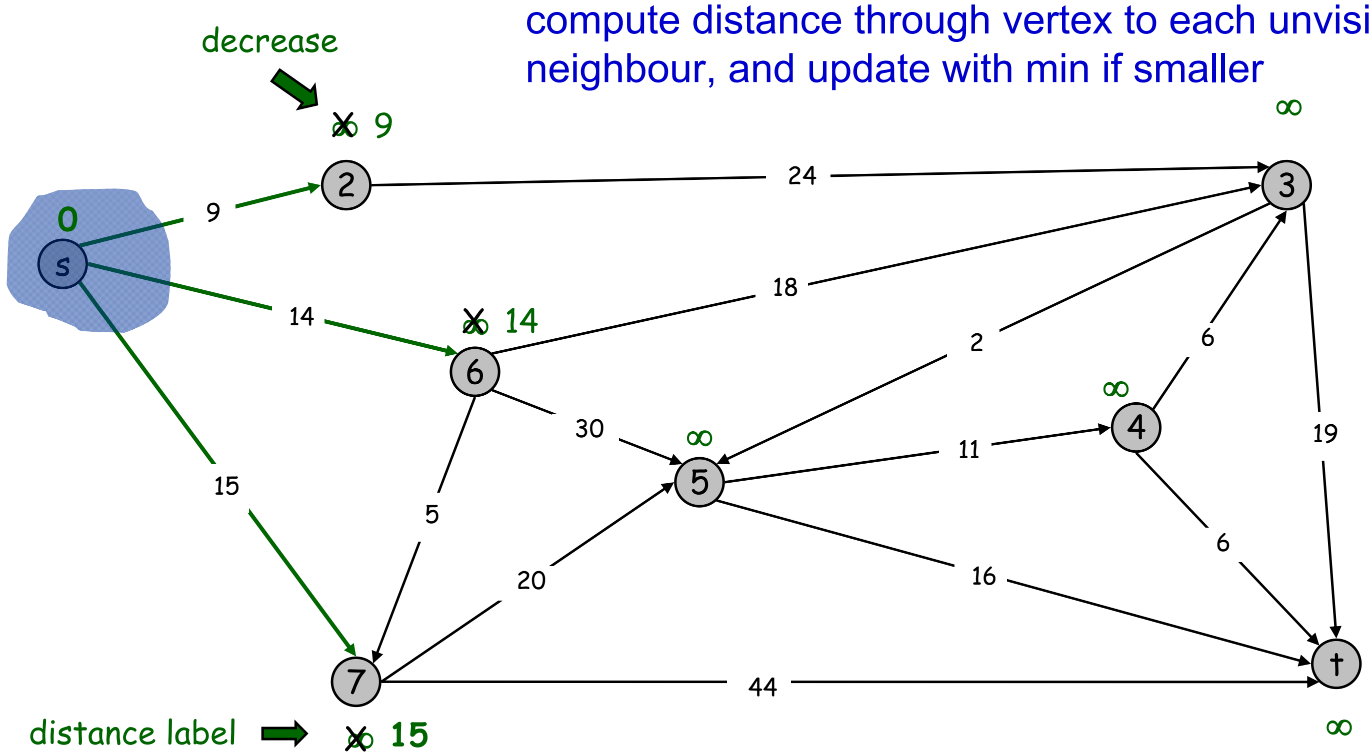
S = { }
PQ = { s, 2, 3, 4, 5, 6, 7, t }



Dijkstra's Shortest Path Algorithm

S = { s }
PQ = { 2, 3, 4, 5, 6, 7, t }

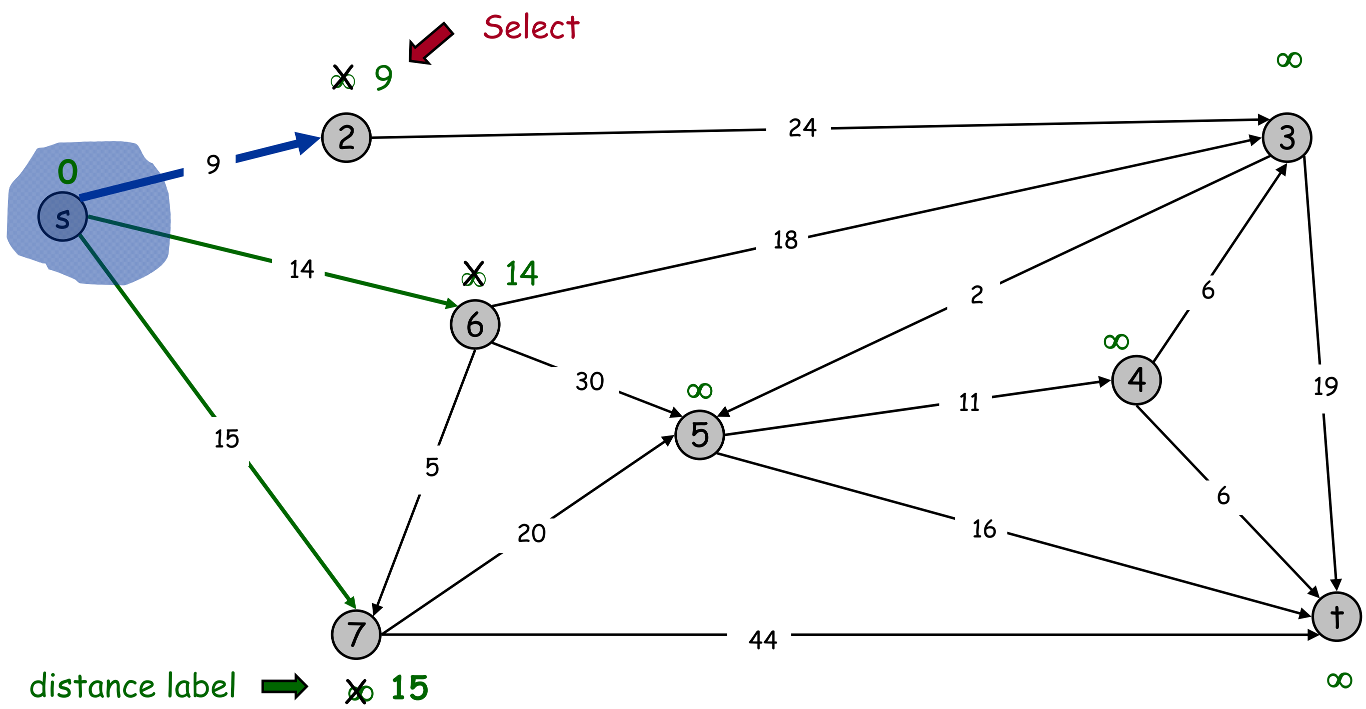
compute distance through vertex to each unvisited neighbour, and update with min if smaller



Dijkstra's Shortest Path Algorithm

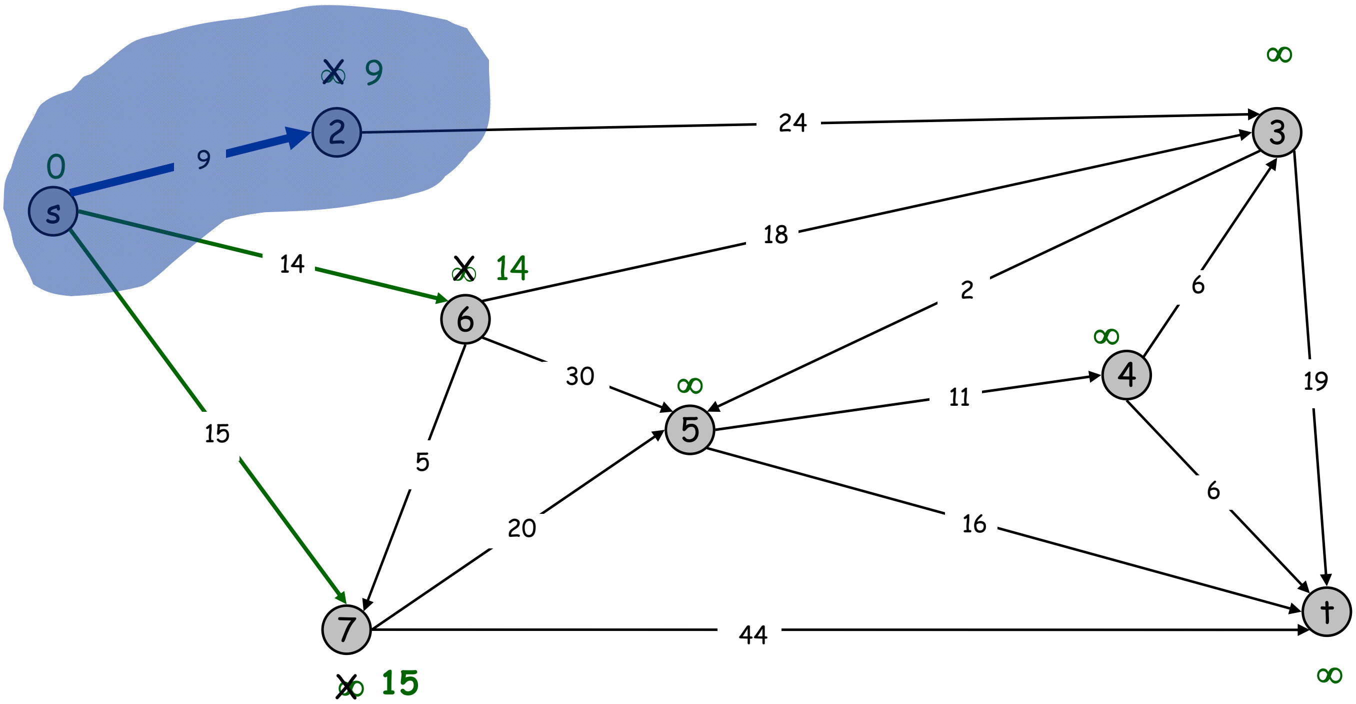
S = { s }
PQ = { 2, 3, 4, 5, 6, 7, t }

pick the unvisited vertex with the lowest distance to s



Dijkstra's Shortest Path Algorithm

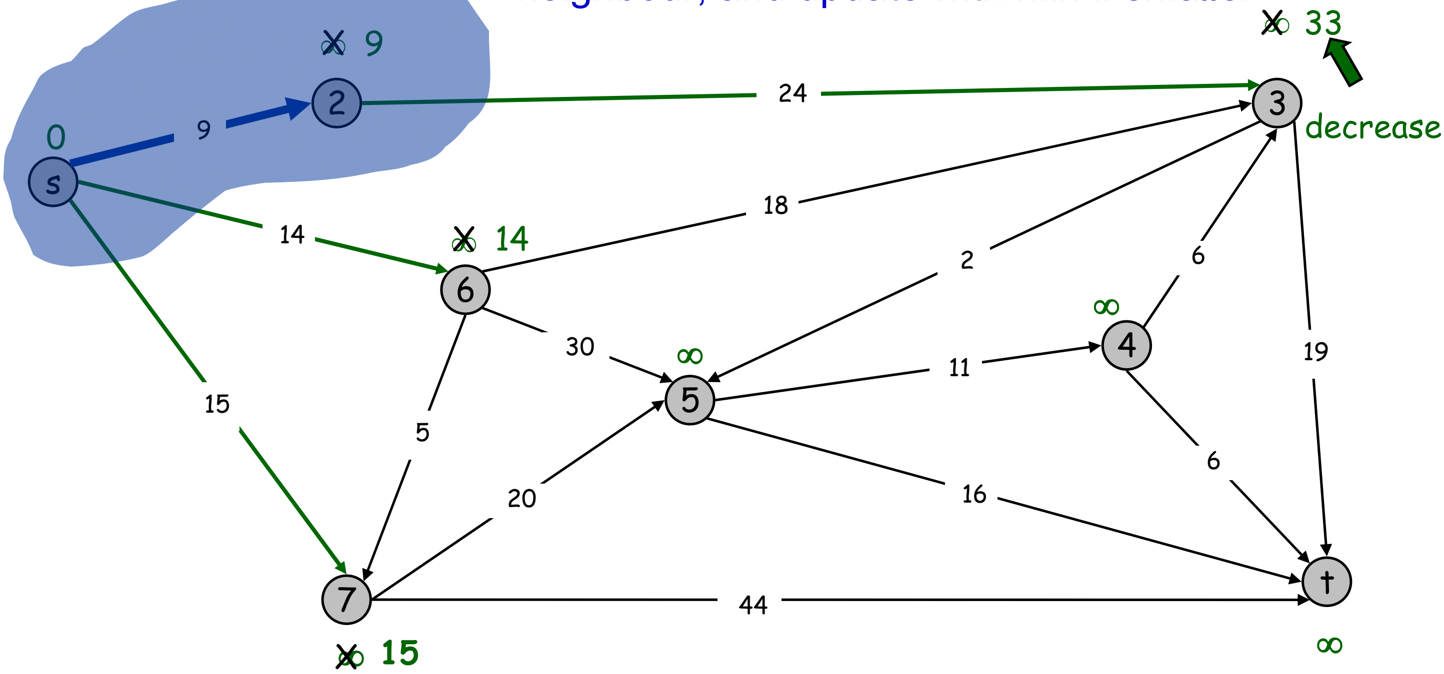
S = { s, 2 }
PQ = { 3, 4, 5, 6, 7, t }



Dijkstra's Shortest Path Algorithm

S = { s, 2 }
PQ = { 3, 4, 5, 6, 7, t }

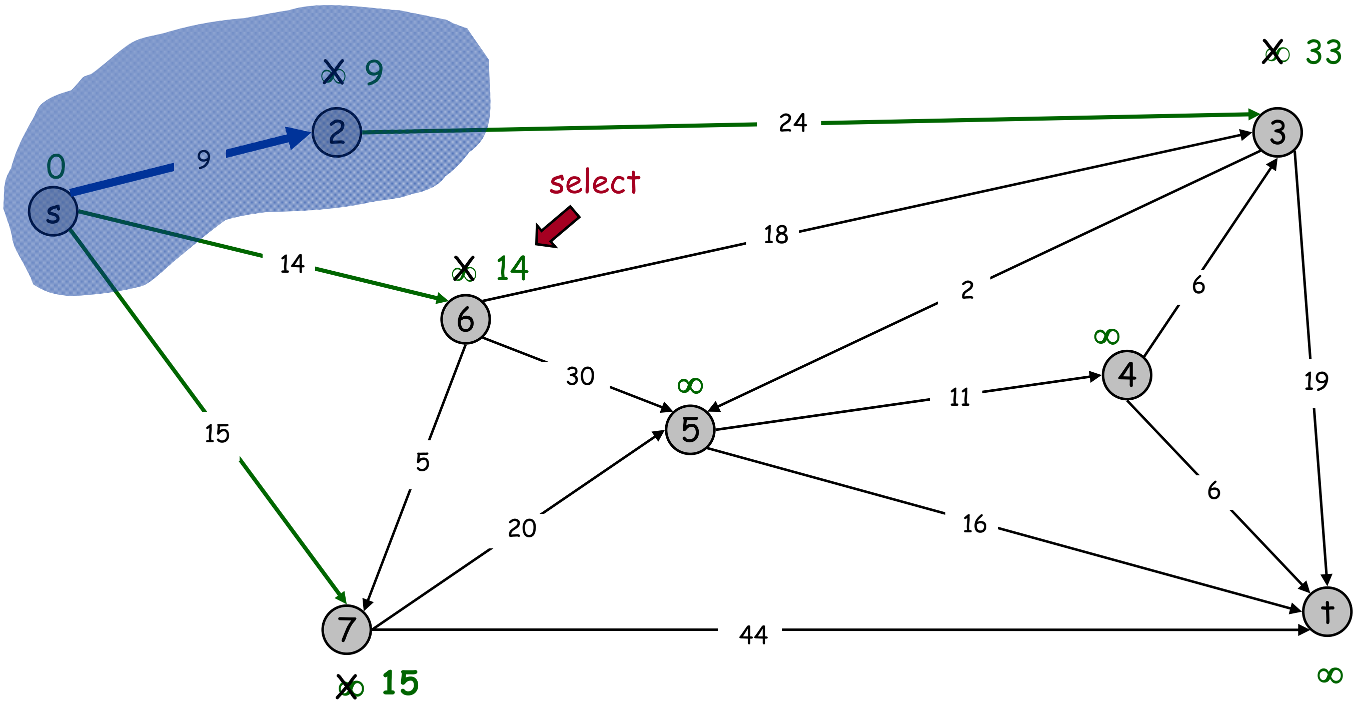
compute distance through vertex to each unvisited neighbour, and update with min if smaller



Dijkstra's Shortest Path Algorithm

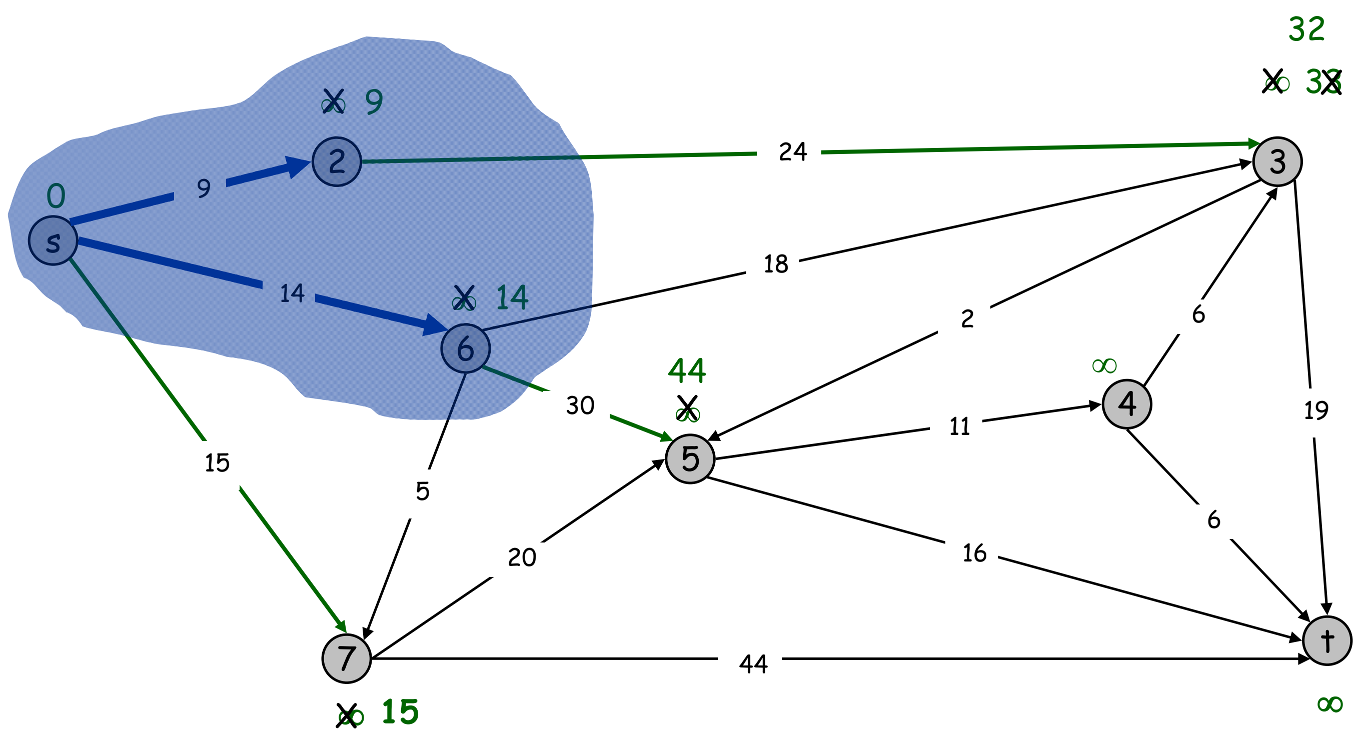
S = { s, 2 }
PQ = { 3, 4, 5, 6, 7, t }

pick the unvisited vertex with the lowest distance to s



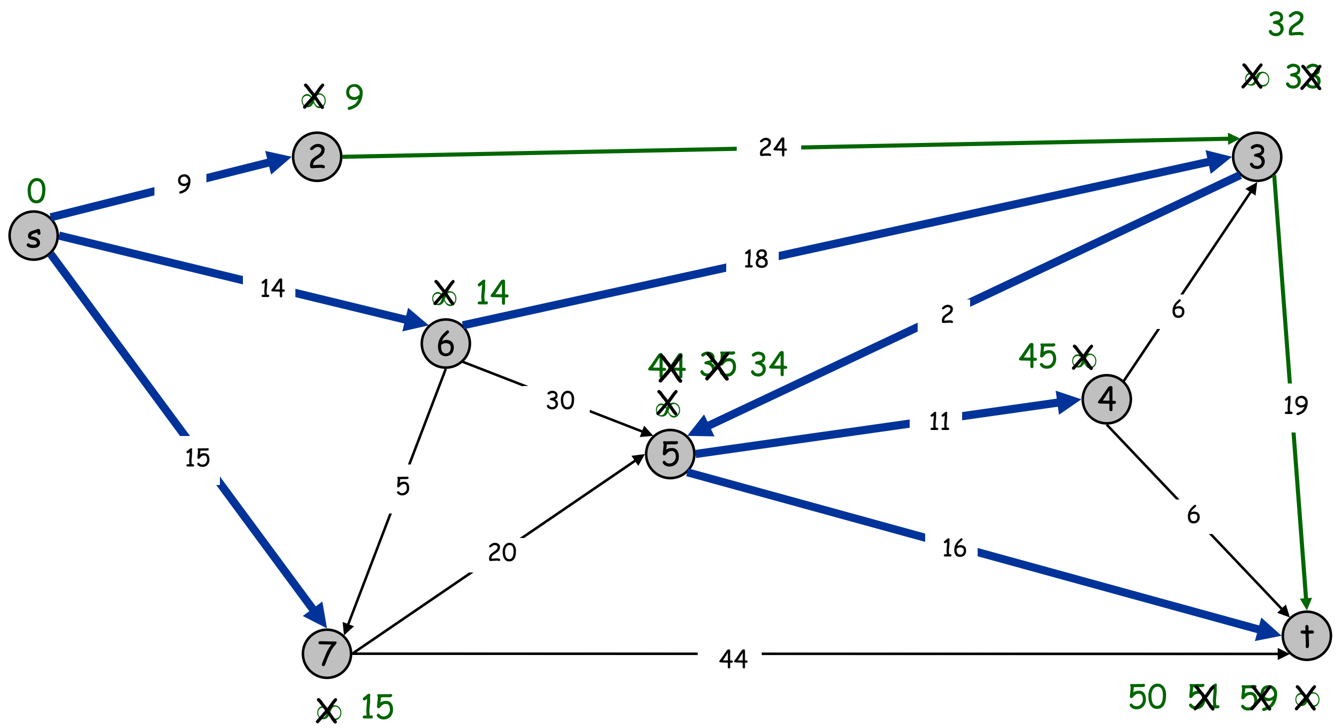
Dijkstra's Shortest Path Algorithm

S = { s, 2, 6 }
PQ = { 3, 4, 5, 7, t }



Dijkstra's Shortest Path Algorithm - solution

S = { s, 2, 3, 4, 5, 6, 7, t }
PQ = { }



Applications of shortest path algorithms:

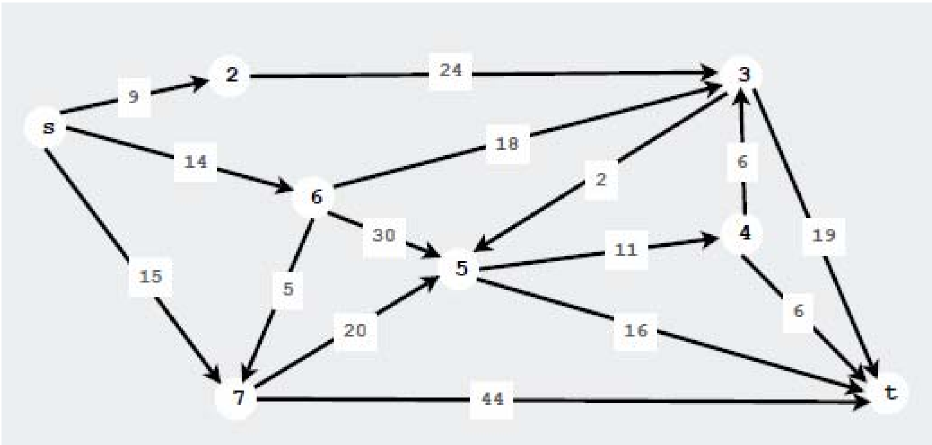
- plotting routes on Google maps
- robot path planning
- urban traffic planning (e.g. through congestion/road works)
- routing of communications messages
- and many more …

Shortest path using linear programming

Given: a weighted directed graph, with a single source s

Distance from s to v: length of the shortest part from s to v

Goal: Find distance (and shortest path) to every vertex



More …

- Applications:
  - Snakes in computer vision: dynamic programming on a closed or open chain
  - Detecting human limb layout in images: dynamic programming on trees

- David Mackay's book for message passing

- More on web page:
  - http://www.robots.ox.ac.uk/~az/lectures/opt