

# Graph Theory: Week 9

## Message passing in graphs

John Quinn

16<sup>th</sup> October, 2007

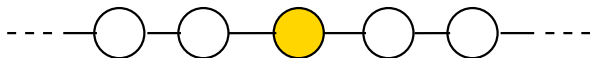
# Counting people in a queue

- ▶ While waiting in a huge queue at Stanbic bank, you might want to try and calculate how many people are in the queue (it goes outside the door and round the corner, so it is not possible to count everyone by sight).

Start of queue

You

End of queue



- ▶ A simple way to do this is to announce with a loudspeaker that everyone in the queue is to shout out their name. You then note down all the responses and count how many there are.
- ▶ However, this method is not very efficient or reliable—you have a lot of communication and processing to do.

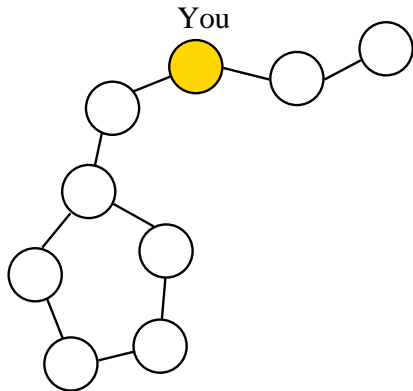
# A solution using message passing

The following method is more efficient:

1. If a person is at either end of a queue, they send the message “1” to their neighbour.
2. If a person has received a number from his neighbour, he adds one to that number and passes that message on to the neighbour on the other side.
3. When anyone has received messages from both their neighbours, they can calculate how many people there are in the queue by adding both messages and then adding one (for themselves).

# Cyclic graphs

The solution does not apply when there are cycles. Consider trying to count people standing in the following arrangement:



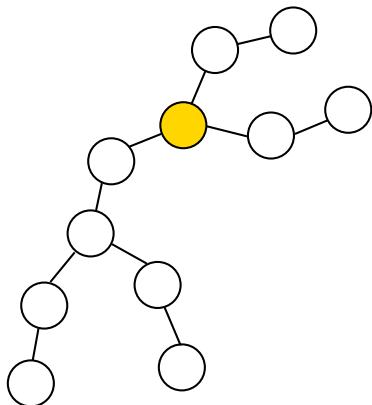
There are two problems: a vertex with degree three (for which we have no rule defined), and the presence of the cycle.

# Separable problems

- ▶ The problems which we can solve using message-passing algorithms have a property of *separability*.
- ▶ The simplest example is for counting the number of people in the queue. The number of people in front is separate from the number of people behind; when there is a cycle the separability property no longer holds.
- ▶ Trees also have the separability property, because every vertex is a cut vertex.

## Counting vertices in a tree

The people-counting problem should therefore be solvable if everyone is arranged in a tree structure:



How would the algorithm need to be modified in order to work for trees?

# Counting vertices in a tree with message passing

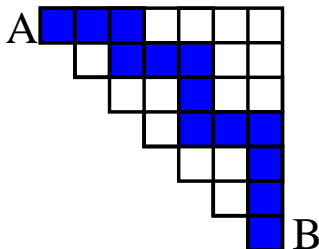
Each person should follow these rules:

1. Count your neighbours,  $d$  (the degree of your vertex).
2. Keep count of how many messages you have received. When you have received  $d - 1$  messages, add them together, then add 1 and pass this message to the remaining neighbour (the one who has not sent you a message).
3. If you have received  $d$  messages:
  - 3.1 The sum of the messages plus 1 is the answer.
  - 3.2 Send to neighbour  $n$  the total answer minus  $v_n$  (the message from that neighbour).

Note that step 2 applies to people who are at the end of a line (vertices with degree 1).

# Counting paths through a grid

- ▶ Say that valid paths are made up of rightward and downward moves.



- ▶ How many possible paths are there between A and B?

# Counting paths with messages

- ▶ The problem is separable because for every vertex there are a number of paths which could have lead there from A, and a certain number of paths which lead from there to B.
- ▶ To pass messages, we therefore start with a “1” at point A. For each vertex, we add the messages from the neighbour to the left and from the neighbour above.

# Frequency of nodes in paths

- ▶ For a particular cell, how many of the possible paths pass through it?
- ▶ We can work this out by passing messages backwards from B to A.

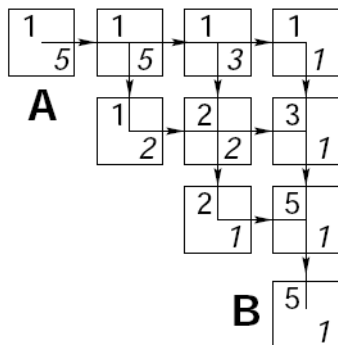


Fig taken from MacKay (2003)

# Why a backwards pass?

- ▶ Think of the messages as probabilities (or proportional to probabilities as they are not normalised).
- ▶ The forward messages from  $A$  are proportional to the probability that a path goes through a particular cell  $C$  if the path  $p$  started from  $A$ ,  $P(C \in p | A \in p)$ .
- ▶ Similarly, the backwards messages are proportional to  $P(C \in p | B \in p)$ .
- ▶ What we require is  $P(C \in p | A \in p, B \in p)$ .
- ▶ We therefore multiply the two messages together to obtain the answer.

# Probabilities of cells being on a path

Calculated for a  $41 \times 41$  grid: (a) shows the probability of a valid path passing through each cell, (b) shows a randomly sampled path.

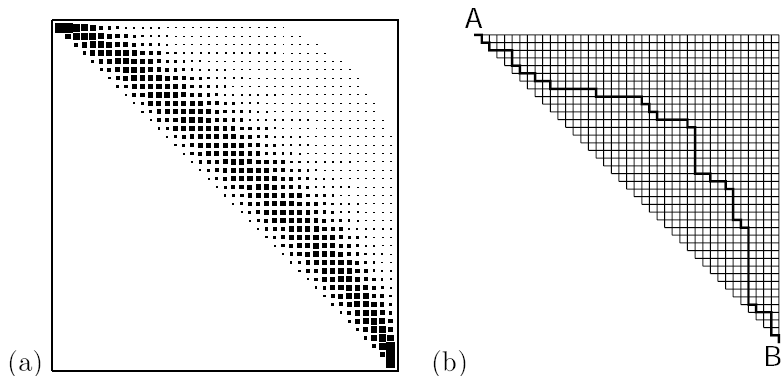


Fig taken from MacKay (2003)

## Exercise: how can a path be randomly sampled?

- ▶ How can we find examples of random paths, using “fair sampling” (that is, each of the possible paths are equally likely)?
- ▶ How about if we start at A, and every time there is a choice, we flip a coin to determine whether to go right or down?
- ▶ This scheme find samples, but are they fair? Hint: think of the probability of a particular path.
- ▶ How about if we take a path with probability determined by the backwards messages?

## Exercise: how can a path be randomly sampled?

- ▶ How can we find examples of random paths, using “fair sampling” (that is, each of the possible paths are equally likely)?
- ▶ How about if we start at A, and every time there is a choice, we flip a coin to determine whether to go right or down?
- ▶ This scheme find samples, but are they fair? Hint: think of the probability of a particular path.
- ▶ How about if we take a path with probability determined by the backwards messages?

## Exercise: how can a path be randomly sampled?

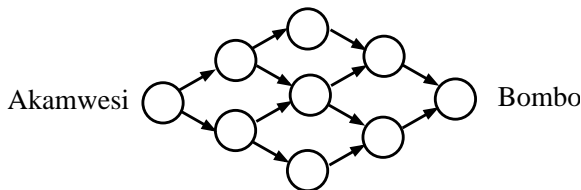
- ▶ How can we find examples of random paths, using “fair sampling” (that is, each of the possible paths are equally likely)?
- ▶ How about if we start at A, and every time there is a choice, we flip a coin to determine whether to go right or down?
- ▶ This scheme find samples, but are they fair? Hint: think of the probability of a particular path.
- ▶ How about if we take a path with probability determined by the backwards messages?

## Exercise: how can a path be randomly sampled?

- ▶ How can we find examples of random paths, using “fair sampling” (that is, each of the possible paths are equally likely)?
- ▶ How about if we start at A, and every time there is a choice, we flip a coin to determine whether to go right or down?
- ▶ This scheme find samples, but are they fair? Hint: think of the probability of a particular path.
- ▶ How about if we take a path with probability determined by the backwards messages?

# Shortest distance in a directed network

- ▶ In the first lecture, we looked at Dijkstra's algorithm for finding the shortest distance between two points in an undirected graph.



- ▶ In a weighted directed graph, the shortest distance can be found with a message-passing scheme called the *Viterbi algorithm* (also known as the *min-sum algorithm*).
- ▶ Because of the directed connections, the problem is separable as we can consider paths among the predecessors of a vertex and among the descendants.

# Viterbi (min-cut) algorithm

1. From starting point, set total current distance to 0.
2. Pass a message to each descendant with current total distance plus distance to descendant.
3. When a vertex has received messages from all predecessors, choose the smallest and ignore the rest—set this to be the total current distance. Go to (2).

## Refs for this week

Information Theory, Inference and Learning Algorithms, David MacKay, Chapter 16 (available online).