University of Sheffield, Department of Computer Science

COM1005: Machines and Intelligence

Semester 2: AI Techniques

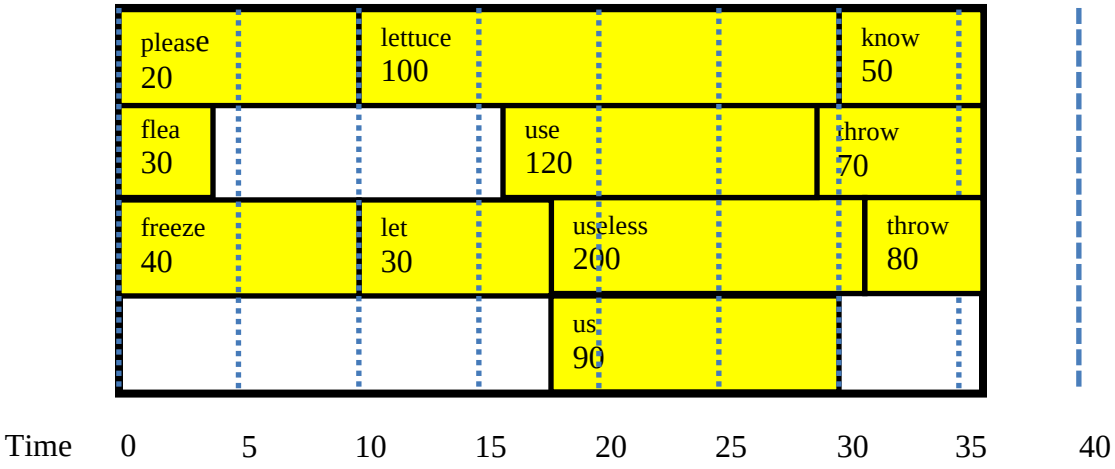
Assignment 1: Searching a Word Lattice

This assignment counts for 12.5% of the assessment for COM1005

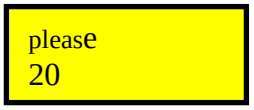
1. Introduction

Word Lattices

Here is an example of a **word lattice**:



A word lattice might be produced by an Automatic Speech Recogniser as the output of its acoustic analysis stage.

Each yellow box, e.g 

is a **word hypothesis,** which extends over a particular time segment: so the top-left box means that the recogniser thinks that the word 'please' might have been spoken between times 0 and 10. Each word hypothesis has a numeric **word cost** (20 in this case), which is a measure of how confident the recogniser is in the hypothesis. Low numbers are better: an ideal match would score 0.

The problem is to **find the best path through a word lattice**, where a path is a sequence of word hypotheses which spans the entire lattice. The word hypotheses must line up in

time, with no gaps. Valid paths in this example are *'please lettuce know'*, '*please let us know*' and *'freeze let useless throw'*.

Language Models

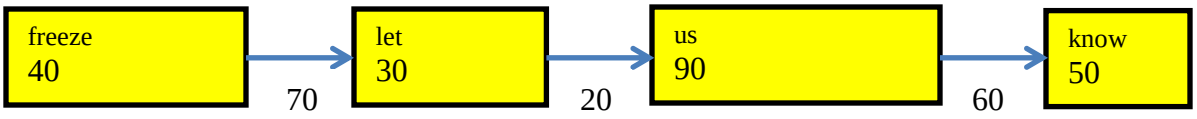
A language model enables the ASR to take word sequence likelihoods into account. For instance, *'freeze let useless throw'* is a pretty unlikely thing to say: '*please let us know*' is more plausible.

We'll use a **bigram LM**, which tells us how likely it is that one word will follow another. For this example we might have

w1\w2	please	lettuce	know	flea	use	throw	freeze	let	useless	us
please	500	100	80	120	20	40	60	20	120	30
lettuce	50	500	100	120	90	60	90	70	110	120
know	50	80	500	90	100	80	70	80	100	40
flea	60	120	110	500	100	90	80	60	100	100
use	30	70	80	90	500	50	100	70	90	40
throw	50	60	90	80	80	500	120	90	100	40
freeze	60	80	120	100	110	100	500	70	120	70
let	70	50	70	60	40	80	90	500	90	20
useless	100	90	80	60	80	40	90	80	500	90
us	20	100	60	100	70	60	80	40	90	500

The numbers here are again to be treated as costs – the smaller the better. We'll call them **transition costs**. So for example the transition cost associated with *'please lettuce'* is 100 and with '*lettuce please'* is 50. The diagonal corresponds to the same word twice in succession, which is very unlikely.

The total cost of a path through a word lattice is obtained by summing the word costs and transition costs along it: this path



Has total cost of 360.

2. What you must do

The aim of this assignment is to use the state-space search framework developed in the lectures to solve word lattice search problems.

1. This is a variable cost problem, as discussed in section 2.12 of the notes.
2. The version of the java search engine to use is in the **java\search3** folder, and is copied into the **WordLatticeAssignment19** folder you can download from the folder for this assignment on Mole: see § 3 below.

3. You must write classes **LatticeSearch** and **LatticeState**, following the notes in section 2.9. You will need a class **RunLatticeSearch** to run tests. A version of that which sets up the word lattice and language model above is included in **WordLatticeAssignment19.**
4. By experimenting with a number of word lattices and language models, show that branch-and-bound search will always find the best path through a word lattice, but depth-first and breadth-first searches do not. An alternative language model is provided .. see Section 3 below.
5. Investigate the word lattice search domain further. What you do is up to you but you could, for instance, consider how the A* algorithm might be applied to this domain or explore the relationship between the complexity of the lattice and the efficiency of the search.

3. Software to help you

In the folder **WordLatticeAssignment19** on MOLE is Java code for all the classes you will need. There are lattice classes which are similar to those provided for map traversal:

- **WordLattice.java** is the class of word lattices. An instance of **WordLattice** holds an **ArrayList** of word hypotheses (**WordH's**). Methods are
    - **latticeFromFile(String fname)** reads a lattice from a test file fname. The example above is in **latt1.txt**
    - **wordsAtT(t)** finds all the word hypotheses starting at a given time **t**, returning an **ArrayList** of **WordH**.
    - **getEndTime()** returns the lattice end time.

- **WordH.java** is the class of word hypotheses. A word hypothesis consists of a word (String), start time, end time and word cost. The constructor is supplied with these and there are corresponding accessors and a **toString** method.

- **LM.java** implements bigram language models. It takes the vocabulary as a **String** array and the cost matrix as a 2D array of **int**, as illustrated above. The method you need is

    - **getCost(String w1, String w2)** returns the cost of word **w2** following word **w1.** It returns 0 if **w1** is ***start***: make this the 'word' in the initial LatticeState.
    - The language model illustrated above is created in the given **RunLatticeSearch.** An alternative, with a larger vocabulary, is provided in the assignment folder on MOLE as **TVLM.txt.**

4. Mark Scheme

Credit will be assigned as follows:

Implementing LatticeSearch	20%
Implementing LatticeState	40%
Experiments with search strategies (§2.4 above)	25%
Further experiments (§2.5)	15%

For implementation, 60% of the credit is for the quality of the code, 20% is for testing and 20% is for commenting.
Testing should cover all the logically different cases, e.g. identical and different states for **sameState**. A good way of reporting test results is in a table:

Method	**Testing for**	**Call**	**Returns**
sameState	Identical states		true
	Different states		false
goalP	A goal state		true
	A non-goal state		false

In this the 'Call' column shows what arguments you supplied in the test

5. How to hand in

Hand in by MOLE. You should hand in **a single zip archive** containing:

- Your java classes.
- A document (word or pdf) summarising your testing and experimental results. A format for your report is on MOLE.

Deadline: Wednesday of Week 7 (20th March) at midnight.