3F4: Data Transmission

Handout 10: Convolutional Codes

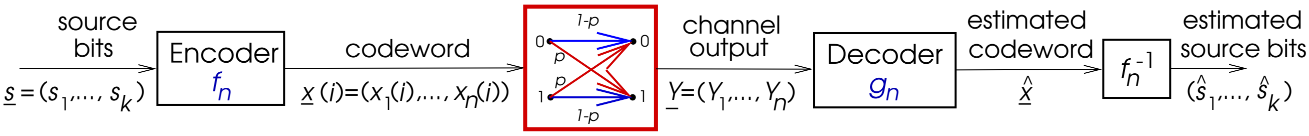
Ioannis Kontoyiannis
[based on notes by Ramji Venkataramanan]

Signal Processing and Communications Lab
Department of Engineering
i.kontoyiannis@eng.cam.ac.uk

Lent Term 2019

Block codes for the BSC(p)

- **Codebook** B_n consists of $M = |B_n| = 2^k$
  codewords $\underline{x}(i) = (x_1(i), \ldots, x_n(i)), \quad i = 1, 2, \ldots, M$
- **Size** of the code is $M = 2^k$ = size of codebook
- **Encoder** $f_n : \{0,1\}^k \to B_n$ maps source strings to codewords
- **Rate** of the code is $R_n = k/n$ bits/transmission
- **Channel output** string $\underline{Y} = (Y_1, \ldots, Y_n) \in A_Y^n$
- **Decoder** $g_n : A_Y^n \to B_n$



- **Idea**: Since the BSC(p) will flip $\approx np$ of the bits of $\underline{x}$
  add redundancy to $\underline{x}$ so that the channel errors
  can likely be corrected!

Example: The redundancy of natural language

Plain text message

```
This is a very simple test sentence, used in class
to illustrate the effects of random noise
```

In binary ASCII

```
10101001101000110100111100110100000110100111100110100000110000101000001110110110010111100101111100
10100000011100111101001110110111100001101100110010101000001110100110010111100111110100010000011100
11110010111011101110100110010111011101100011110010101011000100000111010111100111100101110010001001
00011010011101110010000011000111101100110000111100111110011010000001110100110111101000000110100111 0
11001101100111010111100111110100111001011000011110100110010101000001110100110100011001010100000011
00101110011011001101100101110001111101001110011010000001101111110011001000000111001011000011101110 1
1001001101111111011010100000110111011011111101001111100111100101
```

Reconstructed after going through a BSC(p)

p`=0.05` `This is a very 3i-pl% tes4 s%n4ence <usgt i| cmasS`
`toAilluCdr!te uèm eFdeëtc Of vandom nois%`

p`=0.01` `This is a very$sim`le test sentence,(used in clasó`
`to illuótrAte the effects of rándom noise`

Good codes and capacity

- **Error probability** (maximal) of an (n, k) code code is

$$P_e^{(n)} = \max_{1 \le i \le M} \Pr(g_n(\underline{Y}) \neq \underline{x}(i) \,|\, \underline{x}(i) \text{ was sent})$$

- Rate $R = k/n$ is **achievable** if there are (n, k) codes with $R_n \to R$ bits/trans, and $P_e^{(n)} \to 0$, as $n \to \infty$
- **Capacity** C = the largest achievable rate
- For the BSC: $C = 1 - H_2(p)$ bits/trans
  where $H_2(p) = -p \log_2 p - (1-p) \log_2 (1-p)$

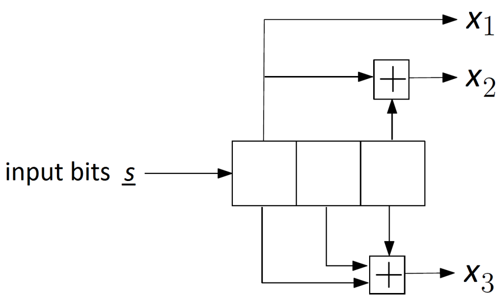
Good codes have:

- High rate $R = k/n \approx C$
- Low probability of error $P_e^{(n)} \approx 0$
- **Computationally efficient encoding and decoding**

- A code is **linear** if, whenever $\underline{s}$ gets encoded as $\underline{x}$ and $\underline{s}'$ gets encoded as $\underline{x}'$, the sum $\underline{s} + \underline{s}'$ gets encoded as $\underline{x} + \underline{x}'$

Convolutional codes

In convolutional codes, a stream of input bits is transformed into a stream of code bits using a shift register (filter)



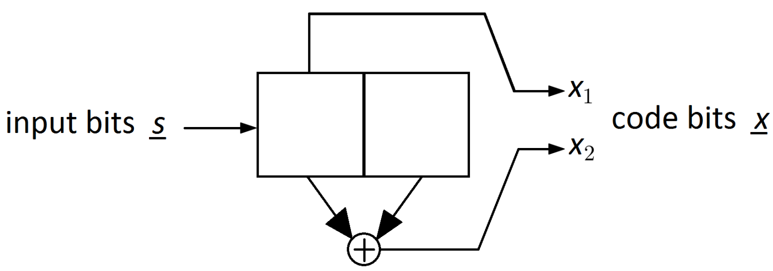
- Here, for every $k = 1$ data bit, we have $n = 3$ code bits
- Assuming the initial state of the shift register is (0 0 0) the code bits corresponding to the input $\underline{s} = 1010$ are

$$(111\ 001\ 100\ 001)$$

- Dependence between code bits is created via the shift register

Parameters of a convolutional code

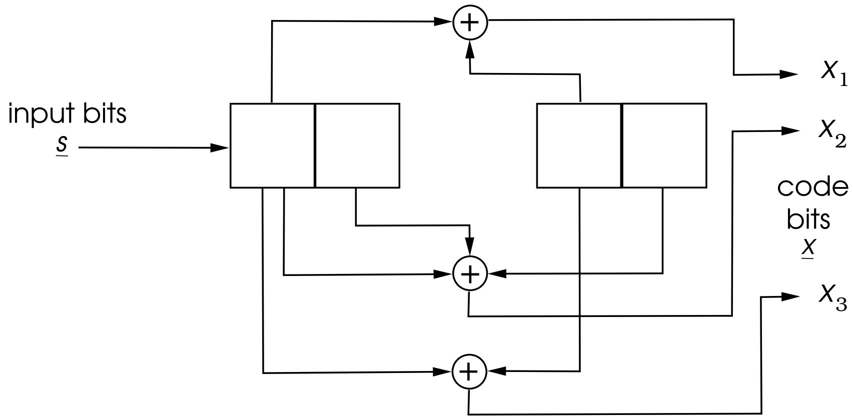
A rate $\frac{1}{2}$ convolutional code with $k = 1$ input bit and an $S = 2$-stage shift register



In a general convolutional code, at each *time instant*: k input bits are fed into a shift register with S stages, and its contents are used to produce n code bits, for a rate of $R = k/n$

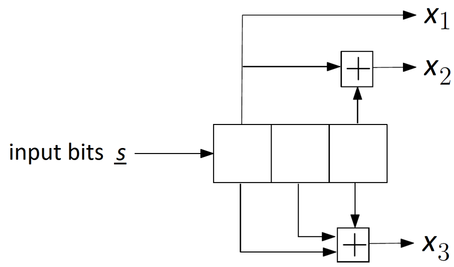
Another example of a convolutional code

A rate $\frac{2}{3}$ convolutional code with $k = 2$ input bits and an $S = 2$-stage shift register



- We often, but not always, consider $k = 1$:
  one bit shifted into the shift register at a time
- We always assume that the shift register is initially all zeros
- Although a convolutional code can be thought of as a block code, there is no fixed blocklength for the input sequence or the coded sequence

Generators of a convolutional code



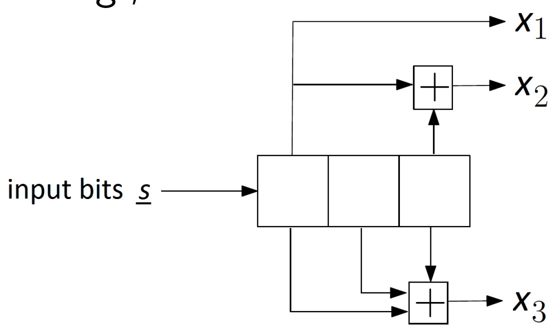
- If $\underline{x}, \underline{x}'$ are the code sequences corresponding to source sequences $\underline{s}, \underline{s}'$, respectively, the code sequence for $(\underline{s} + \underline{s}')$ is $(\underline{x} + \underline{x}')$
- Thus convolutional codes are *linear codes*
- Similarly to a generator matrix, the generation of the code bits can be described by *n generators*, which indicate which shift register bits are added to generate each code bit
- In the example above, these generators are

$$g_1 = (1\,0\,0), \quad g_2 = (1\,0\,1), \quad g_3 = (1\,1\,1)$$

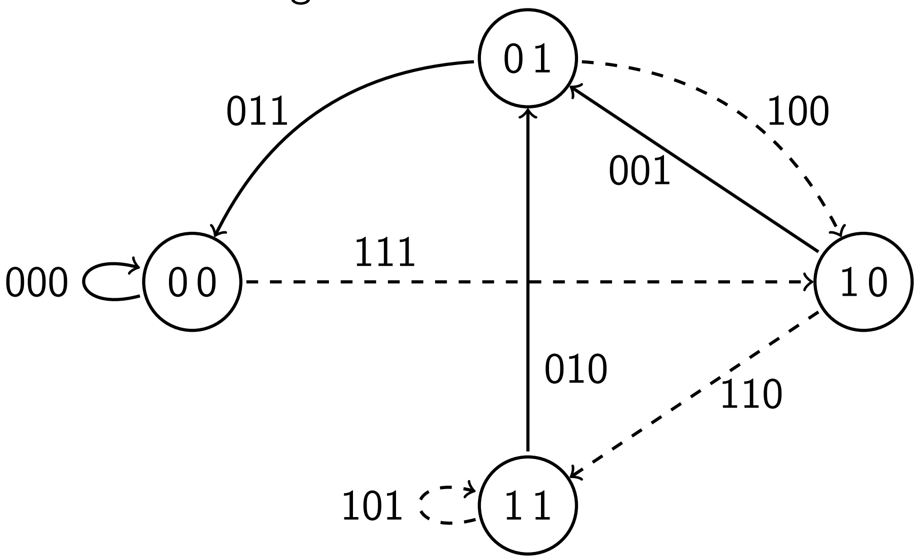
- Specifying the generators is equivalent to describing the convolutional code via a block diagram

Finite-state machine description

Convolutional codes can also be represented via *state diagrams* corresponding to *finite-state machines*. E.g., the code:



has the state diagram:



The state diagram

Suppose we have a code with S shift register stages and assuming $k = 1$ bit is fed into the register at each time:

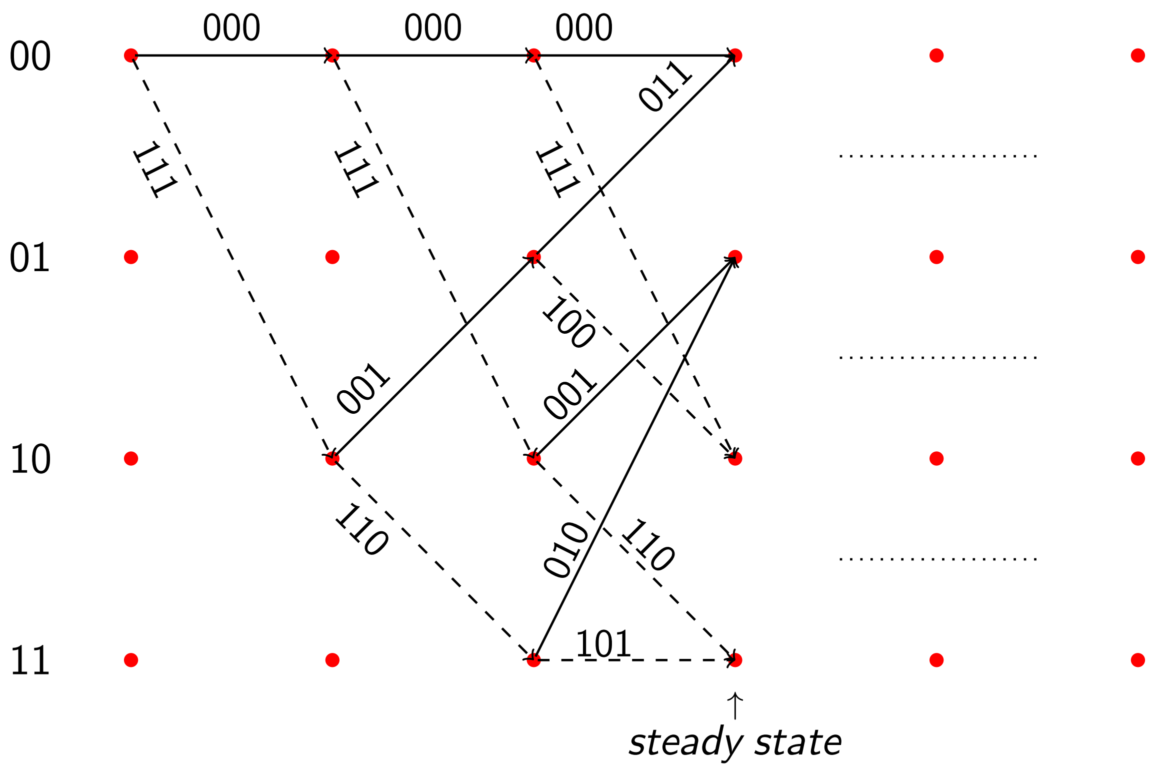
- The **states**, represented as the nodes of the state diagram, are the contents of the first $S - 1$ stages of each shift register; so there are 2^{S-1} states
- Two possible **transitions** out of each state, represented as directed edges, one for each possible input bit 0 or 1
- In the example above, solid lines correspond to input bit 0 and dashed lines to input 1
- Each edge has a **label**, which is the output corresponding to that transition

Simple exercise: Draw the state diagram for the code on slide 6.
Important exercise: Draw the (four-state) state diagram for the code on slide 7. Label each branch with the input and output bits corresponding to the transition along that branch.

N.B. The most important and useful representation of a convolutional code is via a *trellis diagram* . . .

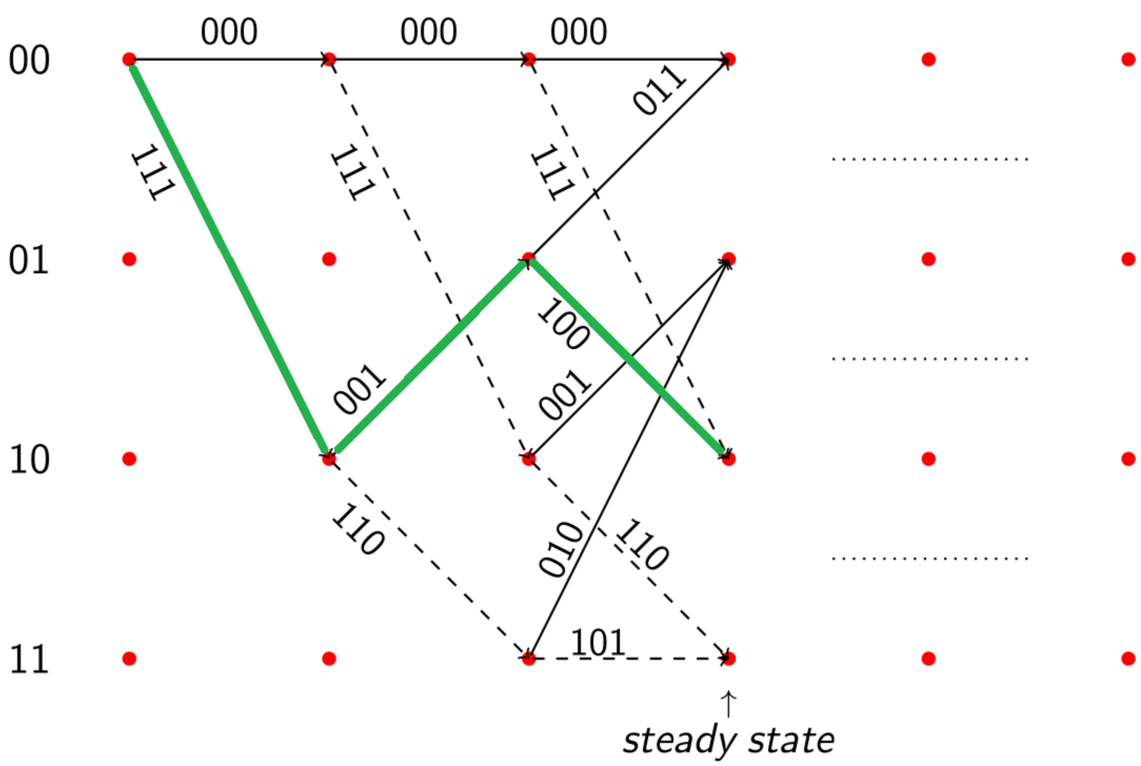
The trellis representation

For the earlier example of an $(n = 3, k = 1, S = 3)$ code:

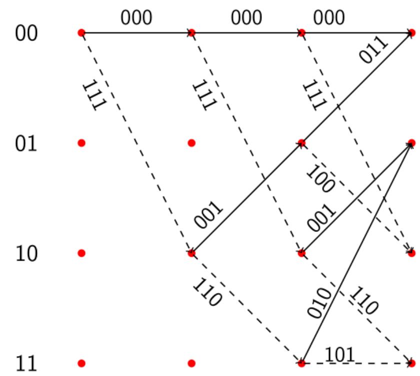


The trellis representation

E.g.: The path traced by the input string $\underline{s} = 101$ producing the code string $\underline{x} = 111\ 001\ 100$



The trellis representation

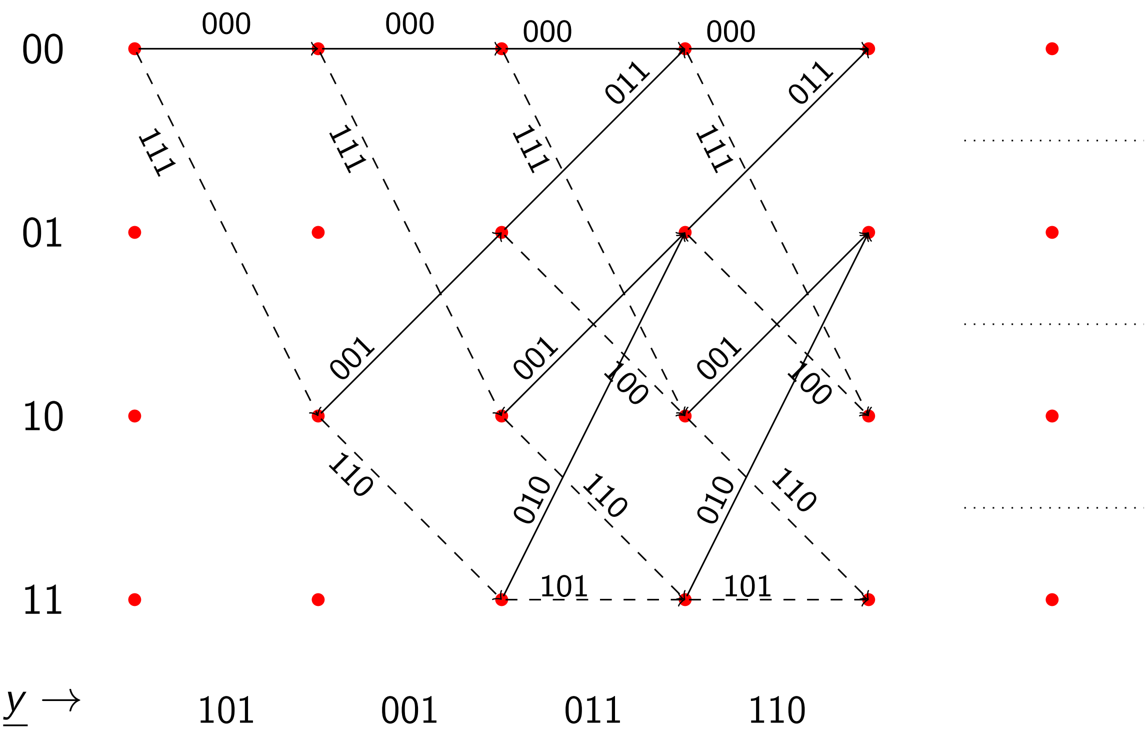


- Number of states in each step of trellis $= 2^{S-1} =$ number of possibilities for first $(S-1)$ stages of shift register
- Each sequence of input bits determines a path in the trellis
- The path gives the sequence of states and code bits corresponding to the input (source or information) bits

  Ex. What is the path corresponding to the $k = 5$ source bits 01010?
  What is the corresponding codeword $\underline{x}$ of length $n = 15$?
- The finite-state machine and the trellis are equivalent representations of a convolutional code except that the former does not have the notion of *time*
- The trellis representation is very helpful in visualising the *decoding* of a convolutional code

Decoding convolutional codes

Recall: Optimal decoding is minimum distance decoding
E.g.: If we receive $\underline{y} = 101\ 001\ 011\ 111$, we need to find a path in the trellis which gives a code sequence $\hat{\underline{x}}$ minimising $d(\underline{y}, \hat{\underline{x}})$



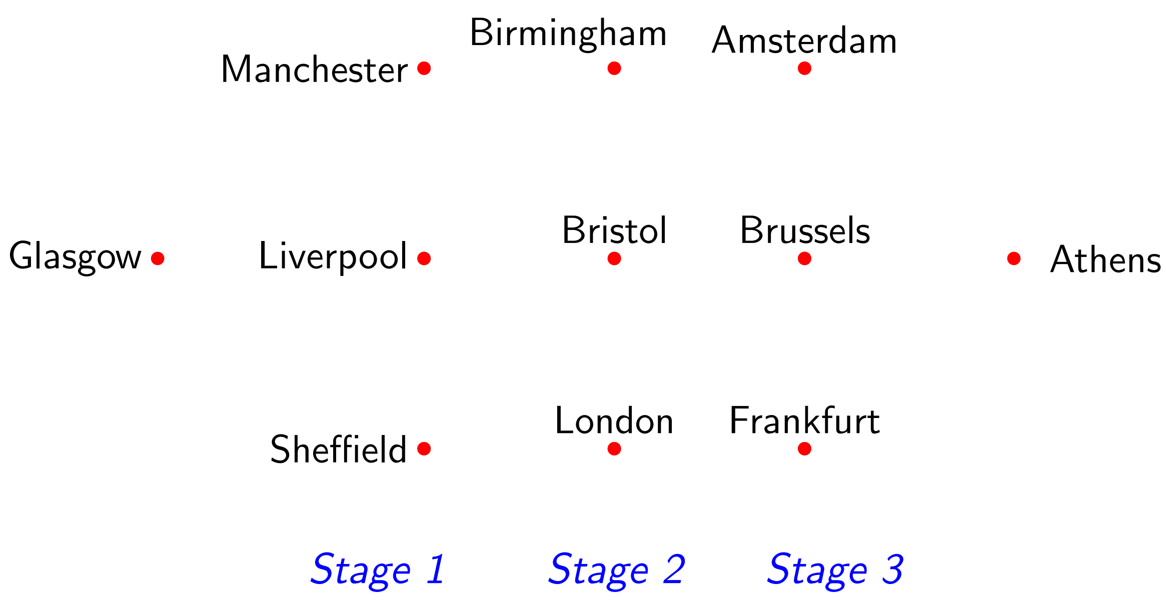
Decoding convolutional codes

- For k input bits, so that $\underline{y}$ has length $n = 3k$ the number of possible trellis paths is 2^k which grows exponentially with k
- Finding the min-distance path for given $\underline{y}$ via exhaustive search, is impossibly complex

There is a simple algorithm to find the minimum-distance path called the **Viterbi algorithm**

It is an instance of a general idea called **dynamic programming**

To understand the idea, let us consider an example of finding the shortest path between two cities, say Glasgow and Athens . . .



- *Problem*: Given the distances between all pairs of cities find the min-distance path Glasgow $\rightarrow$ Athens, subject to:
- Your path should pass through exactly one city in each stage
- E.g.: Glasgow–Manchester–London–Amsterdam–Athens is a valid path

If s_3 is a city in Stage 3:

A) $\text{min-dist}(\text{Gla} - \text{Ath}) = \min_{s_3}\{\text{min-dist}(\text{Gla} - s_3) + \text{d}(s_3 \to \text{Ath})\}$

Thus we now have to find the min-distance from Gla
to each of the stage 3 cities. For each fixed s_3:

B) $\text{min-dist}(\text{Gla} - s_3) = \min_{s_2}\{\text{min-dist}(\text{Gla} - s_2) + \text{d}(s_2 \to s_3)\}$

Similarly, for each s_2:

C) $\text{min-dist}(\text{Gla} - s_2) = \min_{s_1}\{\text{min-dist}(\text{Gla} - s_1) + \text{d}(s_1 \to s_2)\}$

Finally, for each stage 1 city s_1, $\text{min-dist}(\text{Gla} - s_1)$ is known:

1. Use this to solve (C) for each s_2
2. Then use the solution to solve (B) for each s_3
3. Then use solution of (B) to solve (A)

This algorithm, called **dynamic programming**,
is much more efficient than an exhaustive search among all paths:
Complexity only grows linearly with the number of stages

Next time we will apply it to trellis decoding ...