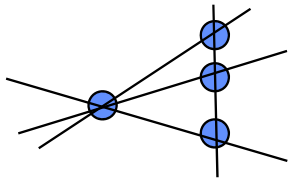
Global Processing via the Hough Transform

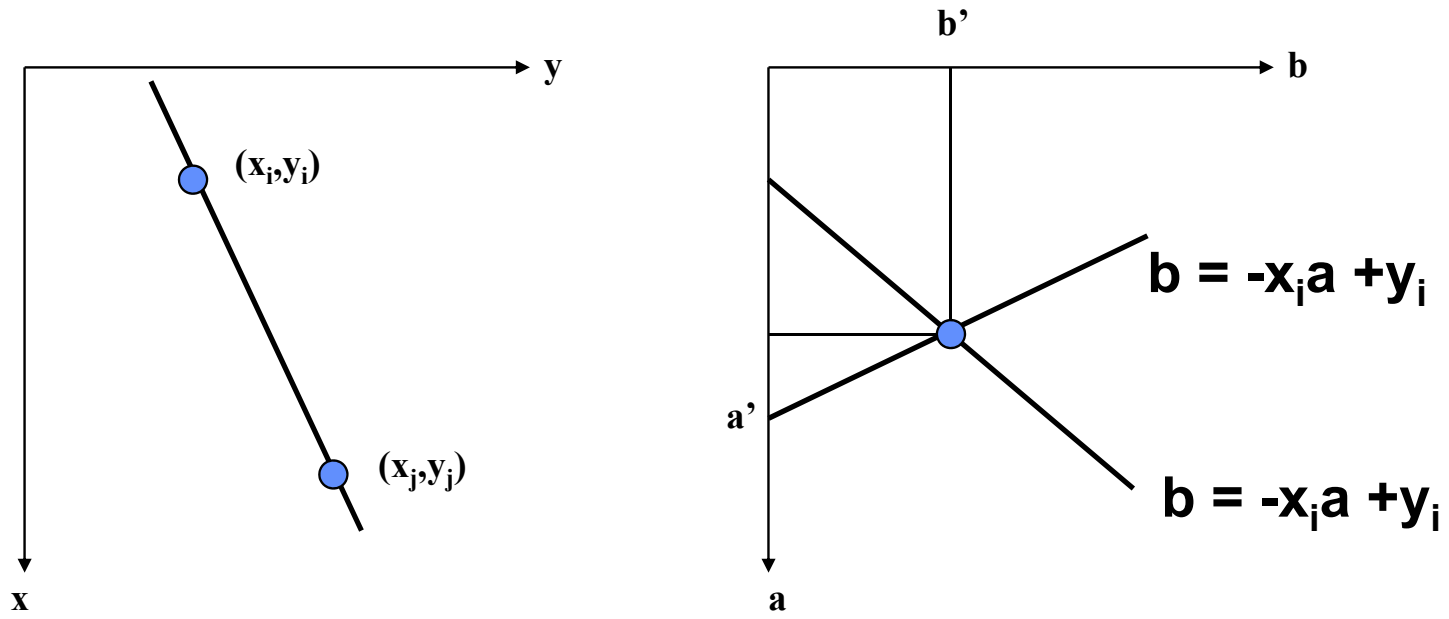
- Here we consider linking points by determining whether they lie on a specified curve or line
- This processing considers global relationships between pixels
- Suppose for *n* points in an image, we want to find subsets of these points that lie on straight lines
- One possible solution is to first find all lines determined by every pair of points and then find subsets of points associated with each line
- Involves $n(n-1)/2 \sim n^2$ lines and $(n)(n(n-1))/2 \sim n^3$ comparisons
- prohibitively expensive in all but trivial situations!



Hough Transform

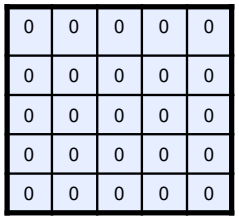
- Hough (1962) proposed an alternative referred to as the *Hough transform*
- Consider a point (x_i,y_i) and the general equation of a straight line $y_i = a\,x_i + b$
- Infinitely many lines pass through (x_i,y_i) but they all satisfy $y_i = a\,x_i + b$ for varying values of a and b
- However, expressing this equation as $b = -x_i a + y_i$ and considering the (a,b) *parameter* plane yields the equation of a single line for a fixed pair (x_i,y_i)
- Further a second point (x_j,y_j) also has a line associated with it and the two lines intersect at (a',b')

XY and Parameter Planes



Accumulator Cells

- Subdivide a and b axes in parameter space into bins and set bin value to zero
- a and b ranges can be limited to cover only expected slopes of lines in image
- For each (x,y) point in edge image, increment the corresponding line of bins
- Peaks of Hough transform indicate groups of collinear pixels
- Reverse process to find edge pixels on straight lines



Comments

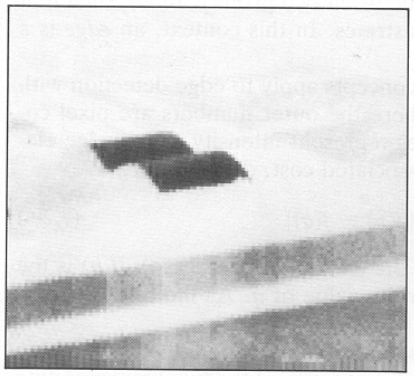
- One problem with using ab space is that the slope and the intercept approach infinity as the line approaches the vertical
- A solution is to use the following representation for a line

$$x\cos\theta + y\sin\theta = \rho$$

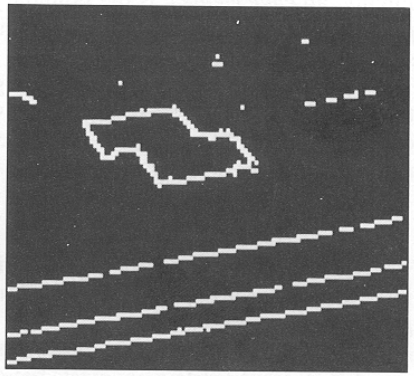
- Now points yield sinusoidal curves in parameter space
- Method can be extended to detect circles and other shapes rather than lines.

Hough Transform Example

Original Infrared Image

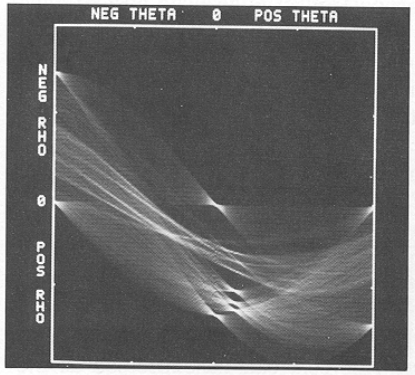


(a)



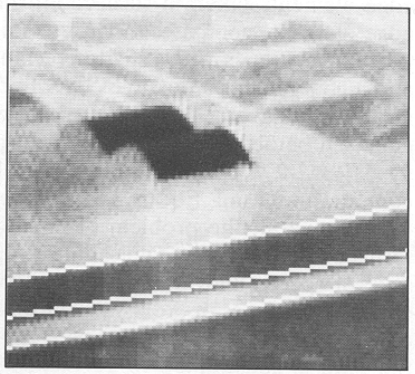
(b)

Edge Detected



(c)

Hough Transform



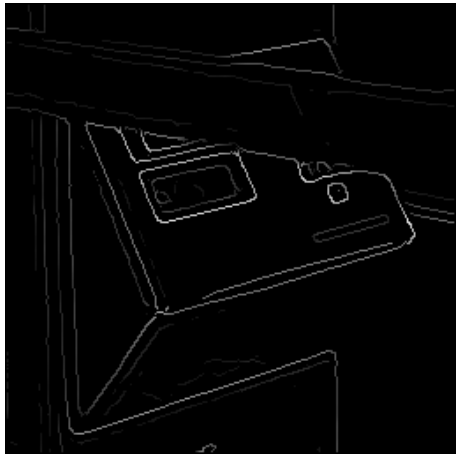
(d)

Projected Back onto Image

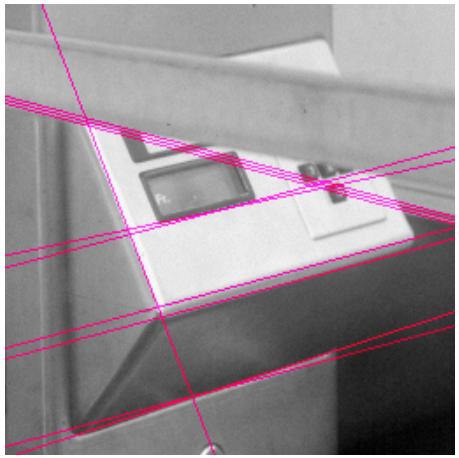
Figure 7.19 *(a) Infrared image; (b) gradient image; (c) Hough transform; (d) linked pixels. (Courtesy of D. R. Cate, Texas Instruments, Inc.)*

Another Example

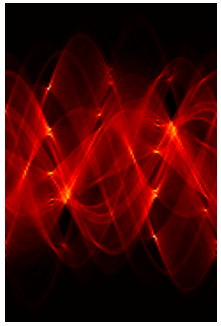
Original



Edge Detection



Found Lines

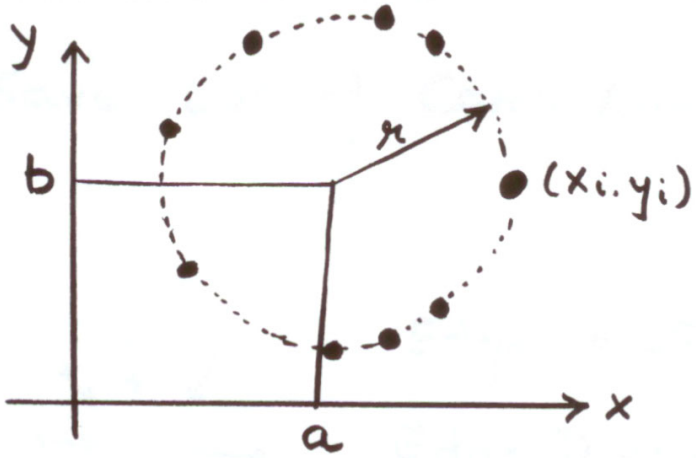


Parameter Space

Finding Circles by Hough Transform

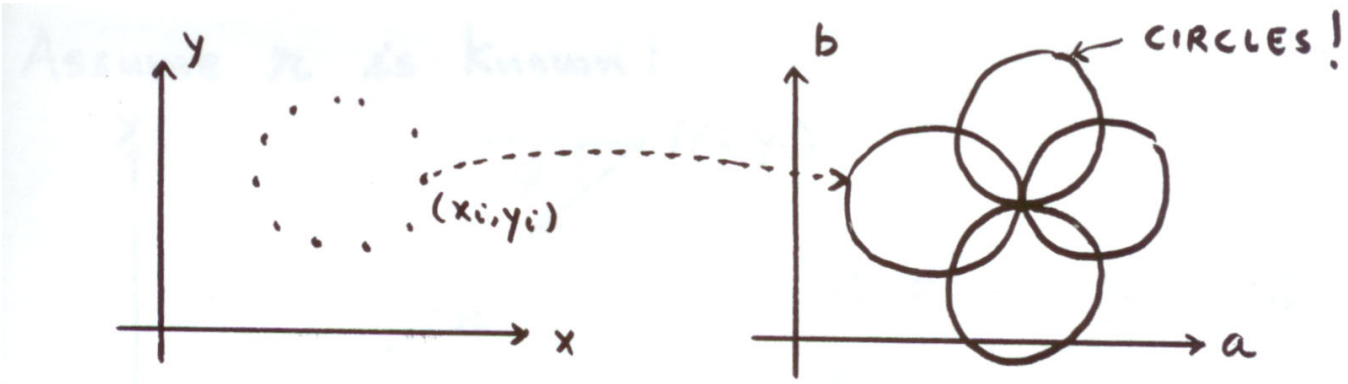
Equation of Circle:

$$(x_i - a)^2 + (y_i - b)^2 = r^2$$



If radius is known: (2D Hough Space)

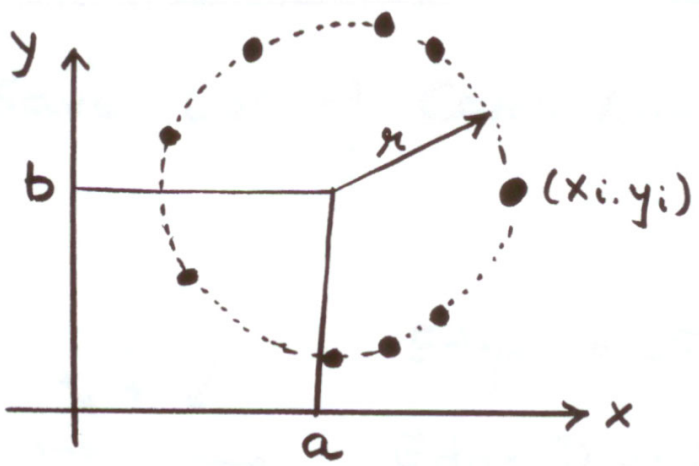
Accumulator Array $A(a,b)$



Finding Circles by Hough Transform

Equation of Circle:

$$(x_i - a)^2 + (y_i - b)^2 = r^2$$



If radius is not known: 3D Hough Space!

Use Accumulator array $A(a, b, r)$

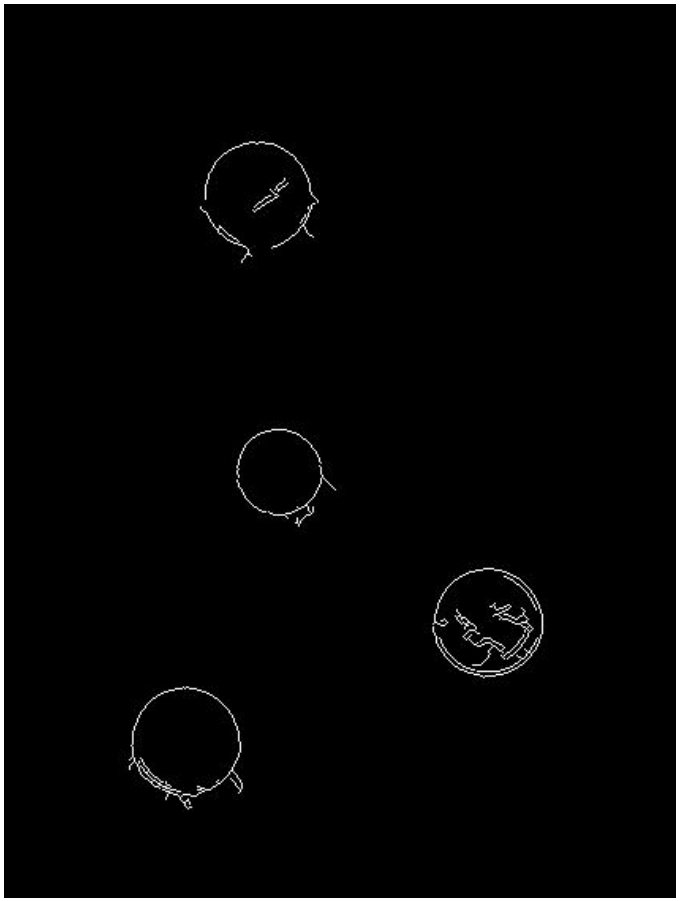
Real World Circle Examples

Crosshair indicates results of Hough transform, bounding box found via motion differencing.

Finding Coins

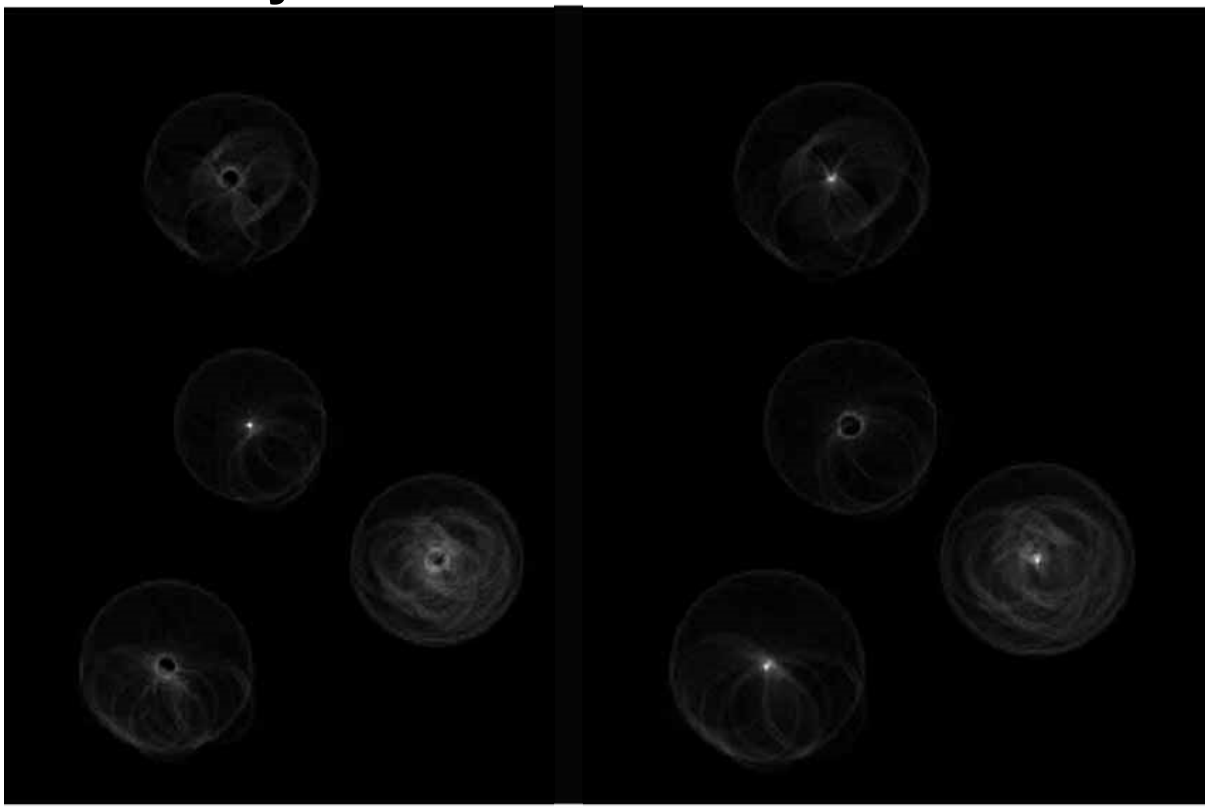
Original

Edges (note noise)

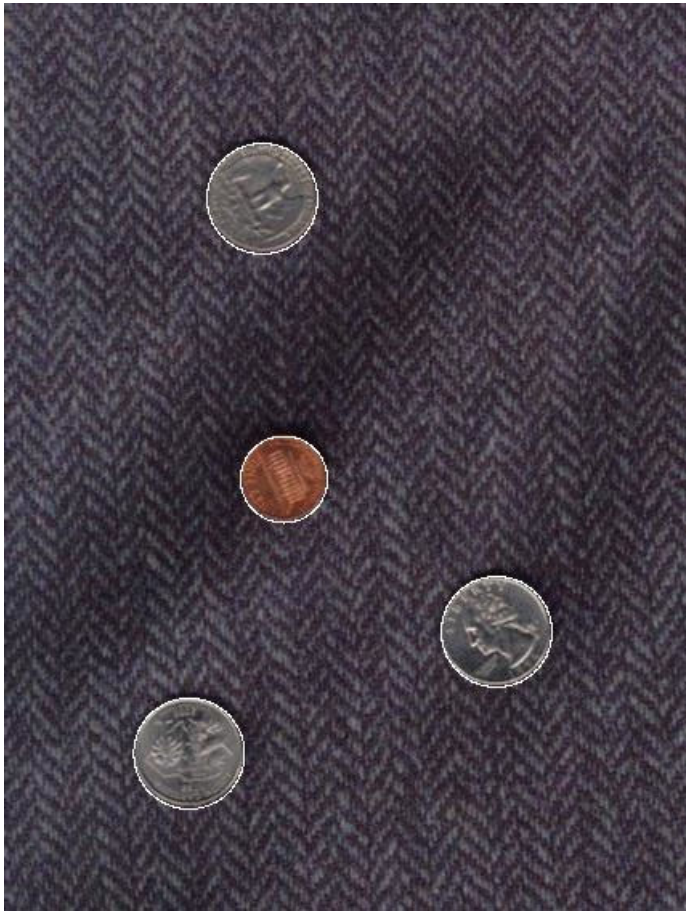


Finding Coins (Continued)

Penny **Quarter**



Finding Coins (Continued)



Note that because the quarters and penny are different sizes, a different Hough transform (with separate accumulators) was used for each circle size.

Hough Transform: Comments

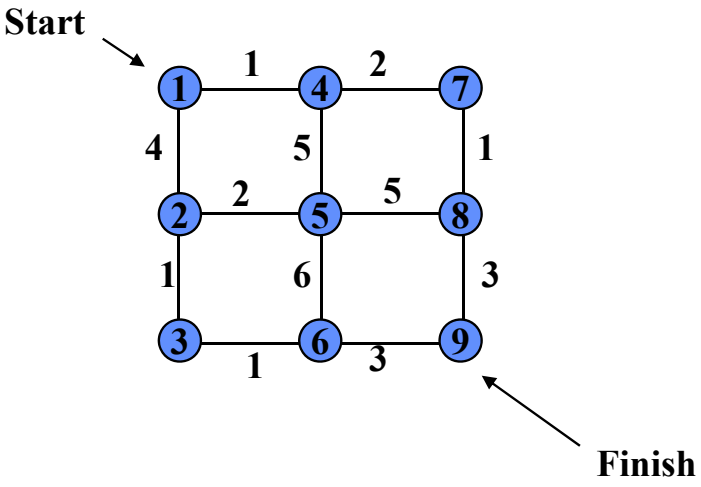
- **Works on Disconnected Edges**
- **Relatively insensitive to occlusion**
- **Effective for simple shapes (lines, circles, etc)**
- **Often used to find symbols on scanned maps etc**
- **Trade-off between work in Image Space and Parameter Space**
- **Handling inaccurate edge locations:**
    - **Increment Patch in Accumulator rather than a single point**

Graph Theoretic Approaches

- Gradient detection methods by themselves are seldom suitable for preprocessing when there is any noise present
- A better approach is a global approach based on treating the pixel as a graph and finding a low cost paths that correspond to significant edges
- These methods tend to work well but are usually computationally expensive

Finding a Low Cost Path

Consider a set of nodes, n_i, with differing costs associated with paths between the nodes, c_i. A graph in which the arcs are directed is called a *directed graph*. A sequence of nodes n_1, n_2,… n_k with each node n_i being a successor of node n_{i-1} is called a *path*.



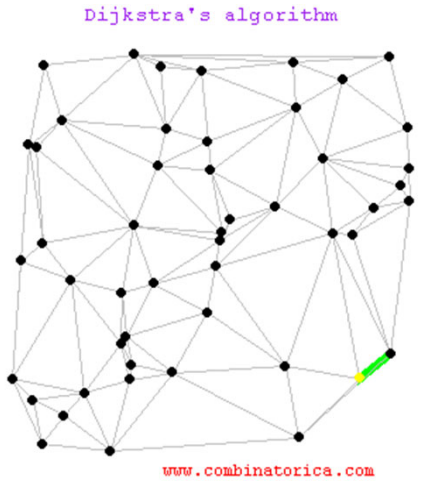
Cost of the path is

$$c = \sum_{i=2}^{k} c(n_{i-1}, n_i)$$

Principle of Optimality

- "If a node n is on the optimal (lowest cost) path between nodes and A and B, then the paths *An* and *nB* must also be optimal"
- For example, this means that if the shortest route between say Brisbane and Sydney includes Newcastle, then that section of the route between Newcastle and Sydney must also be the shortest route between those two cities.
- This principle may be considered self evident, but it allows great simplification in computation of optimal paths because suboptimal routes can be very rapidly discarded.

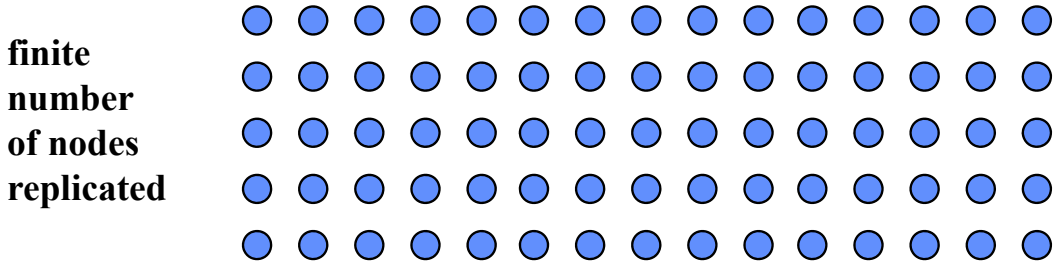
Dijkstra's Algorithm



- **Shortest Path from single source problem on a graph (or network)**
  - Given a connected graph G=(V,E), a weight d:E->R+ and a fixed vertex s in V, find a shortest path from s to each vertex v in V.
- **Dijkstra's Algorithm**
  - Dijkstra's algorithm is known to be a good algorithm to find a shortest path.
  - Set i=0, S0= {u0=s}, L(u0)=0, and L(v)=infinity for v <> u0. If |V| = 1 then stop, otherwise go to step 2.
  - For each v in V\Si, replace L(v) by min{L(v), L(ui)+dvui}. If L(v) is replaced, put a label (L(v), ui) on v.
  - Find a vertex v which minimizes {L(v): v in V\Si}, say ui+1.
  - Let Si+1 = Si cup {ui+1}.
  - Replace i by i+1. If i=|V|-1 then stop, otherwise go to step 2.
  - The time required by Dijkstra's algorithm is O(|V|2). It will be reduced to O(|E|log|V|) if heap is used to keep {v in V\Si : L(v) < infinity}.

Viterbi Search on a Trellis

- The Viterbi algorithm is very famous and this algorithm is used in many applications including Hidden Markov Modeling and the V32/V90 telephone modem
- This algorithm is a very fast and optimal method for searching a trellis for a minimum cost path
- Similar to Dijkstra's shortest paths from single source algorithm except that self-transitions are allowed and have a cost.



Trellis

Viterbi Algorithm as Used in HMMs

1) Initialization:

$$\delta_1(i) = \pi_i b_i(O_1), \quad 1 \le i \le N \tag{32a}$$

$$\psi_1(i) = 0. \tag{32b}$$

2) Recursion:

$$\delta_t(j) = \max_{1 \le i \le N} [\delta_{t-1}(i) a_{ij}] b_j(O_t), \quad 2 \le t \le T, \quad 1 \le j \le N \tag{33a}$$

$$\psi_t(j) = \operatorname{argmax}_{1 \le i \le N} [\delta_{t-1}(i) a_{ij}], \quad 2 \le t \le T, \quad 1 \le j \le N. \tag{33b}$$

3) Termination:

$$P^* = \max_{1 \le i \le N} [\delta_T(i)] \tag{34a}$$

$$q_T^* = \operatorname{argmax}_{1 \le i \le N} [\delta_T(i)]. \tag{34b}$$

4) Path (state sequence) backtracking:

$$q_t^* = \psi_{t+1}(q_{t+1}^*), \quad t = T-1, T-2, \cdots, 1. \tag{35}$$

Set initial costs

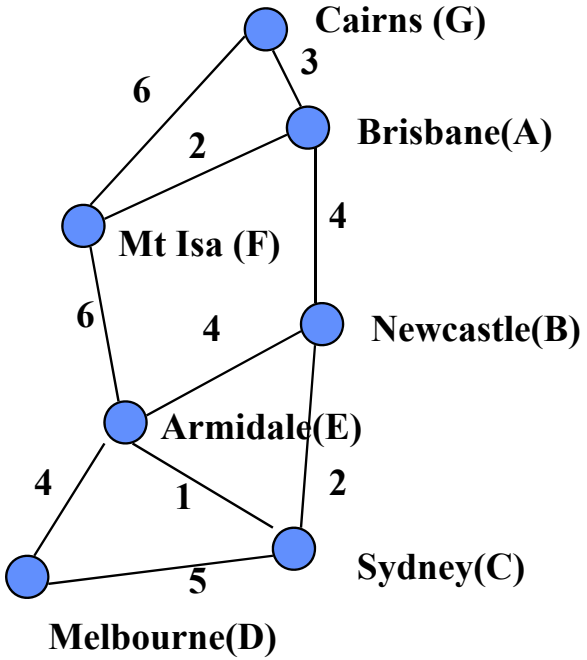
Move through trellis keeping only optimal paths

Find destination node

Backtrack to find path

Viterbi-Like Shortest Path Search

Find Lowest Cost Path from Brisbane to Sydney



Algorithm

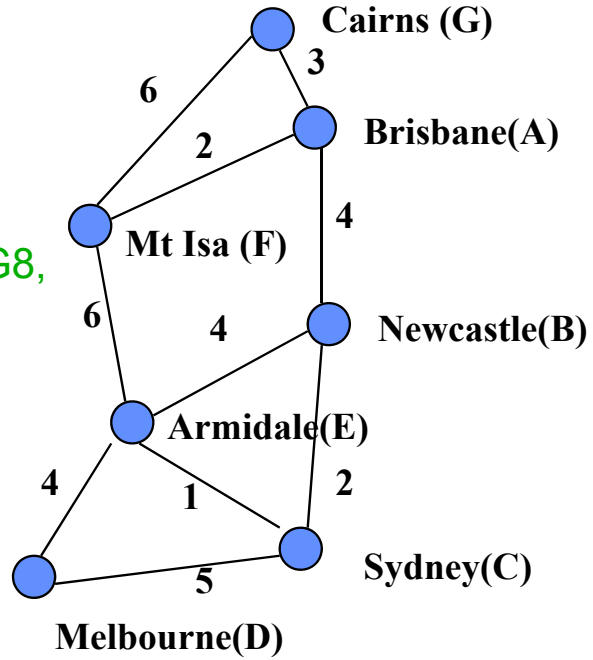
1. Find all cities reachable with one hop from Brisbane. Prune to keep only best path for each destination city.
2. Find all cities reachable with two hops from Brisbane. Prune to keep best path for each destination city. etc.

Continue until destination city is found, and all paths exceed current destination city path cost. (need to ensure that no low cost, high hopcount path exists)

Worked Example

A	B	C	D	E	F	G
0	4	6	-	8	2	3

- Looking for node C (Sydney)
- One Hop: AB4, AF2, AG3
- All Paths Optimal: AB4, AF2, AG3
- Two Hops: ABE8, ABC6, AFG8, AFE8, AGF9
- Prune to Optimal paths: ABC6, ABE8, AFG8, AGF9
- No shorter paths possible
- Solution: ABC6



Notes

- The number of optimal paths at each stage will never exceed the number of nodes since we can immediately prune sub-optimal paths.
- If we had a trellis of 30 nodes with a path length of 30, we would require 30^{30} ~$2x10^{44}$ path evaluations to find the optimal path by exhaustive searching
- With the Viterbi algorithm it only takes of the order of 30x30 ~ 1000 evaluations (also depends on average node fan-out). That is, the algorithm is linear in computational complexity.

Application

- This method has been used to speed up a patented cell image segmentation algorithm as described in
- *Pascal Bamford and Brian Lovell, "Unsupervised Cell Nucleus Segmentation with Active Contours," Signal Processing – Special Issue on Deformable Models and Techniques for Signal and Image Processing, V 71, pp 203-213*