A* Algorithm

April 29, 2015

Group 13

Mahindar A Kota
Rohit Kumar
Pranay Dhondi

Objective

- To implement the A* algorithm general enough to be able to adapt to any search problem
- To get the baseline figure of nodes expanded when h=0
- To experimentally verify the intuition "better heuristic performs better"
- Measure node expansion when displaced tiles and manhattan distance heuristics are applied
- Prove/disprove the admissibility (i.e., h(n) ¡= h*(n), for all n) for a third heuristic
- Introduce a non-reachability test for start and goal node pair right at the start

A* Algorithm

- Create a search graph G, consisting solely of the start node start node S; put S on a list called on a list called OPEN.
- Create a list called CLOSED that is initially empty.
- Loop: if OPEN is empty, exit with failure. is empty, exit with failure.
- Select the first node on OPEN, remove from OPEN and put on CLOSED, call this node n
- if n is the goal node, exit with the solution obtained by tracing a path along the pointers from n to s in G. (pointers are established in step 7) (pointers are established in step 7).
- Expand node n, generating the set M of its successors that are not ancestors of n. Install these memes of M as successors of n in G.

A* Algorithm

- Establish a pointer to n from those members of M that were not already in G (i.e., not already on either OPEN or CLOSED) Add these members of). Add these members of M to OPEN. For each member of M that was already on OPEN or CLOSED, decide whether or not to redirect ts pointer to n. For each member of M already on CLOSED, decide for each of its descendents in G whether or not to redirect its pointer. whether or not to redirect its pointer.
- Reorder the list according to a function $f(n) = g(n) + h(n)$ where
  - g(n) is the least cost path to n from Start node found so far.
  - h(n) is the heuristic function which estimates cost of path from n to goal node
  - g(n) ¿= optimal cost path between n from Start node (g*(n))
  - In A* algorithm the following condition will be satisfied , h(n) ¡= optimal cost path between n and goal node (h*(n))
- Go Loop(Step 3)

Implementation

Data Structures for Open List and Closed List :

- Open List : Multimap on nodes sorted w.r.t f(n) values.
- Closed List: 'Set' data structure for storing nodes.
- A hashmap is defined between the nodes in Open List and their corresponding position in the multimap for the maintaining the Open List

Results - 8 puzzle problem

8 puzzle problem is a sliding puzzle that consists of a frame of numbered square tiles in random order with one tile missing

Aim of the problem is to obtain the goal state from tthe starting state by moving the blank tile

2	1	4
7	8	3
5	6	

s

1	6	7
4	3	2
5		8

n

1	2	3
4	5	6
7	8	

g

Baseline Figure for h = 0

- S = 4,1,2,0,8,7,6,3,5;
- G = 1,2,3,4,5,6,7,8,0;
- Optical cost found between S and G using A* algorithm is 17.
- No of nodes expanded in the process is 19963

Heuristic1 - Manhattan Distance

- Manhattan Distance : Manhattan distance, also known as L1-distance, of tile is defined as the sum of x and y distances of tiles of the current state from the corresponding counter parts in the goal states.
- If xi(s) and yi(s) are the x and y coordinates of tile i in state s, and if upper-line(xi) and upper-line (yi) are the x and y coordinates of tile i in the goal state, the heuristic is

$$h(s) = \sum_{i=1}^{8}(|x_i(s) - \overline{x}_i| + |y_i(s) - \overline{y}_i|).$$

- Optical cost found between S and G using A* algorithm is 17.
- No of nodes expanded in the process is 113
- We can observe the drastic decrease in number of nodes expanded

Heuristic2 - Misplaced Tiles

- Misplaced Tiles: Number of tiles that are not in the final position (not counting the blank)
- Optical cost found between S and G using A* algorithm is 17.
- No of nodes expanded in the process is 883
- We know that Manhattan heuristic is better than dispaced tiles. Hence we can observe that number of nodes expanded by Manhattan is less than displaced tiles

Table

Heuristic	Expanded Nodes	Optimal Path
h = 0	181321	30
Displaced Tiles	122962	30
Manhattan	5281	30
Manhattan with Linear Conflict	826	30
h = 0 (Bidirectional)	11956	30
Displaced Tiles (Bidirectional)	11192	30
Manhattan (Bidirectional)	1408	30
Manhattan with Linear Conflict (Bidirectional)	1022	32

Conclusion

- Better heuristic performs better
- For h ¿ h* we always do not get the optimised path, eventhough

Automatic Theorem Prover for Propositional Logic

April 29, 2015

Group 13

Mahindar A Kota
Rohit Kumar
Pranay Dhondi

Objective

- Aim of the assignment is to prove a theorem syntactically in propositional logic using only first principles or deduction theorem
- Main goal of the assignment is to observe the sensitivity of syntax and semantics seperation

First Principles

- Axioms

A1: $(A \rightarrow (B \rightarrow A))$

A2: $((A \rightarrow (B \rightarrow C)) \rightarrow ((A \rightarrow B) \rightarrow (A \rightarrow C)))$

A3 $(((A \rightarrow F) \rightarrow F) \rightarrow A)$

- Modus Ponens

Given $A \rightarrow B$ and

A

write B

Deduction Theorem

<u>Statement</u>

If

$A_1, A_2, A_3 \ldots\ldots\ldots\ldots A_n \vdash B$

then

$A_1, A_2, A_3, \ldots\ldots\ldots\ldots\ldots A_{n-1} \vdash \quad A_n \rightarrow B$

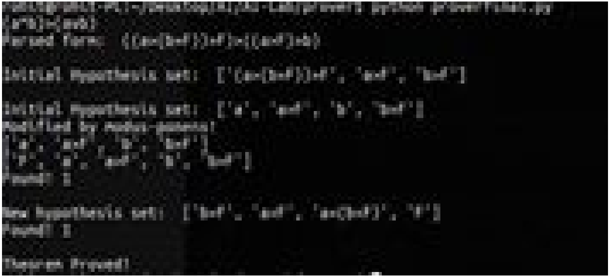
- We use the inverse of deduction theorem to prove the theorems in this assignment
- Our objective would be to derive F

Input & Output

- Input: The theorem to be proven
- Output: The output would be that the theorem is proved.

Approach

- **Step 1:**First the given input would be used to construct the left hand side using the reverse of Deduction theorem leaving only F on the right hand side (this is due to the fact that deduction theorem is true for both if and only if)
- **Step 2:**Now using Modus ponens we would exhaust all possibilities to derive F
- **Step 3:**If we are unable to derive F ,then we would try to prove left hand side of a hypothesis using remmaining of the hypothesis present on the left hand set of the hypothesis in deduction theorem
- **Step 4:**If a left hand side of a hypothesis can be derived using the remmaning ones its right hand side part would be added to the hypothesis set and again we would continue from the Step 2



- In the above theorem without any human help the theorem prover was able to prove $(a\ and\ b) -> (a\ or\ b)$

Results

```
rohit@nsl-67:~/Desktop/AI-Lab/prover$ python proverfinal.py
 ((p->q)->((r->s)->t)) ->  ( (u->((r->s)->t))-> ((p->u)->(s->t)))
Parsed form:  ((p>q)>((r>s)>t))>((u>((r>s)>t))>((p>u)>(s>t)))

Initial Hypothesis set:  ['(p>q)>((r>s)>t)', 'p>u', 's', 't>F', 'u>((r>s)>t)']

Initial Hypothesis set:  ['p', 'p>u', 'q>F', 's', 't>F', 'u>((r>s)>t)']
Modified by modus-ponens!
['p', 'p>u', 'q>F', 's', 't>F', 'u>((r>s)>t)']
['p', 'p>u', 'q>F', 's', 't>F', 'u', 'u>((r>s)>t)']
Modified by modus-ponens!
['p', 'p>u', 'q>F', 's', 't>F', 'u', 'u>((r>s)>t)']
['(r>s)>t', 'p', 'p>u', 'q>F', 's', 't>F', 'u', 'u>((r>s)>t)']

Initial Hypothesis set:  ['p', 'p>u', 'q>F', 'r', 's', 's>F', 't>F', 'u', 'u>((r>s)>t)']
Modified by modus-ponens!
['p', 'p>u', 'q>F', 'r', 's', 's>F', 't>F', 'u', 'u>((r>s)>t)']
['(r>s)>t', 'F', 'p', 'p>u', 'q>F', 'r', 's', 's>F', 't>F', 'u', 'u>((r>s)>t)']
Found! 1

New hypothesis set:  ['p', 's', 'r>s', 't>F', 't', 'q>F', 'p>u', 'u', 'u>((r>s)>t)']
Modified by modus-ponens!
['p', 's', 'r>s', 't>F', 't', 'q>F', 'p>u', 'u', 'u>((r>s)>t)']
['(r>s)>t', 'F', 'p', 'p>u', 'q>F', 'r>s', 's', 't', 't>F', 'u', 'u>((r>s)>t)']
Found! 4

New hypothesis set:  ['s', 't>F', 'p>u', '(r>s)>t', 'u>((r>s)>t)', 'p>q']
Modified by modus-ponens!
['s', 't>F', 'p>u', '(r>s)>t', 'u>((r>s)>t)', 'p>q']
['(r>s)>t', 'p>q', 'p>u', 's', 't>F', 'u>((r>s)>t)']

Initial Hypothesis set:  ['p>q', 'p>u', 'r', 's', 's>F', 't>F', 'u>((r>s)>t)']
Modified by modus-ponens!
['p>q', 'p>u', 'r', 's', 's>F', 't>F', 'u>((r>s)>t)']
['F', 'p>q', 'p>u', 'r', 's', 's>F', 't>F', 'u>((r>s)>t)']
Found! 1

New hypothesis set:  ['s', 'r>s', 't>F', 't', 'p>u', 'u>((r>s)>t)', 'p>q']
Modified by modus-ponens!
['s', 'r>s', 't>F', 't', 'p>u', 'u>((r>s)>t)', 'p>q']
['F', 'p>q', 'p>u', 'r>s', 's', 't', 't>F', 'u>((r>s)>t)']
Found! 4

Theorem Proved!
```

- In the above theorem without any human help the theorem prover was able to prove $((p->q)->((r->s)->t))->((u->((r->s)->t))->((p->u)->(s->t)))$

Conclusion

- We have observed how a machine can syntactically attempt to prove a theorem
- But the problem is when can we decide a theorem is not provable
- We can observe the fine difference between proving semantically and syntactically

Digital Circuit Simulator: Using Prolog

April 29, 2015

Group 13

Mahindar A Kota
Rohit Kumar
Pranay Dhondi

Objective

- Aim of the assignment is to simulate the working of a boolean circuit
- Main goal of the assignment is to observe how predicate calculus can be used to simulate boolean cicuits

Input & Output

- Input: The input would be the predicate $verify(GATE, INPUT, OUPUT)$
- Output: The output would be whether that is the desired output

Approach

- We have two prolog files namely *general.pl* which would simulate the basic gates *and*, *or*, *not* and the file *gate.pl* which would have the connections between the basic gates .
- This connections are made to represent the logic represented by gate

Predicates

Following predicates are used :

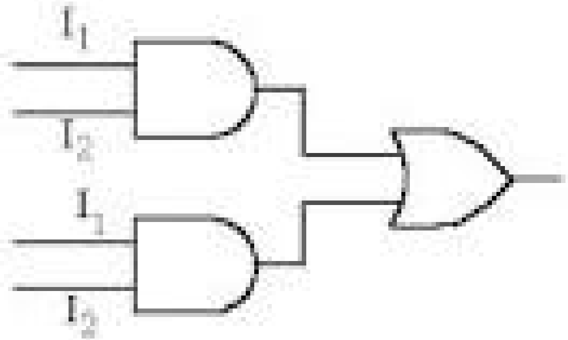
- signal(t,x) : t is the terminal , x represents signal value 0 or 1
- high(t) : t is the terminal , indicates whether singal value is 1
- low(t) : t is the terminal , indicates whether singal value is 0
- connected(t1,t2) : t1 is an output terminal and t2 is input terminal
- in(n,x) which denotes nth input of gate 'x'
- Output(x) output of circuit element 'x'

Predicates - Continued

- truthtable(x,I,O) where x is circuit element , O is expected output when I is input to x .
- is_earthed(t) t is ther terminal , indicates whether terminal is earthed or not .
- is_open(t1,t2) : t1 is the output terminal , t2 is input terminal , indicates whether t1 and t2 are connected or not .
- checkfault(x) : check whether circuit element 'x' has any faults

Circuit

XOR Circuit



Faulty Circuit

- For a given gate for its truth table if we get one of its input to be always zero then we say its earthed
- For a gate if its output is neither an input nor connected to an output terminal of another we determining output of that gate to be broken

Conclusion

- We can observe how using predicate calculus, a circuit simulator could be developed
- Prolog exhausts all possible values for a variable in a given predicate to evaluate to true. This feature makes a prolog a practical implementation of predicate calculus.

Hidden Markov Model: Graphene to Phoneme Conversion and viceversa

April 29, 2015

Group 13

Mahindar A Kota
Rohit Kumar
Pranay Dhondi

Objective

- Aim of the assignment is to use the Vitterbi Algorithm to give the best phoneme sequence for a give grapheme sequence and viceversa
- Main goal of the assignment is to observe how a machine trained using the training data provided can infer new state sequences

Vitterbi Algorithm

1. Initialization

SEQSCORE(1,1)=1.0

BACKPTR(1,1)=0

For(i=2 to N) do

    SEQSCORE(i,1)=0.0

[expressing the fact that first state is S_1]

2. Iteration

For(t=2 to T) do

  For(i=1 to N) do

    SEQSCORE(i,t) = $\text{Max}_{(j=1,N)}$

$$[SEQSCORE(j,(t-1)) * P(Sj \xrightarrow{a_k} Si)]$$

    BACKPTR(I,t) = index *j* that gives the MAX above

3. Seq. Identification

C(T) = i that maximizes SEQSCORE(i,T)
  For i from (T-1) to 1 do
    C(i) = BACKPTR[C(i+1),(i+1)]

- $SEQSCORE(i, j)$ keep tracks of the best probability by which we can have the state i at the j^{th} position in the state sequence for the given output sequence
- $BACKPTR(i, j)$ keeps track of the state at the $j-1$ for which the best probability of state i at the j^{th} position is obtained

Input & Output

- We are provided with a training data i.e phonemes and their respective graphemes. We consider grapheme and phoneme sequence of equal length
- Graphene to phoneme
  - Input: Grapheme
  - Output: Phoneme
- Phoneme to grapheme
  - Input: Phoneme
  - Output: Grapheme

Approach

- **Step 1:**First we calculated the transmission and emission probabilities using the training data
- **Step 2:**Now we would apply the Vitterbi Algorithm to calculate the state sequence for the observation sequence
- **Step 3:**Now we do a 5-fold cross validation accuracy on phoneme and grapheme level respectively

Trigram

- According to Markov Asssumptions the current state depends only on the previous state whereas in the trigram approach the state depends on the previous two states
- The transition probabilities table would now contain NXN rows and N coloumns

Results - Grapheme to Phoneme Conversion - With Trigram

- The above resulted in a percentage of 81.5002%

Results - Phoneme to Grapheme Conversion - With Trigram

- The above resulted in a percentage of 83.0071%

Confusion Matrix - Without Trigram

- We can observe that it is being confused out of this many times

Conclusion

- We have learnt the implementation of Vitterbi algorithm to predict state sequence for a given observation sequence depending on previously trained data
- We have observed that changing the Markov Assumption to implement trigram lead to an improvement of about 1%
- We can observe how a machine gets confused at phonemes related to vowels. We can observe as how we can distinguish between the sounds of vowels whereas a machine easily gets confused
- This assignment shows a beautiful implementation how text can be converted into speech and viceversa by machine (Festival Interface works very well)

PTA and Feed Forward Neural Network

April 29, 2015

Group 13

Mahindar A Kota
Rohit Kumar
Pranay Dhondi

Objective

- Aim1: To use PTA to calculate weights for a single perceptron for a majority function
- Aim2: To use the Backpropagation algorithm to convert graphemes to phonemes and phonemes to graphemes for a sigmoid network
- Main goal of the assignment is to observe how a trained neural network gives a state sequence on new training sequence

Perceptron Training Algorithm

1. Start with a random value of w
   ex: <0,0,0…>
2. Test for $wx_i > 0$
   If the test succeeds for i=1,2,…n
   then return w
3. Modify w, $w_{next} = w_{prev} + x_{fail}$

Back Propagation Theorem

- General weight updating rule:

$$\Delta w_{ji} = \eta \delta j o_i$$

- Where

$$\delta_j = (t_j - o_j) o_j (1 - o_j) \quad \text{for outermost layer}$$

$$= \sum_{k \in \text{next layer}} (w_{kj} \delta_k) o_j (1 - o_j) \quad \text{for hidden layers}$$

- Δw_{ji} denotes the amount by which the weight between the neuron i and neuron j has to be updated
- η denotes the learning rate and o_j denotes the output at neuron j

Input & Output - PTA

- Input: The input is the truth table
- Output: The output would the weights of inputs connected to the neuron including the bias.

Results-PTA

- Implented the majority function

```
rohit@nsl-67:~/Desktop/AI-Lab/perceptron$ ./a.out
Enter no. of variables in truth table: 3
Enter output for 0 0 0  : 0
Enter output for 0 0 1  : 0
Enter output for 0 1 0  : 0
Enter output for 0 1 1  : 1
Enter output for 1 0 0  : 0
Enter output for 1 0 1  : 1
Enter output for 1 1 0  : 1
Enter output for 1 1 1  : 1
Printing weights::
3.03488 3.04 3.04355
nIterations : 84
Value of theta is: 4.40049
```

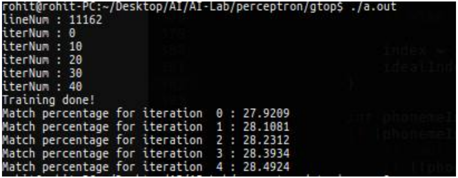
- In 84 iterations the algorithm terminated to find the suitable weights

Results-BackPropagation - Grapheme to Phoneme

```
rohit@rohit-PC:~/Desktop/AI/AI-Lab/perceptron/gtop$ ./a.out
lineNum : 11162
iterNum : 0
iterNum : 10
iterNum : 20
iterNum : 30
iterNum : 40
Training done!
Match percentage for iteration  0 : 40.8428
Match percentage for iteration  1 : 40.9309
Match percentage for iteration  2 : 41.3166
Match percentage for iteration  3 : 41.1233
Match percentage for iteration  4 : 41.2954
```

- We have taken 1 hidden layer, 70 sigmoid neurons in hidden layer, obtained an accuracy of 41%

Results-BackPropagation - Phoneme to Grapheme



```
rohit@rohit-PC:~/Desktop/AI/AI-Lab/perceptron/gtop$ ./a.out
lineNum : 11162
iterNum : 0
iterNum : 10
iterNum : 20
iterNum : 30
iterNum : 40
Training done!
Match percentage for iteration  0 : 27.9209
Match percentage for iteration  1 : 28.1081
Match percentage for iteration  2 : 28.2312
Match percentage for iteration  3 : 28.3934
Match percentage for iteration  4 : 28.4924
```

-
- We have taken 1 hidden layer, 50 sigmoid neurons in hidden layer, obtained an accuracy of 28%

Comparision between Neural and HMM

Conversion	Accuracy without trigram	Accuracy with trigram	Accuracy with neural network
Grapheme to Phoneme	~79%	[illegible]	~41%
Phoneme to Grapheme	~43%	~40.5%	~29%

Conclusion

- PTA takes long time (due to inherent hardness) to calculate weights even for small input linearly seperable functions
- Main difficulty in Backpropagation algorithm is to decide on the number of neurons in the hidden layer
- We can observe as we increased the number of iterations to train, there is an increase in the percentage of conversion