Lecture 15: MDP4

Victor R. Lesser
CMPSCI 683
Fall 2010

Today's Lecture

- Quick Review of First Part of Exam
- Hidden Markov Processes

General Comments

- *I will grade the exam based on 91 points total rather than 100 points. So many people were unprepared to answer question 1F. The grade on the answer to this question can be thought of as extra credit.*

Question 1A

Suppose you had two admissible A* heuristics (h1 and h2) for a specific problem application and there was respectively cost (c1 and c2) every time you applied the heuristic in a search. How would you go about deciding which heuristic to use for the entire class of problems?

*Run experiments on a number of comparison cases using each of the heuristics to get the average time for each search with different heuristics. The heuristic whose average search time over the set of examples is the lowest would be the one chosen. Another way to do think about it is get the average number of nodes expanded in each search. Then it would be E of h1 (average number of nodes expanded) *c1; and similarly for h2. The formula that gave the lowest value would determine what heuristic to choose. Obviously, if h2 dominates h1 and the cost of applying h1 (c2) has lower cost than c1, you would choose heuristic h2 for all problems and no experimentation is necessary. In my answer, I did not think of trying experimentally the case of max (h1,h2) which incurs the cost of applying both heuristics to each search node expanded. However, in some cases that could be the best choice but I suspect it is very rare. Further in considering which heuristic to choose for a class of problems, I was only considering which would lead to the smallest expected search cost for solving a problem. Another criterion could be to minimize the number of nodes expanded, this would lead to a slightly different reasoning about which heuristic to choose. Very few people got the idea that the number of nodes searched was key!*

Question 1B

What are the similarities and differences between Anytime A* and RTA*?

Both are doing an approximate search given a fixed amount of time that can be used. They both exploit an admissible and monotonically increasing h heuristic. However, their search strategies are very different. RTA* use a limited (based on a fixed horizon) depth-first search to get a better approximation of a node's f value to make a decision what operator to apply next. There is some interesting pruning going on in how the f value's at the horizon are backed-up – but that was not important for the answer. It then applies after each search the chosen operator in the real world and then repeats the procedure to choose the next operator after the move is completed. Anytime A* in contrast is doing a complete search trying to get an acceptable solution quickly and then over time improving the solution. When it is finally terminated either because of time limits or an optimal solution is found, the best (lowest cost) complete path/plan that has been encountered is chosen. I was surprised that people talked about Anytime A* in terms of a non-admissible heuristic, the h heuristic is in fact admissible and this is exploited in the pruning.*

Question 1C

What are the similarities and differences between SMA* and RBFS?

Both exploit the fact that f is monotonically increasing and there is a remembrance of the f values of previously encountered partial solutions to focus what node should be next (re)expanded; they also both are trying to reduce the amount of memory necessary for the search, and for that reason both may generate repeatedly the same node. In the case of SMA, it deletes nodes due to fixed memory limitations while RBFS may delete nodes because it keeps a very restricted open list based on a depth-first type of search. SMA* needs to have as much memory as the length of the optimal path otherwise it will be able to find this optimal path.*

Question 1D

Explain the common reason/principle for the use of the techniques of beam search in Genetic Algorithms and random restart in GSAT. Could you apply beam search to GSAT?

Both search techniques are trying to avoid getting stuck in local minima. The beam search has the potential advantage over random restart since it is able to constantly readjust what solutions are in the beam according to the quality and potentially the diversity of these solutions, and to be able to take parts of one solution and combine with parts of another solution in the beam to create a new solution. Maybe, the beam search could be applied to GSAT, it is interesting question of how often in GSAT do you need to do random restarts versus paying the overhead of concurrently processing multiple solutions (this is general problem with the beam search in contrast to random restart). To really exploit the beam search idea in GSAT, you would in some sense need to alter the basic search strategy of GSAT so that there was more than one next solution generated at each iteration. In this way, at each stage, the beam could be narrowed back to k width based on "fitness" of the current solutions in the beam. I am not sure whether this will be effective?

Question 1E

What would be your explanation for why GSAT does not exploit a specialized procedure to generate a "good" initial assignment for the truth values of the literals?

One possible explanation is that the cost of getting a good initial solution is quite expensive and it is better just searching based on a random initial solution and if that is not progressing well just try another random initial solution. It also may be that there are no general heuristics for a getting a good initial solution for an arbitrary problem though there may be good heuristics for a specific class of problems. – This was a think question and generally everyone got it right.

Question 1F

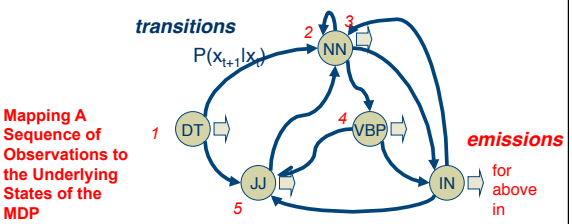
The HEARSAY-II speech understanding system as described in class is not based on the A* search because of the difficult of constructing an admissible and effective heuristic. However, it uses a termination procedure resembling Anytime A*. When Hearsay-II search found a complete solution that was above a certain rating, it could prune partial solutions (nodes) on the blackboard based on calculating a measure using all the words that had been constructed either through bottom-up or top-down processing at the point that a complete solution was generated. *Explain the basis for the pruning and also why this approach could potentially lead to incorrectly pruning a correct partial solution though we never saw an example of this.*

Based on an analysis of the word lattice, a measure can be constructed for the highest ranking word in each segment of the speech signal. This rating can be used to construct the "highest" possible score that a partial solution could get when it is completed. This is not totally accurate because in expanding a partial solution, it is possible that new higher rated words could be generated as a result the top-down word verification process. For this reason, the heuristic is not admissible and thus could lead to pruning of a partial solution that could have created a higher score than the current best solution.

Hidden Markov Models: (slides courtesy of Andrew McCallum)

Input: `the lead paint is unsafe`

TaggedOutput: `the/Det lead/N paint/N is/V unsafe/Adj`



Mapping A Sequence of Observations to the Underlying States of the MDP

Part-of-speech tags, examples

PART-OF-SPEECH	TAG	EXAMPLES
Adjective	JJ	happy, bad
Adjective, comparative	JJR	happier, worse
Adjective, cardinal number	CD	3, fifteen
Adverb	RB	often, particularly
Conjunction, coordination	CC	and, or
Conjunction, subordinating	IN	although, when
Determiner	DT	this, each, other, the, a, some
Determiner, postdeterminer	JJ	many, same
Noun	NN	aircraft, data
Noun, plural	NNS	women, books
Noun, proper, singular	NNP	London, Michael
Noun, proper, plural	NNPS	Australians, Methodists
Pronoun, personal	PRP	you, we, she, it
Pronoun, question	WP	who, whoever
Verb, base present form	VBP	take, live

VB - Verb, base form
VBZ - Verb, 3rd person singular present

Part-of-speech ambiguities

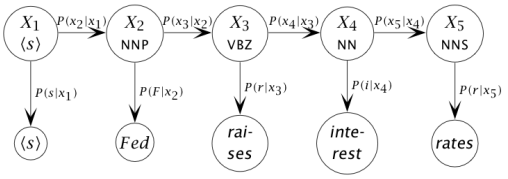
	VBZ	VB VBZ	VBZ			
NNP	NNS	NNS	NNS	CD	NN	
Fed	raises	interest	rates	0.5	%	in effort to control inflation

(Hidden) Markov Model

- View sequence of states(tags) as a Markov chain. Assumptions:
  - *Limited horizon* $P(x_{t+1}|x_1, ...x_t) = P(x_{t+1}|x_t)$
  - *Time invariant (stationary)* $P(x_{t+1}|x_t) = P(x_2|x_1)$
  - We assume that a word's tag only depends on the previous tag (limited horizon) and that his dependency does not change over time (time invariance)
  - A state (part of speech) generates an output (word). We assume it depends only on the state.

$$P(o_t|x_1, ...x_T, o_1, ...o_{t-1}) = P(o_t|x_t)$$

A Possible Path Thru the Network



- Top row is unobserved states, interpreted as POS tags
- Bottom row is observed output observations (words)
- *What is the likelihood of this path??*

Applications of HMMs

- NLP
  - Part-of-speech tagging
  - Word segmentation
  - Information extraction
  - Optical Character Recognition (OCR)
- Speech recognition
  - Modeling acoustics
- Computer Vision
  - gesture recognition
- Biology
  - Gene finding
  - Protein structure prediction
- Economics, Climatology, Communications, Robotics…

(One) Standard HMM formalism

- *(X, O, x_s, A, B) are all variables. Model μ = (A, B)*
- *X* is state sequence of length T; *O* is observation seq.
- x_s is a designated start state (with no incoming transitions).
- *A* is matrix of transition probabilities (each row is a conditional probability table (ConditionalProbablityTable)
- *B* is matrix of output probabilities (vertical CPTs)

$$P(X, O|\mu) = \prod_{t=1}^{T} a[x_t|x_{t-1}]\ b[o_t|x_t]$$

Prob of sequence X generating sequence O given the model μ

- HMM is a probabilistic (nondeterministic) finite state automaton, with probabilistic outputs (from vertices, not arcs, in the simple case)

Review of POMDP Model

Augmenting the completely observable MDP with the following elements:

- O – a finite set of observations
- P(o|s',a) – observation function: the probability that o is observed after taking action a resulting in a transition to state s'
- A discrete probability distribution over starting states (the initial belief state):

$$b_0 = \{b_0(0), b_0(1), \ldots, b_0(|S|-1)\}$$

Connection Between HMM and POMDP

- Similar set up but different problem being solved
  - Given an observation sequence, find the most likely hidden state sequence (tagging)
    - *No actions and rewards in HMM – thus you are not trying to find a optimal policy*
  - Length of sequence of observations in HMM is finite
  - Start out with well-defined initial state

Most likely hidden state sequence

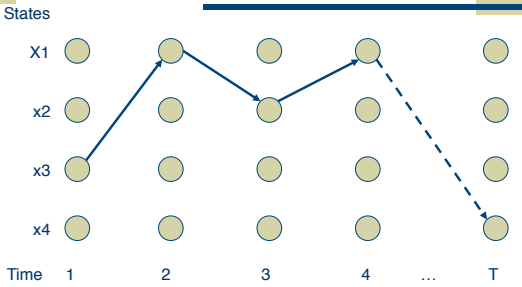
- Given O = (o_1,…,o_T) and model μ = (A,B)
- We want to find

$$\arg\max_X P(X|O,\mu) = \arg\max_X \frac{P(X,O|\mu)}{P(O|\mu)} = \arg\max_X P(X,O|\mu)$$

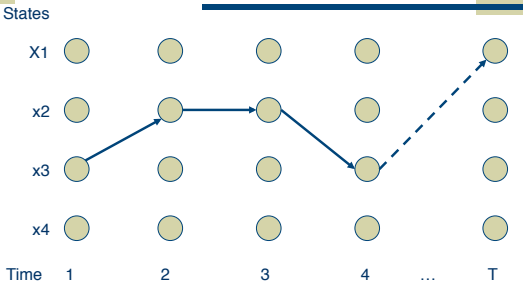
Constant with respect to X

- P(X,O| μ) = P(O|X, μ) P(X| μ)
- P(O|X, μ) = b[o_1|x_1] b[o_2|x_2] … b[o_T|x_T]
- P(X| μ) = a[x_2|x_1] a[x_3|x_2] … a[x_T|x_{T-1}]
- arg max$_X$ P(X,O| μ) = arg max x_1, x_2,… x_T P(X,O| μ)
- *Problem: arg max is exponential in sequence length*

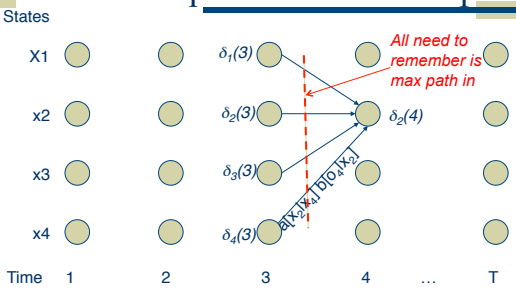
Representation for Paths: Trellis



Representation for Paths: Trellis



Exploit Markov Assumption to Cut Down Exponential Search Space



$\delta_i(t)$ = Probability of most likely path that ends at state *i* at time *t*.
Avoid repeated path computations by keep tracking of $\delta_i(t)$

Finding Probability of Most Likely Path using Dynamic Programming

- Efficient computation of max over all states
- Intuition: *Probability of the first t observations is the same for all possible t+1 length sequences*.
- Define forward score:

$$\delta_i(t) = \max_{x_1 \ldots x_{t-1}} P(o_1 o_2 \ldots o_t, x_1 \ldots x_{t-1}, x_t = i|\mu)$$

$$\delta_j(t+1) = \max_{i=1..N} \delta_i(t) a[x_j|x_i]\; b[o_{t+1}|x_j]$$

- Compute it recursively from the beginning
- (Then must remember best paths to get arg max.)

Finding the Most Likely State Path with the Viterbi Algorithm [Viterbi 1967]

- Used to efficiently find the state sequence that gives the highest probability to the observed outputs
- Maintains two dynamic programming tables:
  - The probability of the best path (max)

$$\delta_j(t+1) = \max_{i=1..N} \delta_i(t) a[x_j|x_i]\; b[o_{t+1}|x_j]$$

  - The state transitions of the best path (arg) – allows for backtracing

$$\psi_j(t+1) = \arg \max_{i=1..N} \delta_i(t) a[x_j|x_i]\; b[o_{t+1}|x_j]$$

- Note that this is different from finding the most likely tag for each time *t*! – you are instead looking for the optimal sequence of tags

Viterbi Recipe

- Initialization

$$\delta_j(0) = 1 \text{ if } x_j = x_s. \quad \delta_j(0) = 0 \text{ otherwise.}$$

- Induction

$$\delta_j(t+1) = \max_{i=1..N} \delta_i(t) a[x_j|x_i]\; b[o_{t+1}|x_j]$$

Store backtrace

$$\psi_j(t+1) = \arg \max_{i=1..N} \delta_i(t) a[x_j|x_i]\; b[o_{t+1}|x_j]$$

- Termination and path readout at terminal states

Most likely terminal state and Backtracing to get state at previous time it came from

$$\hat{x}_T = \arg \max_{i=1..N} \delta_i(T)$$

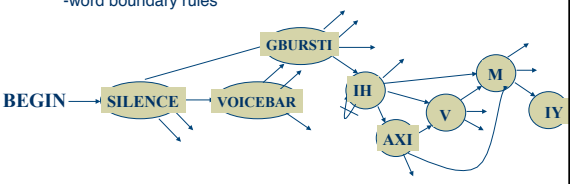
$$\hat{x}_t = \psi_{\hat{x}_{t+1}}(t+1)$$

Probability of entire best seq.

$$P(\hat{X}) = \max_{i=1..N} \delta_i(T)$$

HMM For Speech Understanding

Harpy network from
-lexical representations
-syntactic production rules
-word boundary rules



Acceptable Paths for "Give Me..."

How is the Network Constructed

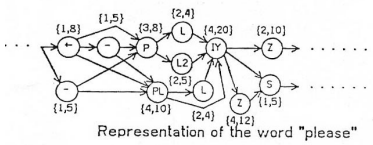
- Grammatical Knowledge
  - BNF Grammar that doesn't contain substrings of the form
    … A B C …
    … A^1 B C^1 …
    where B is a non-terminal, and B is recursive
- Lexical Knowledge
  - Finite-state "phoneme" network with duration information
- Contextual Knowledge
  - Juncture rules and juncture "phonemes"

HARPY

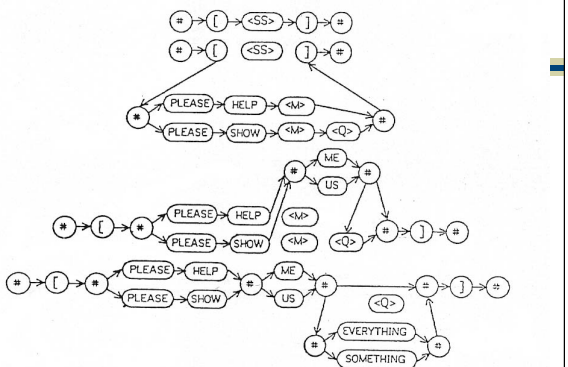
HMM Word Network

everything	(-,0) (EH,EH2) V R IY2 TH IH3 NX
help	(-,0) HH AA3 EL3 (← (-,0), -) P
me	(-,0) M IY
please	(← (-,0), -) (P (L,L2), PL (L,0)) IY (Z{4}, (Z,0) S)
show	(-,0) SH AA5 (OW,0)
something	(-,0) S AA M TH IH3 NX
us	(-,0) IH6 S (HH,0)
[	-
]	-

Pronunciation Dictionary for MQL

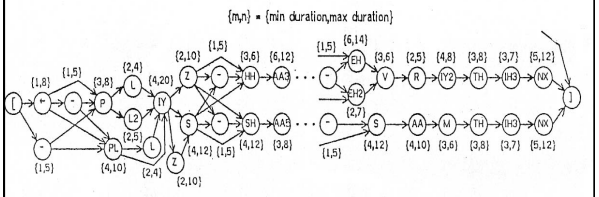


Representation of the word "please"



Generation of the Word Network From the Grammar Specification

HARPY Combined Network



Network After Replacing all Words With Phone Subnetworks
(Part of network not shown)

Heuristic Search in HARPY

Given a segmented acoustic signal with probabilities for each phone at each segment and a network how do we search it?

Each State in the Network

- Phoneme (from either phonetic dictionary or word juncture phonemes)
- Word
- Unique ID number
- Duration information
- A list of successor/previous states

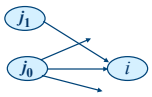
Network Search Algorithm

$$P_{i,t} = A_{i,t} \underset{j}{\text{Max}} (P_{j,t-1}\ T_{j,i})$$

Where

$T_{j,i}$ is the probability of transitioning from state j to i
- -based on network (0,1)
- -duration of being in state j

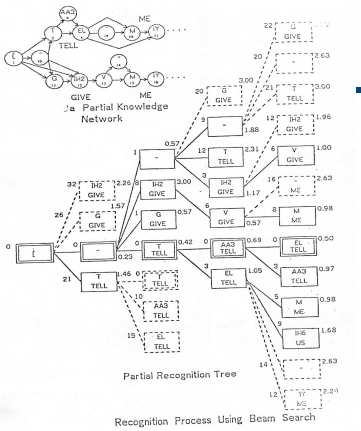
$A_{i,t}$ is the probability of being in state i given the acoustic event at time t.



BEAM Search

Heuristic version of Viterbi Search to reduce computation

- Compute probability for each Active State in segment i: keep pointer to state in segment (i-1) that is max transaction to each active state
- **Prune list of active states and normalize problem**
- Compute list of active states for segment i+1
- Repeat 1-3 until no more segments
- Backtrace



Next Lecture

- Introduction to Uncertainty