Discrete Energy Minimisation

Applications in Image Processing

Credit: Ben Appleton

What's it all about?

- Many methods in image analysis are task-oriented
  - Like a recipe: perform operations a, b, c on the image to produce the desired output
  - Example: Perform a thresholding then an opening to find the left ventricle
- Energy minimisation is goal-oriented
  - State your aim, and then apply an algorithm to find it optimally
  - Your answer will always make sense, as the solution will be the best according to your goal

What we'll be talking about

- Dynamic Programming
- Shortest Path Algorithms: Dijkstra and Fast Marching
- Evolving Contours and Surfaces: Level Sets
- Applications

Dynamic Programming/Viterbi

- Motivating Example: String Matching
  - Given two strings, where one has had letters deleted, inserted or mistyped, find the minimum number of changes to make them match.
  - Used in some word processors to autocorrect a spelling error to the nearest word from a dictionary
  - One of the main techniques to perform stereo matching (with suitable alteration)

Fundamental Operations

Operation	Cost
Delete a letter from the ‘left’ word	1
Delete a letter from the ‘top’ word	1
Match two letters	0 if they match, 1 otherwise

Distance function

- The value of an element (x,y) in the matrix is the cost of matching the corresponding sub words.
- For example, $d(5, 4)$ is the cost of matching 'ELEPH' to 'EALE'
- d(8, 8) will then be the cost of matching 'ELEPHANT' to 'EALEPANT'

Matching 'ELEPHANT' to 'EALEPANT'

	E	L	E	P	H	A	N	T
E								
A								
L								
E								
P								
A								
N								
T								

Recursive Problem Definition

- Recursive definition of matching distance:

$$d(x,y) = \min \begin{Bmatrix} d(x-dx, y-dy) + c(x,y,dx,dy) \\ 0 \le dx, dy \le 1 \end{Bmatrix}$$

where c(x, y, dx, dy) is the transition cost and d(x, y) is the cumulative matching cost

Operation	Corresponding subwords	Additional Cost
Delete left	ELEPHANT EALEPAN	1
Delete top	ELEPHAN EALEPANT	1
Match	ELEPHANT EALEPANT	0

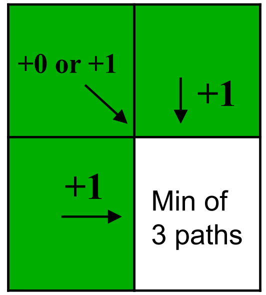
Dynamic Programming

- Solve the recursion by Dynamic Programming
- A clever caching scheme
  - Express your problem recursively
  - When calling the recursive function, save the output (indexed by the input parameters) and never make the same call twice

Matching Algorithm

Distance function

	E	L	E	P	H	A	N	T
E	0							
A								
L								
E								
P								
A								
N								
T								

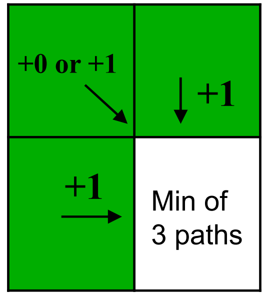


0 if match
1 if no match

Matching Algorithm

Distance function

	E	L	E	P	H	A	N	T
E	0							
A	1							
L	2							
E	3							
P	4							
A	5							
N	6							
T	7							

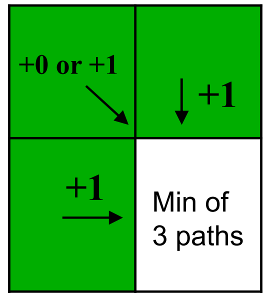


0 if match
1 if no match

Matching Algorithm

Distance function

	E	L	E	P	H	A	N	T
E	0	1						
A	1	1						
L	2	1						
E	3	2						
P	4	3						
A	5	4						
N	6	5						
T	7	6						

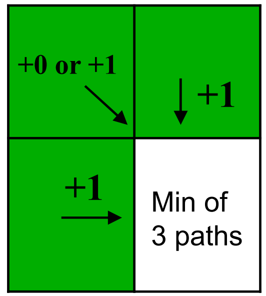


0 if match
1 if no match

Matching Algorithm

Distance function

	E	L	E	P	H	A	N	T
E	0	1	2	3	4	5	6	7
A	1	1	2	3	4	4	5	6
L	2	1	2	3	4	5	5	6
E	3	2	1	2	3	4	5	6
P	4	3	2	1	2	3	4	5
A	5	4	3	2	2	2	3	4
N	6	5	4	3	3	3	2	3
T	7	6	5	4	4	4	3	2



0 if match
1 if no match

Matching Algorithm

Matching path

	E	L	E	P	H	A	N	T
E	0	1	2	3	4	5	6	7
A	1	1	2	3	4	4	5	6
L	2	1	2	3	4	5	5	6
E	3	2	1	2	3	4	5	6
P	4	3	2	1	2	3	4	5
A	5	4	3	2	2	2	3	4
N	6	5	4	3	3	3	2	3
T	7	6	5	4	4	4	3	2

Applications of Dynamic Programming/Viterbi

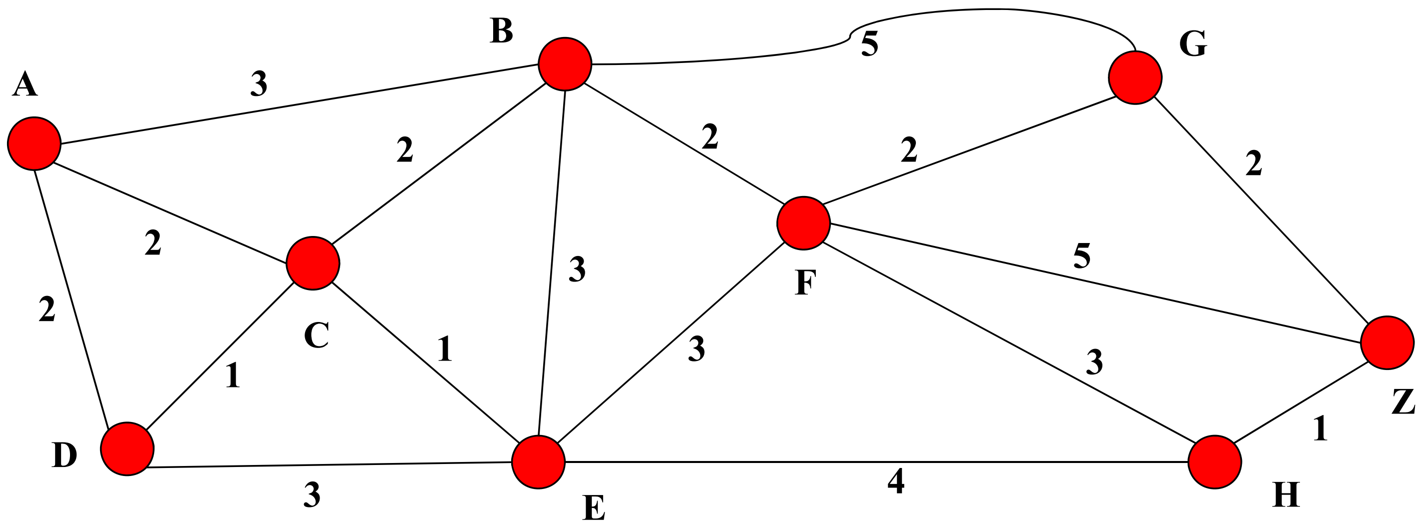
- String matching, stereo matching
- Hidden Markov Models (recognition)
  - Speech recognition - finding the *most likely* word
- Optimal control in State Space
  - Least time path for *speed*
  - Least energy path for *efficiency*
- General 1D sequence optimisation

Dijkstra’s Algorithm and Fast Marching

- DP/Viterbi is good for problems that can be solved ‘one column at a time’
  - Not good for general path finding
- Dijkstra and Fast Marching are general shortest path algorithms

Dijkstra's Algorithm

- Motivating Example:
  - Given a map, with a list of cities and the distances between them, find the shortest path from A to Z



Dijkstra's Algorithm

- Express shortest distance recursively:

$$d(v) = \min_{\forall u \in V} \{d(u) + c(u, v)\}$$

where d(*v*) is the distance to vertex *v*

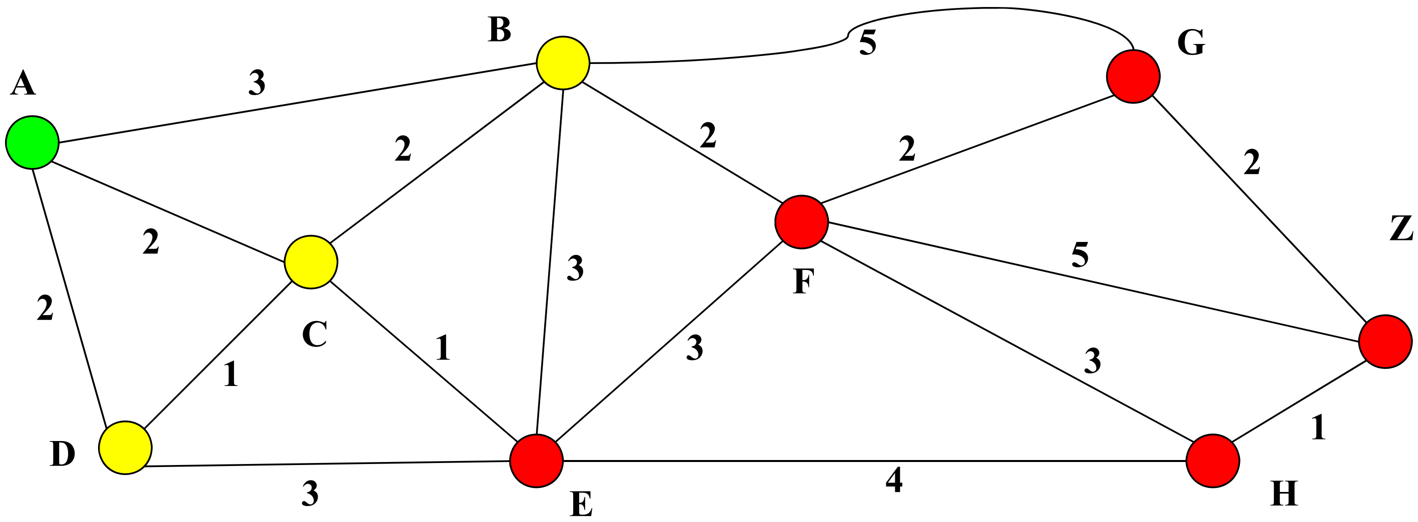
and c(*u*, *v*) is the cost of travelling from *u* to *v*

- **Solve for d(v) in order of distance**
  - **'Expanding wavefront' view of shortest path finding**
  - **Guarantees that we don't miss any shorter paths**

Example: Path planning

- ‘Known’ vertex – distance finalised
- ‘Trial’ vertex – intermediate value uncertain
- ‘Far’ vertex – not yet reached by wavefront propagation

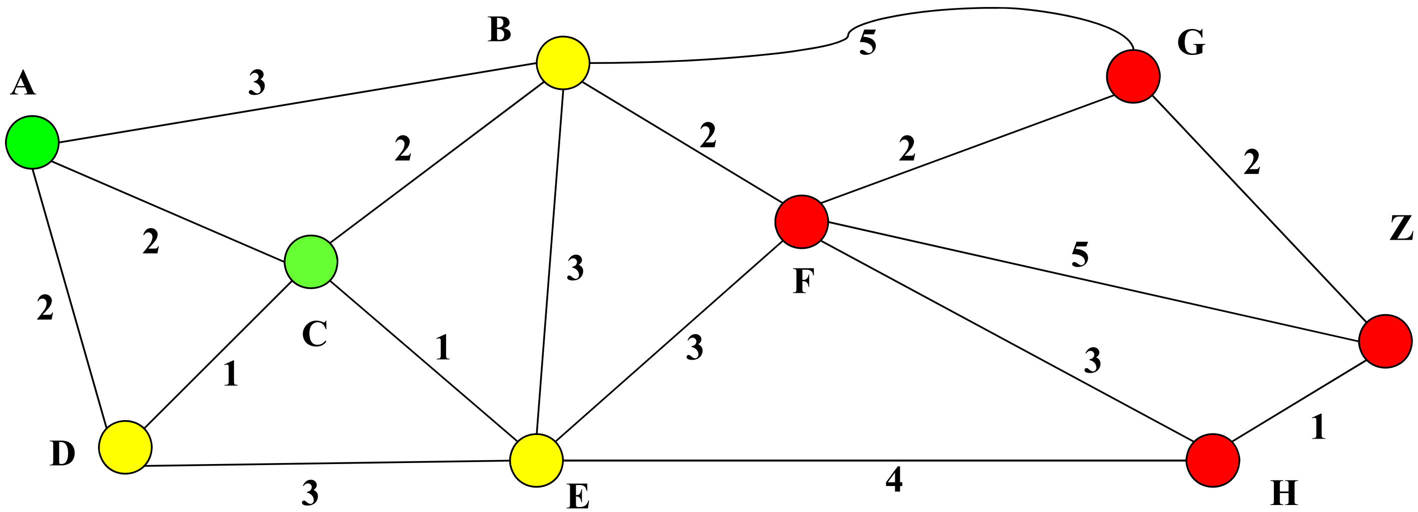
Vertex	A	B	C	D	E	F	G	H	Z
Distance	0	3	2	2	?	?	?	?	?



Example: Path planning

- **‘Known’ vertex – distance finalised**
- **‘Trial’ vertex – intermediate value uncertain**
- **‘Far’ vertex – not yet reached by wavefront propagation**

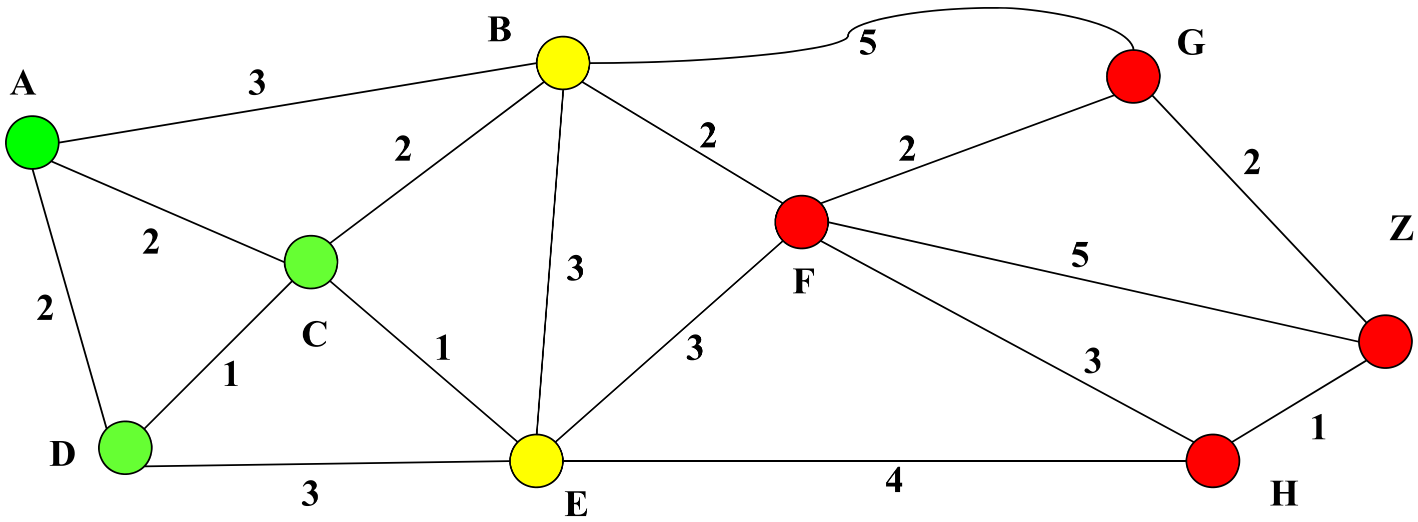
Vertex	A	B	C	D	E	F	G	H	Z
Distance	0	3	2	2	3	?	?	?	?



Example: Path planning

- ‘Known’ vertex – distance finalised
- ‘Trial’ vertex – intermediate value uncertain
- ‘Far’ vertex – not yet reached by wavefront propagation

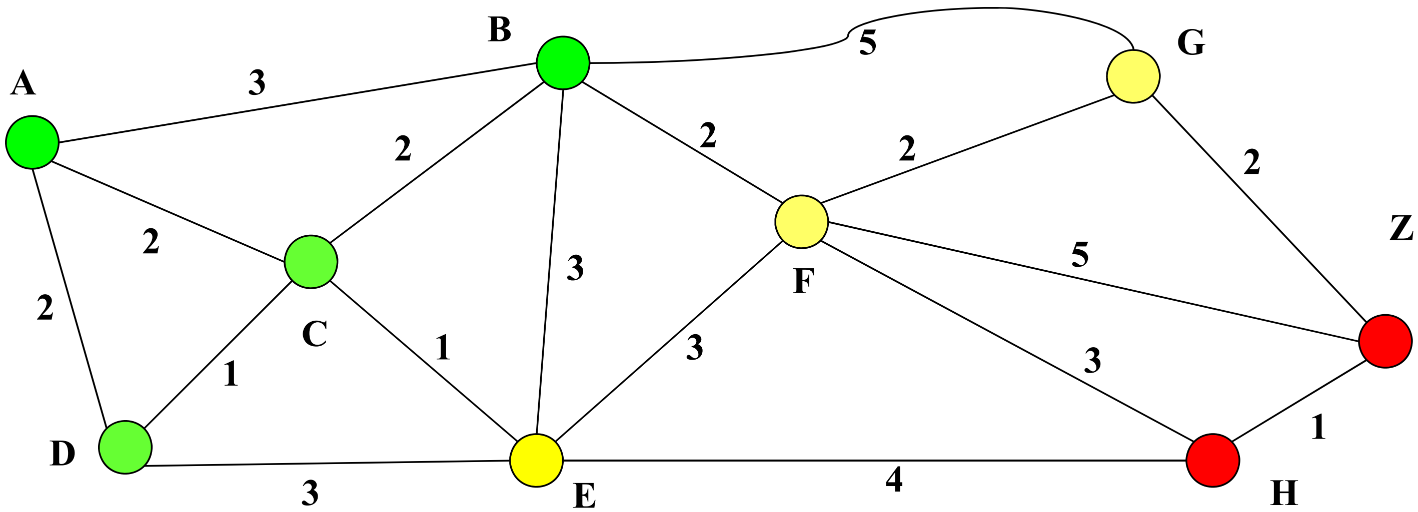
Vertex	A	B	C	D	E	F	G	H	Z
Distance	0	3	2	2	3	?	?	?	?



Example: Path planning

- **'Known' vertex – distance finalised**
- **'Trial' vertex – intermediate value uncertain**
- **'Far' vertex – not yet reached by wavefront propagation**

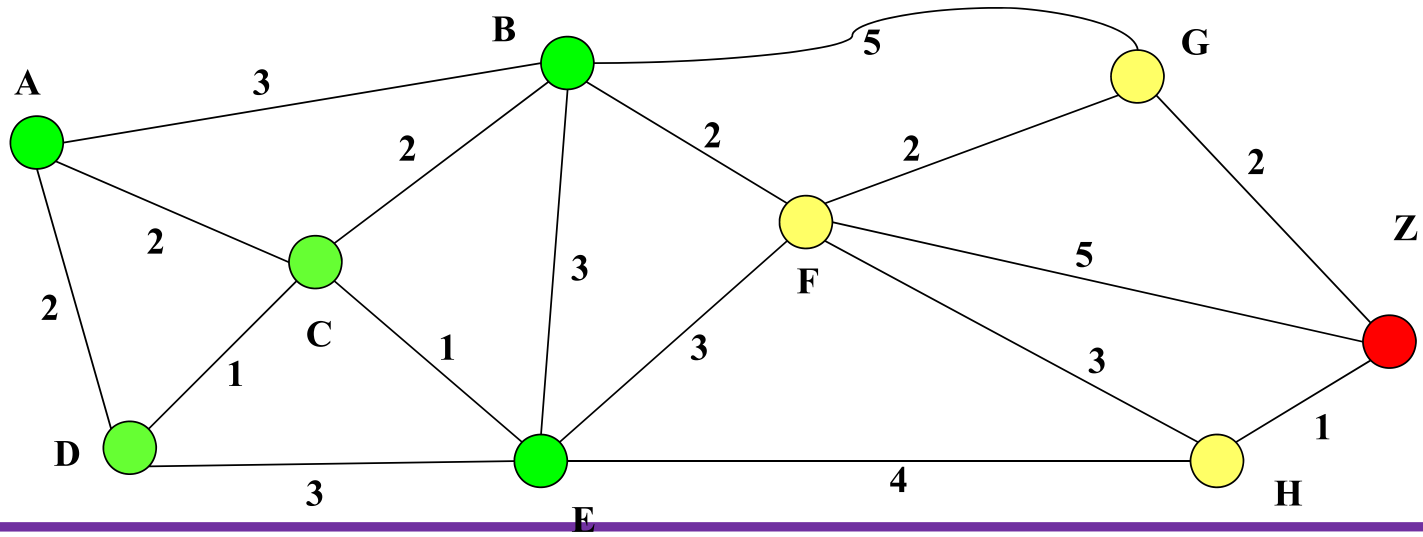
Vertex	A	B	C	D	E	F	G	H	Z
Distance	0	3	2	2	3	5	8	?	?



Example: Path planning

- **‘Known’ vertex – distance finalised**
- **‘Trial’ vertex – intermediate value uncertain**
- **‘Far’ vertex – not yet reached by wavefront propagation**

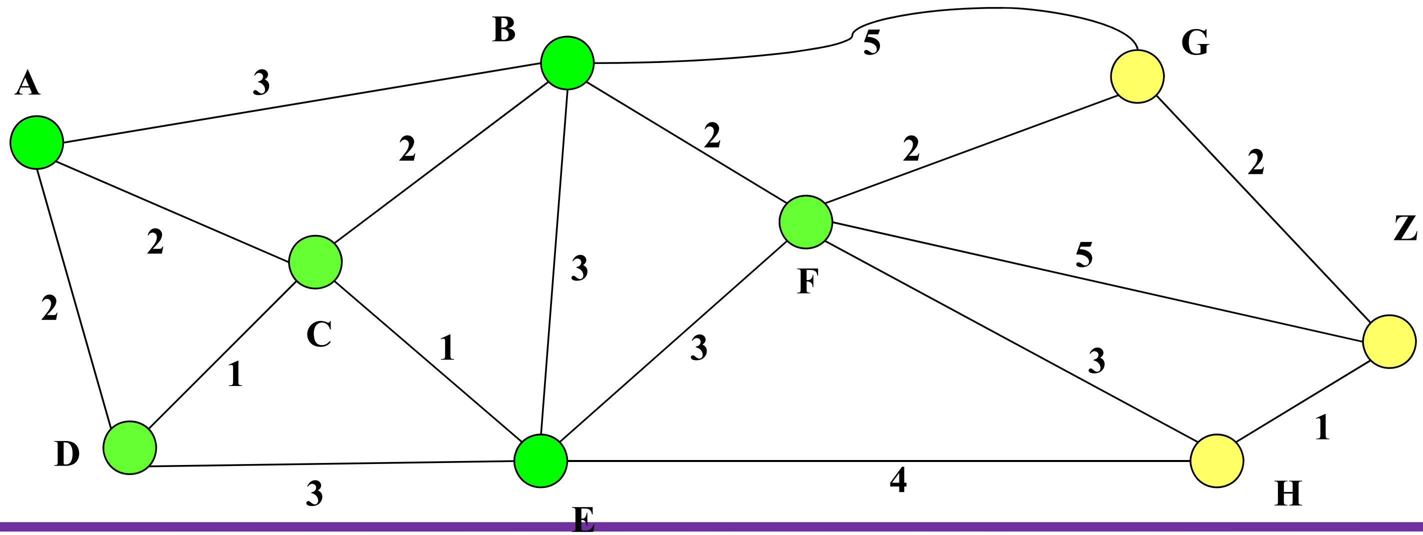
Vertex	A	B	C	D	E	F	G	H	Z
Distance	0	3	2	2	3	5	8	7	?



Example: Path planning

- **'Known' vertex – distance finalised**
- **'Trial' vertex – intermediate value uncertain**
- **'Far' vertex – not yet reached by wavefront propagation**

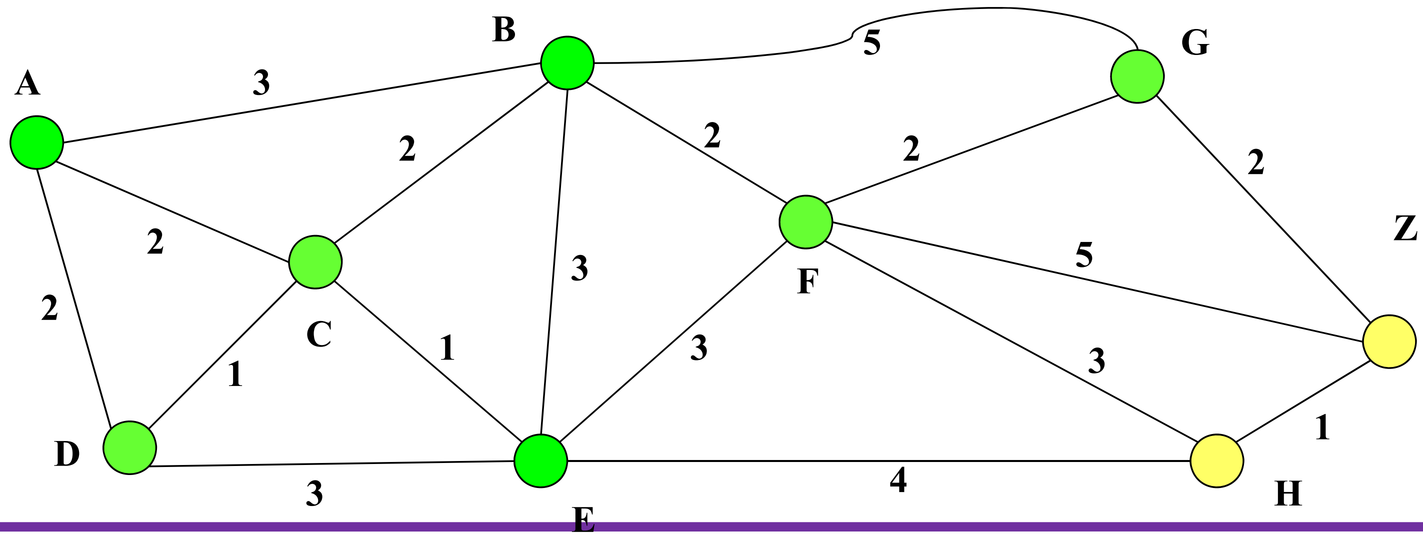
Vertex	A	B	C	D	E	F	G	H	Z
Distance	0	3	2	2	3	5	7	7	10



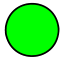
Example: Path planning

- 'Known' vertex – distance finalised
- 'Trial' vertex – intermediate value uncertain
- 'Far' vertex – not yet reached by wavefront propagation

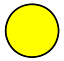
Vertex	A	B	C	D	E	F	G	H	Z
Distance	0	3	2	2	3	5	7	7	9



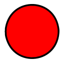
Example: Path planning



'Known' vertex – distance finalised

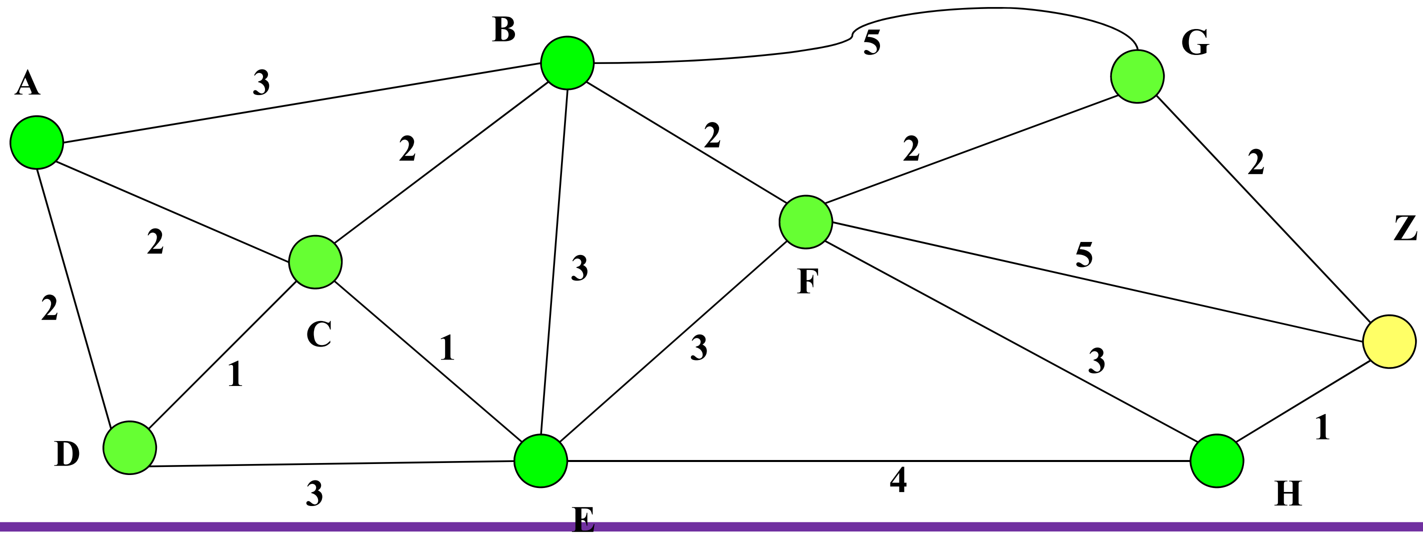


'Trial' vertex – intermediate value uncertain



'Far' vertex – not yet reached by wavefront propagation

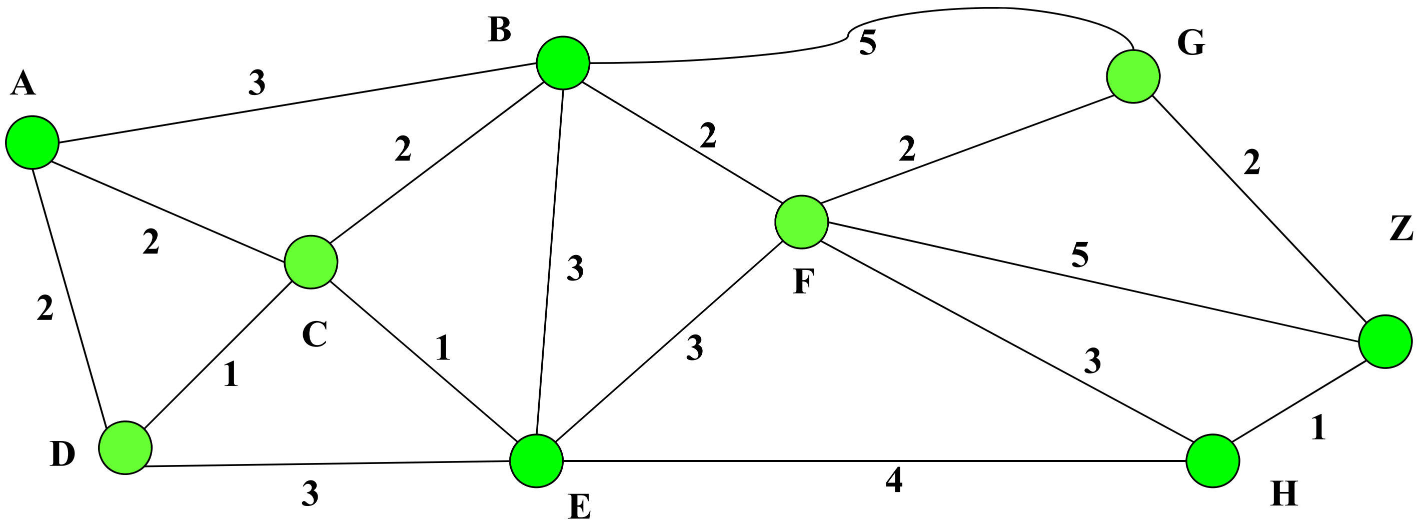
Vertex	A	B	C	D	E	F	G	H	Z
Distance	0	3	2	2	3	5	7	7	8



Example: Path planning

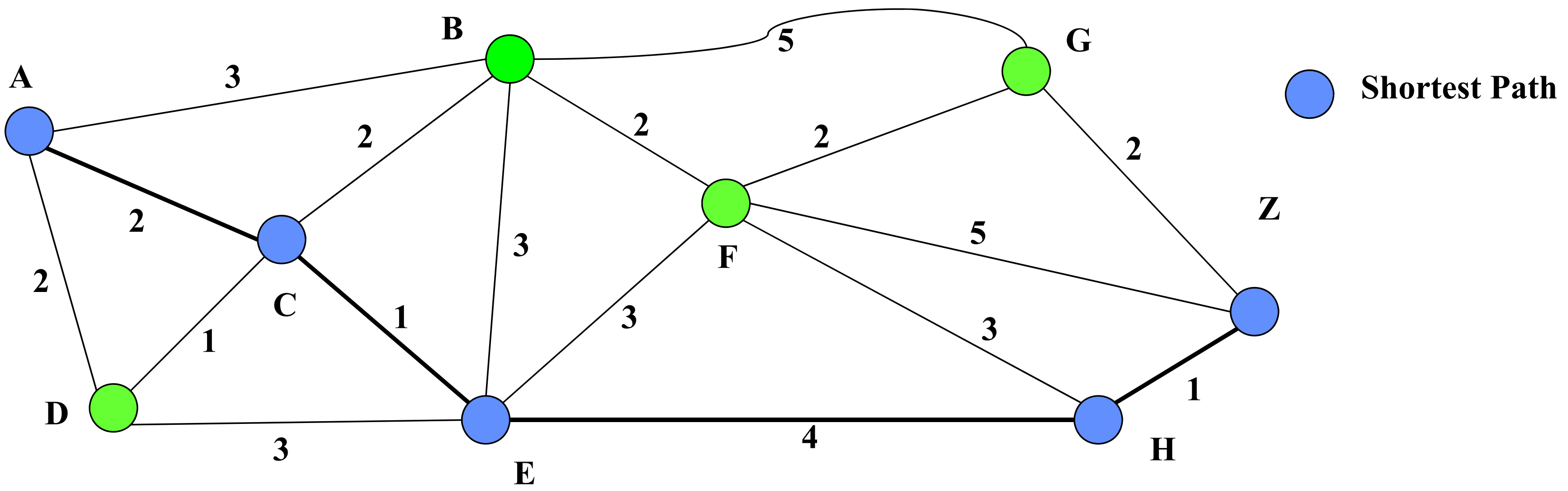
- ‘Known’ vertex – distance finalised
- ‘Trial’ vertex – intermediate value uncertain
- ‘Far’ vertex – not yet reached by wavefront propagation

Vertex	A	B	C	D	E	F	G	H	Z
Distance	0	3	2	2	3	5	7	7	8



Example: Path planning

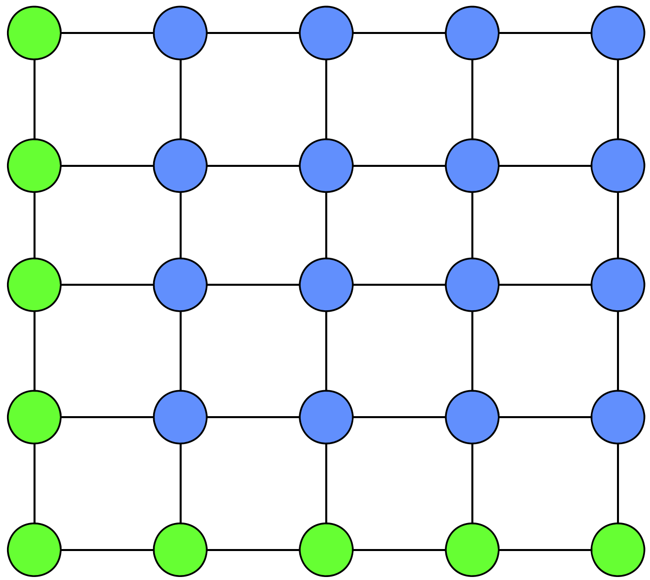
Vertex	A	B	C	D	E	F	G	H	Z
Distance	0	3	2	2	3	5	7	7	8
Backward Pointers	.	A	A	A	C	B	F	E	H



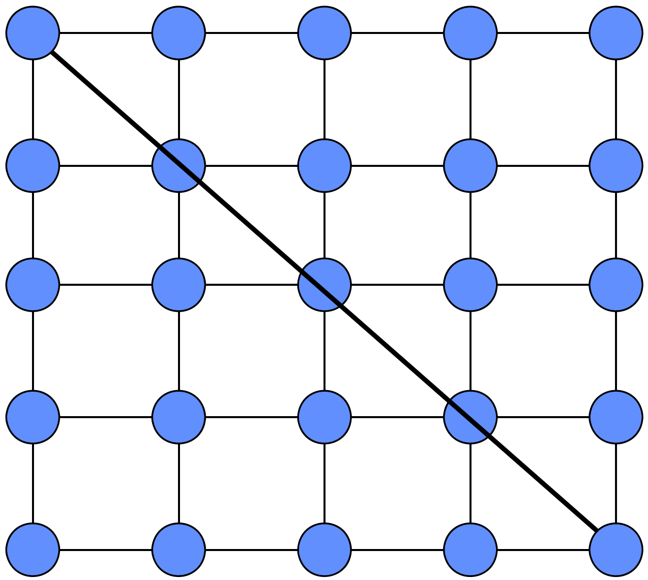
Dijkstra's Algorithm - Anisotropy

- Dijkstra's Algorithm will find the shortest network path, but can't be extended to shortest Euclidean path

Shortest Network Path (Length 8)



Shortest Euclidean Path (Length 5.66)



Fast Marching Method

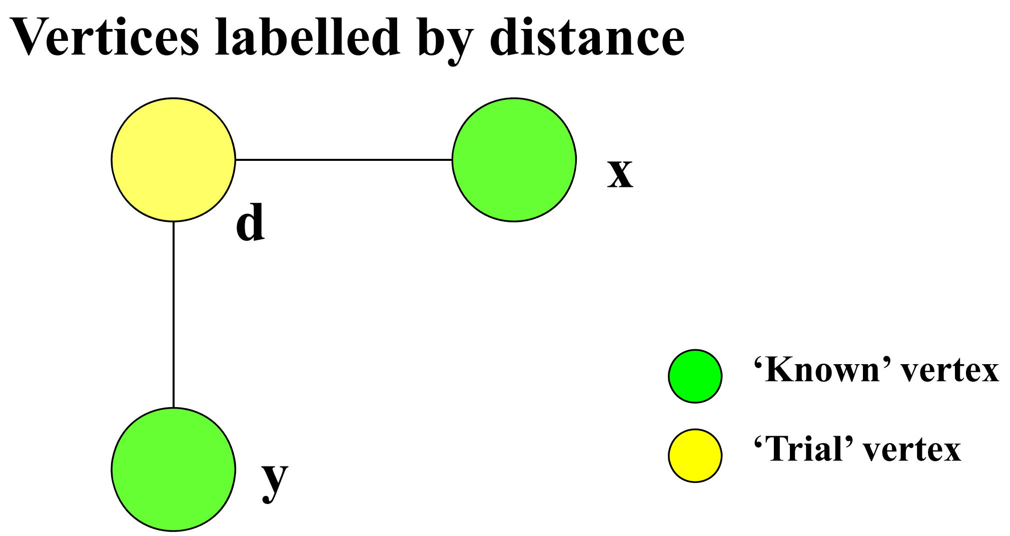
- In Euclidean space, a distance function d(x, y) is defined by

$$\left|\nabla d\right| = 1, d(C) = 0$$

for some initial contour C of zero distance

- **Solving the gradient equation for *d* at each point in the image grid will give an isotropic (Euclidean) distance function**
- **Very similar to Dijkstra's algorithm, except that all 'Known' neighbours (pixels) are used in computing the distance of a 'Trial' point.**

Fast Marching Method

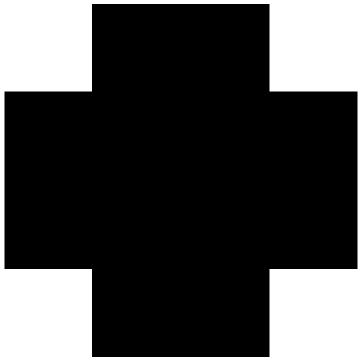


- Distance is computed by solving:

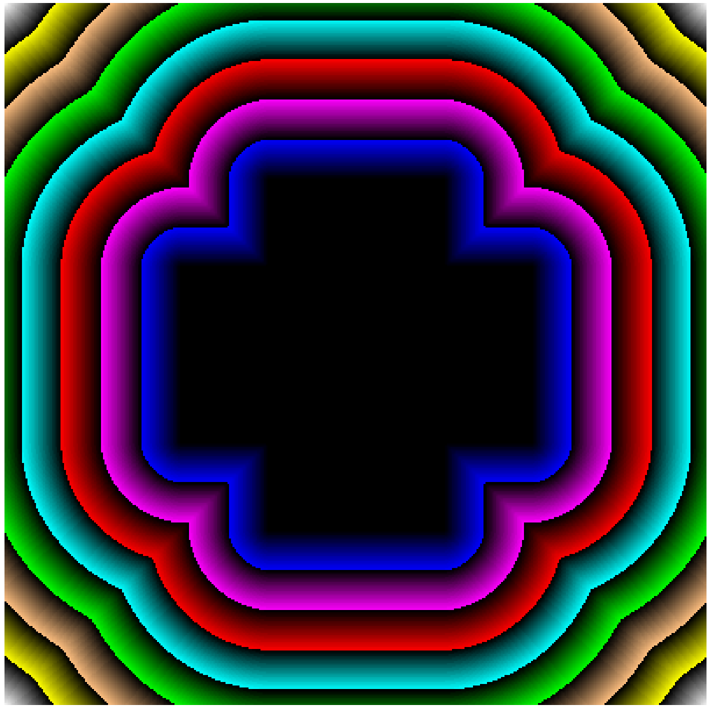
$$(d-x)^2+(d-y)^2=1$$

Fast Marching Examples

A simple contour

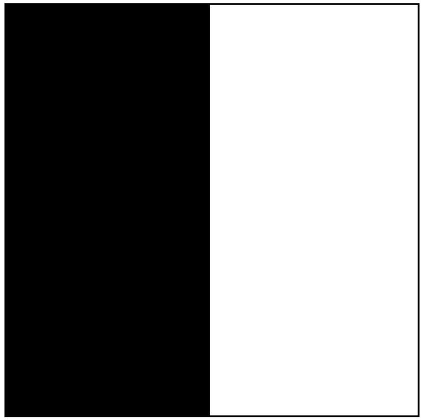


The distance function computed by Fast Marching

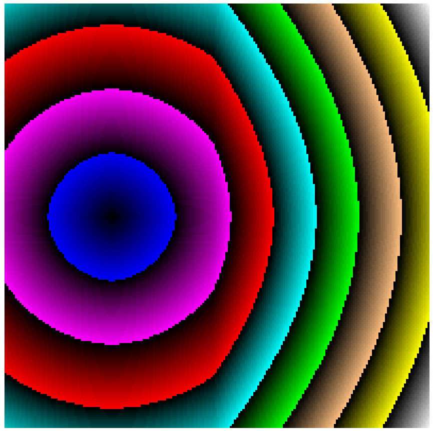


Fast Marching Examples

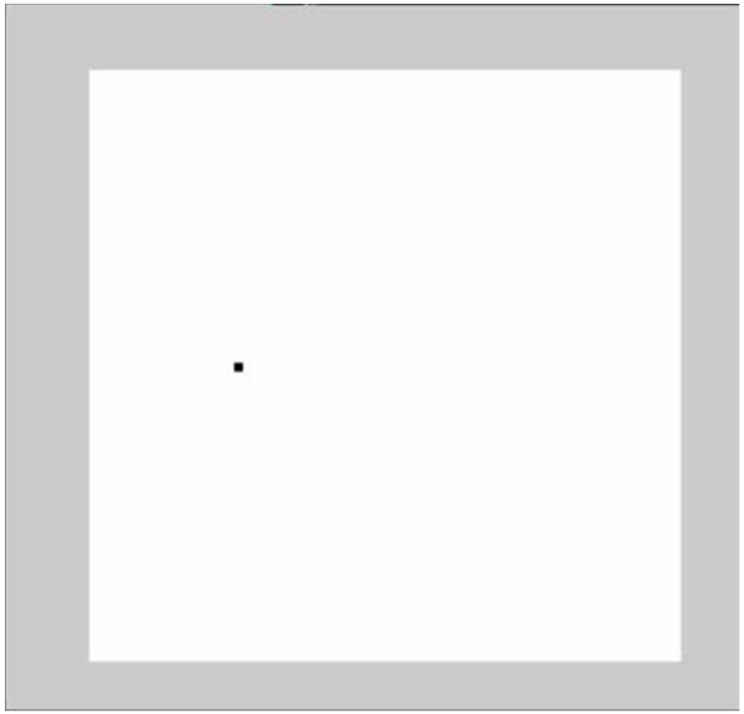
A weighted domain



The distance function
from a point



The Fast Marching computation wavefront



Optimal Control by Shortest Paths

- By suitably phrasing a control theory problem, we may apply shortest path algorithms to solve it optimally
- Here we see a robot navigating through a complicated maze as fast as possible

Path planning example

Taken from http://math.berkeley.edu/~sethian/ - an excellent reference!

Questions?

?

Active Contours and Surfaces

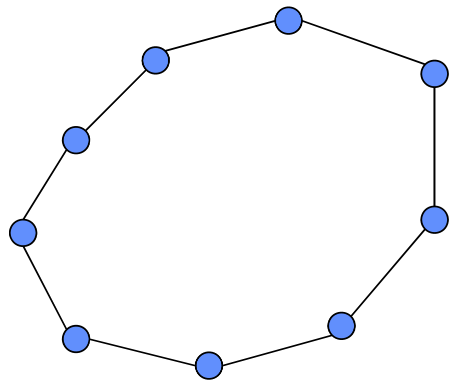
- Many problems in image analysis can be expressed as curve or surface finding
  - Segmentation in 2D searches for closed curves (1D) around each object
  - Segmentation in 3D searches for closed surfaces (2D) around each object
  - Stereo matching searches for a disparity surface (2D) in a 3D space of possible matches
  - Motion estimation searches for a velocity field, a 2D surface in a 4D space
- Some of these problems cannot be solved efficiently by global optimisation

Snakes and Level Sets

- Snakes and Level Sets are representations of curves and surfaces
- They allow for optimisation by curve and surface evolution towards the desired goal

Snakes

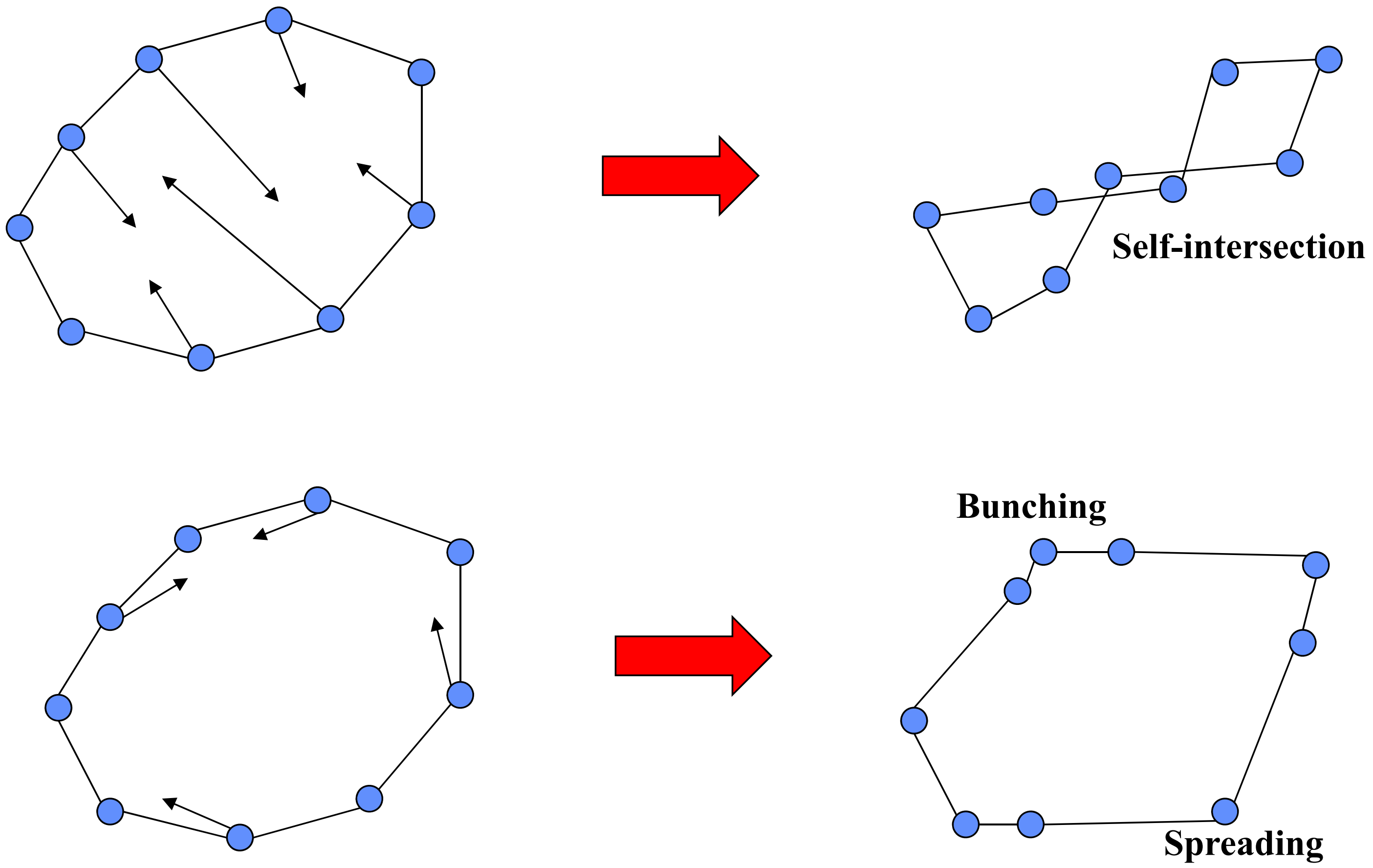
- Snakes represent a curve or surface as a polygon or polyhedra
- Evolution is performed iteratively by moving contour points towards the goal
  - Eg. To segment an object, move the contour points toward the edges of the image. This is one way to perform edge linking.



Snakes

- Snakes have a few problems:
  - They cannot easily change topology to segment multiple objects
  - The contour points often bunch up or spread out, reducing the stability or accuracy of the solution
  - Self-intersections of the curve are difficult to detect
  - They are difficult to extend to higher dimensions

Snakes



Level Sets

- Level Sets overcome these parameterisation problems
- Use an implicit representation of the contour *C* as the 0-level set of higher dimensional function ϕ

$$\phi(C) = 0$$

The Level Set Evolution Equation

- **Manipulate ϕ to indirectly move C:**

$$\phi(C) = 0$$

$$\frac{d\phi(C)}{dt} = \frac{\partial C}{\partial t} \cdot \nabla \phi + \frac{\partial \phi}{\partial t} = 0$$

$$\therefore \frac{\partial \phi}{\partial t} = -F|\nabla \phi|$$

x

where F is the speed function normal to the curve

Level Set example: An expanding circle

Level Set representation of a circle:

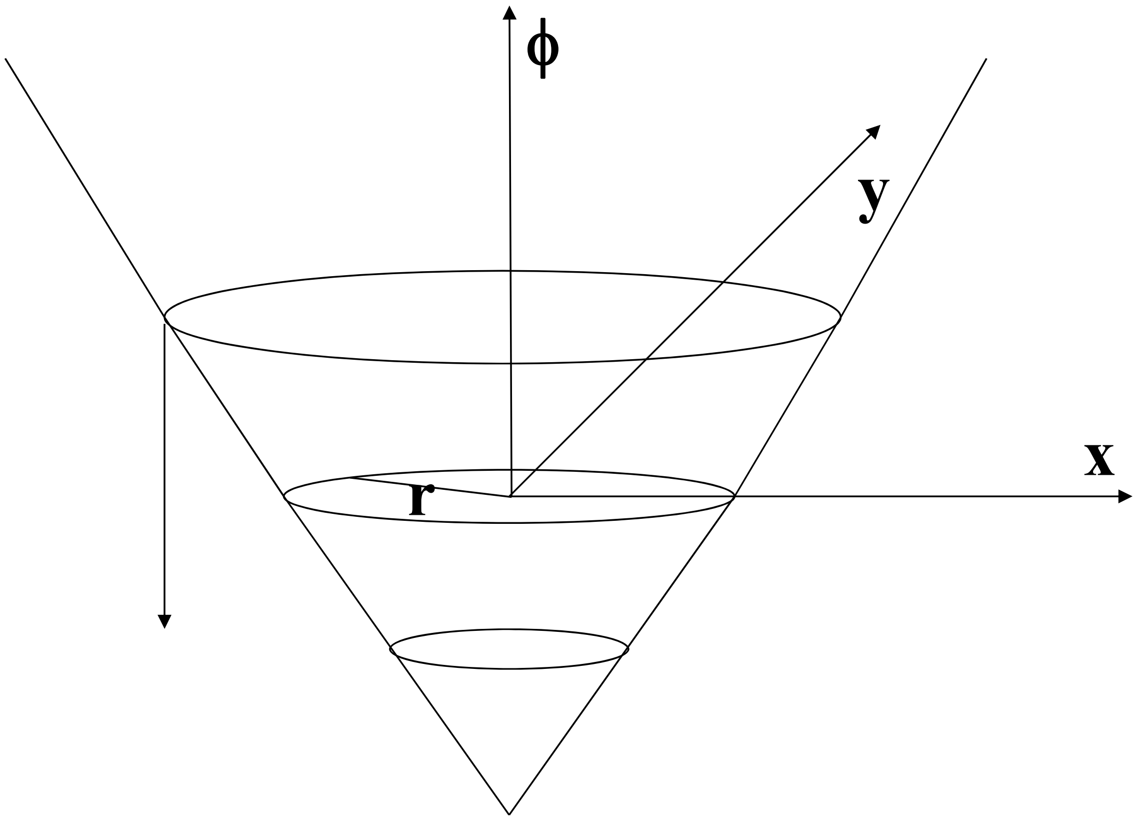
- Setting $F = 1$ causes the circle to expand uniformly

- **Observe that $\nabla\phi = 1$ almost everywhere (by choice of representation), so we obtain the level set evolution equation:**

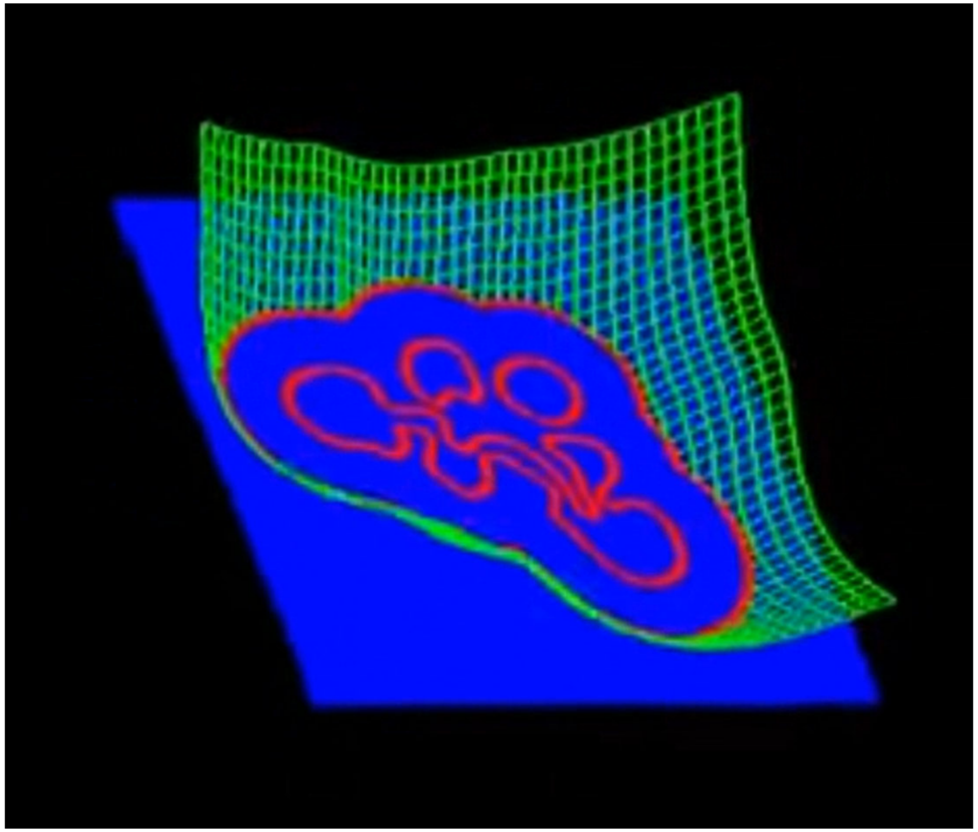
- **Explicit solution:**

which means that the circle has radius $r + t$ at time t, as expected!

Level Set example: An expanding circle



Another Example



Level Set Segmentation

- Since the choice of ϕ is somewhat arbitrary, we choose a signed distance function from the contour.
  - This distance function is negative inside the curve and positive outside.
  - A distance function is chosen because it has unit gradient almost everywhere and so is smooth.
- By choosing a suitable speed function *F*, we may segment an object in an image

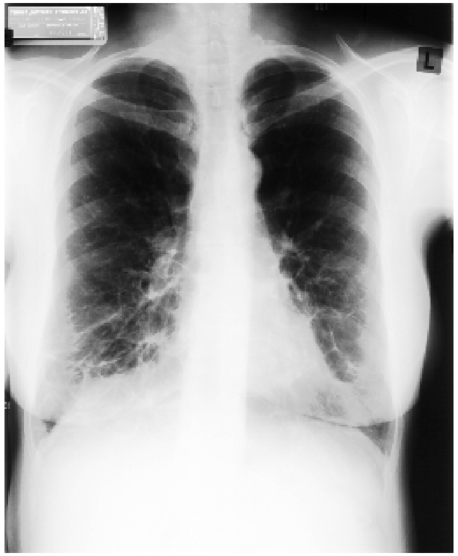
Level Set Segmentation

- The standard level set segmentation speed function is:

$$F = 1 - \varepsilon\kappa + \beta\left(\nabla\phi \cdot \nabla\left|\nabla I\right|\right)$$

- **The *1* causes the contour to inflate inside the object**
- **The *-εκ* (viscosity) term reduces the curvature of the contour**
- **The final term (edge attraction) pulls the contour to the edges**
- **Imagine this speed function as a balloon inflating inside the object. The balloon is held back by its edges, and where there are holes in the boundary it bulges but is halted by the viscosity *ε*.**

Level Set Segmentation Example



Lung x-ray

Viscosity 5

Viscosity 2

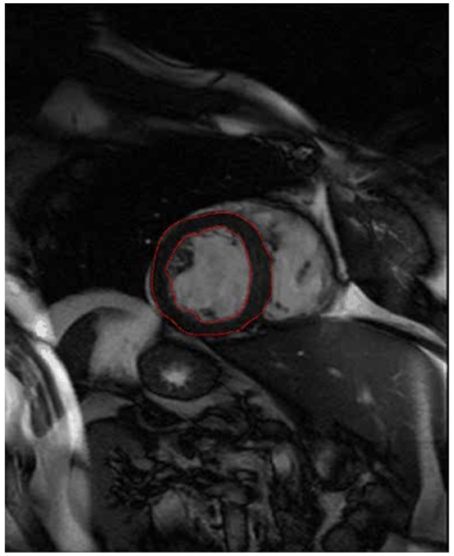
Viscosity 0.5

Level Sets – further examples

Real-time segmentation:

[Sethian Segmentation.htm]

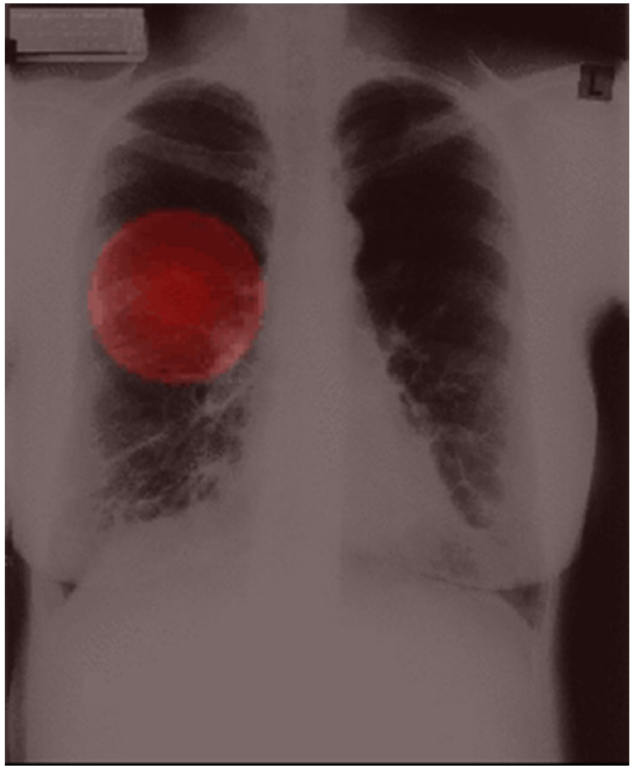
Combined segmentation and tracking:



The problem of structure:

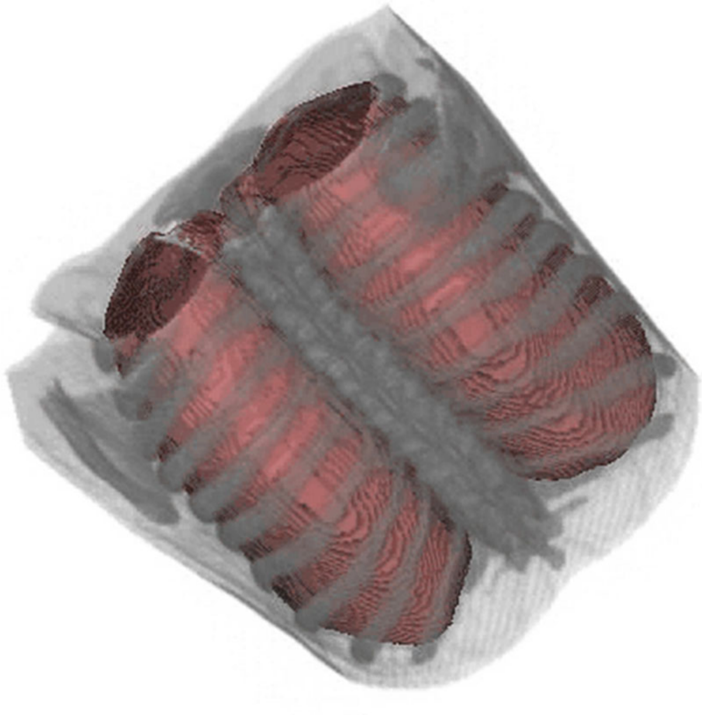
Taken from http://math.berkeley.edu/~sethian/

Our Work: Globally Minimal Surfaces

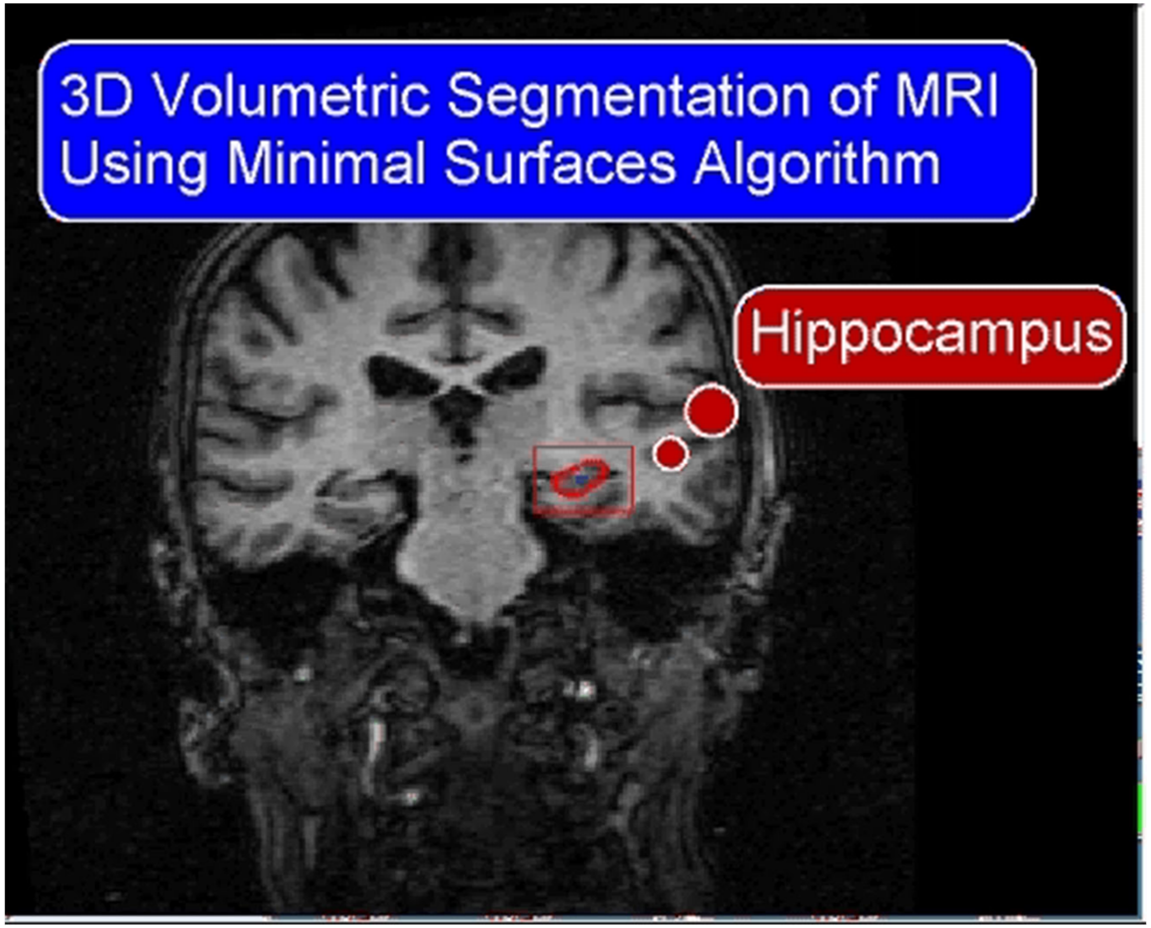


Appleton et al, PAMI

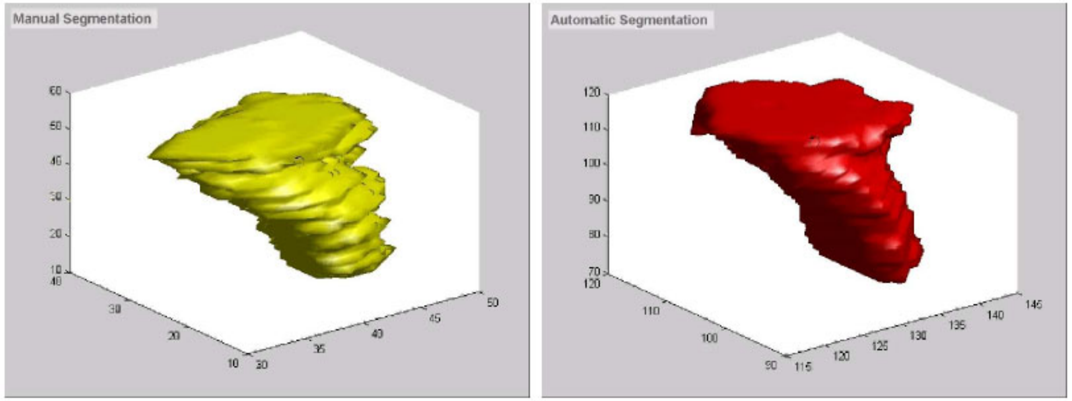
Lung Segmentation



Hippocampus Segmentation



Hippocampus Segmentation



(a) (b)

Fig. 2.23. Comparison of manual and automatic segmentation of the hippocampus in the human brain. Image (a) is a manual segmentation by a clinician which required about 2 hours of labelling and (b) is a fully-automated segmentation via GMS using multiple sources and sinks positioned by cross-validated training on labelled images which required just 2 minutes of computation (from [41]).

Conclusion

What we've covered:

- Dynamic Programming
- Shortest Path Algorithms: Dijkstra and Fast Marching
- Evolving Contours and Surfaces: Level Sets
- Applications

Questions?

?