1 Search

Question 1 exercise 3.5 (4 points)
Consider the $n$-queens problem using the "efficient" incremental formulation given on page 67. Explain why the state space size is at least $\sqrt[3]{n!}$ and estimate the largest $n$ for which exhaustive exploration is feasible. 
(Hint: Derive a lower bound on the branching factor by considering the maximum number of squares that a queen can attack in any column.)

Question 2 exercise 4.1 (5 points)
Trace the operation of A* search applied to the problem of getting to Bucharest from Lugoj using the straight-line distance heuristic. That is, show the sequence of nodes that the algorithm will consider and the $f$, $g$ and $h$ score for each node.

Question 3 exercise 4.11 a, b and d (3 points)
Give the name of the algorithm that results from each of the following spacial cases:
  a) Local beam search with $k = 1$.
  b) Local beam search with one initial state and no limit on the number of states retained.
  d) Genetic algorithm with population size $N = 1$.

Question 4 Cryptarithmetic puzzle (3 points).
Cryptarithmetic problems are defined as follows (see course book, page 140): Each letter should be assigned a distinct digit such that the resulting sum is arithmetically correct with the additional restriction that no leading zeros are allowed.

HELP + FAME = FALEC

Write down the solution and explain briefly your solution method (you may choose any method that you find suitable including software packages).
1.1 Answer 1

1.1.1 State-space

First we recognize that on any one row or column, we can place only one queen. On a \( n \)-queen board, we have \( n \) rows. Let’s consider the task of placing one queen in each row. In the first row we have \( n \) options, but in the next row, we might only have \( n - 3 \) options. Other queens may attack a maximum of three squares in the current row. That means that in row \( i \), we will have \textit{at least} \( n - 3i \) squares to choose from when placing queen \( i \). So the size of the state space, \( S \) will be at least:

\[
S = n(n-3)(n-6)(n-9)\ldots(n-3i)\ldots
\]

(1)

We now see that by multiplying \( S \) with two other similar series, \( S_1 \) and \( S_2 \),

\[
S_1 = (n-2)(n-5)(n-8)\ldots(n-3i+1)\ldots
\]

(2)

\[
S_2 = (n-1)(n-4)(n-7)\ldots(n-3i+2)\ldots
\]

(3)

we get:

\[
n!
\]

(4)

And, since \( S > S_1 \) and \( S > S_2 \) we see that \( S > \sqrt[3]{n!} \).

1.1.2 Maximum problem size

Using this incremental method gives us a state-space in the order \( \sqrt[3]{n!} \). Modern desktop computers have the capacity to do in the order \( 10^{10} \) operations each second. Exploring one state certainly requires more than one operation, but on the other hand we could certainly wait more than one second. So, this method might be practical for problems with up to 30 queens, where the state-space reaches the order \( 10^{10} \).
1.2 Answer 2

A* with tree-search. In this method we always choose to expand (look at the neighbours) the tree-node, \( n \) with the smallest \( f(n) \) where \( f(n) \) is based on both the shortest distance found so far, \( g(n) \) and the estimated distance left to travel given by the heuristic function \( h(n) \). A* might be said to be Dijkstra’s algorithm guided by a heuristic function.

\[
f(n) = g(n) + h(n)
\]  

(5)

1.2.1 The path from Lugoj to Bucharest

<table>
<thead>
<tr>
<th>n</th>
<th>g(n)</th>
<th>h(n)</th>
<th>f(n)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Step 0: Lugoj</td>
<td>0</td>
<td>244</td>
<td>244</td>
</tr>
<tr>
<td>Step 1: Timisoara</td>
<td>111</td>
<td>329</td>
<td>440</td>
</tr>
<tr>
<td>Mehadia</td>
<td>70</td>
<td>241</td>
<td>311</td>
</tr>
<tr>
<td>Step 2: (expanding Mehadia)</td>
<td>Drobeta</td>
<td>145</td>
<td>242</td>
</tr>
<tr>
<td>Step 3: (expanding Drobeta)</td>
<td>Craiova</td>
<td>265</td>
<td>160</td>
</tr>
<tr>
<td>Step 4: (expanding Craiova)</td>
<td>Rimnicu Vilcea</td>
<td>411</td>
<td>193</td>
</tr>
<tr>
<td>Pitesti</td>
<td>403</td>
<td>100</td>
<td>503</td>
</tr>
<tr>
<td>Step 5: (expanding Timisoara)</td>
<td>Arad</td>
<td>229</td>
<td>366</td>
</tr>
<tr>
<td>Step 6: (expanding Pitesti)</td>
<td>Bucharest</td>
<td>504</td>
<td>0</td>
</tr>
</tbody>
</table>

1.3 Answer 3

Local beam search starts with \( k \) random start states. It looks at all the neighbouring states of these states and then pick the \( k \) best successor states among all the neighbours.

Genetic algorithms work with a population of solutions and gradually refine this population through evolution. An iteration with a GA start with the fitness-evaluation of all solutions in the population. Solutions with high fitness value are then selected with high probability to form the genetic material for the next generation. The selected solutions undergo crossover (mating) to swap parts of each solution with each other. Finally, a small number of random mutations to the population gives the ability to explore new solutions.
1.3.1 a)  
Hill-climbing search also known as greedy local search. We start in one random start state, look at all the the neighbouring states and continue our search in the direction of the best neighbour.

1.3.2 b)  
Breadth-first search. If we start in one state, look at all neighbours, then continue to look at their neighbours etc, etc - then we are doing breadth-first search.

1.3.3 d)  
Random walk. With only one individual, we can not do selection or crossover. Random mutation is all that is left of the GA.

1.4 Answer 4  
This is a Constraint Satisfaction Problem, CSP, and an excellent candidate for Local Search. Since we are dealing with a linear problem with linear constraints and integer-only variables we may also use, Integer Linear Programming, ILP, for which there are many libraries.

However, in this particular problem the solution unfolds in a fairly straightforward fashion, as soon as we start to look at the constraints and it’s implications. If we start to look at the column addition constraints, we soon find that $F$ (which may not be 0) must be equal to 1. So then $A$ must be 0 and so on, and so on.

1.4.1 Solution:

```
  H E L P  9 5 6 7
+ F A M E  + 1 0 8 5
---------  ---------
F A L E C  1 0 6 5 2
```

2 ML

Question 1  
1. Generally we never test the same attribute twice along one path in a standard decision tree. Why not? (1.5 points)

2. Can you devise a situation where we may have to test on same attribute twice along one path? Give an example. (1.5 points)

3. Suppose we generate a training set from a decision tree and then apply decision-tree learning to that training set. Is it the case that the learning algorithm will eventually return the correct tree as the training set size goes to infinity? Why or why not? (2 points)

This question is inspired by exercises 18.4 and 18.5

Question 2 Two statisticians go to the doctor and are both given the same prognosis: A 40% chance that the problem is the deadly disease A, and a 60% chance of the fatal disease B. Fortunately, there are anti-A and
anti-B drugs that are inexpensive, 100% effective, and free of side-effects. The statisticians have the choice of taking one drug, both, or neither.

1. What will the first statistician (an avid Bayesian) do? How about the second statistician, who always uses the maximum likelihood hypothesis? (2.5 points)

2. The doctor does some research and discovers that disease B actually comes in two versions, dextro-B and levo-B, which are equally likely and equally treatable by the anti-B drug. Now that there are three hypotheses, what will the two statisticians do? (2.5 points)

This question is inspired by exercise 20.4

Question 3  1. Suppose that a training set contains only a single example, repeated 100 times. In 80 of the 100 cases, the single output value is 1; in the other 20, it is 0. What will a back-propagation network predict for this example, assuming that it has been trained and reaches a global optimum? (Hint: to find the global optimum, differentiate the error function and set to zero.) (2.5 points)

2. Construct by hand a neural network that computes the XOR function of three inputs. Make sure to specify what sort of units you are using along with their weights. (2.5 points)

This question is inspired by exercises 20.11 and 20.19

2.1 Answer 1

1. It’s unnecessary to test the same attribute twice in the path because we already know the outcome of the test.

2. If our test in the decision tree is boolean and the attribute is not. Then we maybe need to check twice or more if the attribute is equal to, greater than, less than and so on.

3. In the end, or long enough, it will generate a tree that gives the same answer as the “correct tree”. But the tree will probably not be equal to the “correct tree” because there is multiple ways to describe a logic expression with a decision tree.

2.2 Answer 2

1. The Bayesian statistician would take both anti drugs because then he is 100% sure that he gone bee cure. The maximum likelihood would take the anti-B drug because he probably have the disease B.

2. The Bayesian statistician would take both drugs. Now the maximum likelihood would take the anti-A drug because it’s now bigger probability that he have the disease A.
2.3 Answer 3

1. The probability that the network give back a 1 is 0.8. The weights in the network will be adjust so that the error is minimal. The error functions look like this:

\[ E = \frac{1}{2} \sum_i (y_i - a_i)^2 = \frac{1}{2} \left( 80 (1 - a_i)^2 + 20 (0 - a_i)^2 \right) = 100a_i^2 - 80a_i + 40 \]

That give us the derivative of the error with respect of the output \(a_1\):

\[ \frac{\delta E}{\delta a_1} = 100a_1 - 80 \]

Now we set the derivative to zero and then we get \(a_1 = 0.8\).

2. We can easy construct a truth table for the XOR with three inputs and then we use DNF to make a network.

<table>
<thead>
<tr>
<th>(x_1)</th>
<th>(x_2)</th>
<th>(x_3)</th>
<th>(x_1 \oplus x_2 \oplus x_3)</th>
<th>DNF</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>(\neg x_1 \land \neg x_2 \land x_3)</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>(\neg x_1 \land x_2 \land \neg x_3)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>(\neg x_1 \land x_2 \land \neg x_3)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>(x_1 \land \neg x_2 \land \neg x_3)</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>(x_1 \land \neg x_2 \land \neg x_3)</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>(x_1 \land \neg x_2 \land \neg x_3)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>(x_1 \land \neg x_2 \land \neg x_3)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>(x_1 \land \neg x_2 \land \neg x_3)</td>
</tr>
</tbody>
</table>

So the network will represent the logic function \((\neg x_1 \land \neg x_2 \land x_3) \lor (\neg x_1 \land x_2 \land \neg x_3) \lor (x_1 \land \neg x_2 \land \neg x_3)\).

3 NLP

Question 1 exercise 22.6 (5 points)

Question 2 exercise 22.8 a and b (5 points)

Question 3 The IBM Model 3 machine translation model assumes that, after the word choice model proposes a list of words and the offset proposes possible permutations of the words, the language model can choose the best permutation. This exercises investigates how sensible that assumption is. Try to unscramble the following sentences into the correct order...
yourself and then give an insight on how machine can unscramble them. The simpler model you can offer, the better. (5 points)

- have programming a seen never I language better
- loves john mary
- is the communication exchange of intentional information brought by about the production perception of and signs from drawn a of system signs conventional shared

3.1 Answer 1

22.6 Determine what semantic interpretation would be given to the following sentences by the grammar of this chapter:

a. It is a wumpus.

b. The wumpus is dead.

c. The wumpus is in 2,2.

Would it be a good idea to have the semantic interpretation for “It is a wumpus” be simil\(\exists x \in Wumpuses\)? Consider alternative sentences such as “It was a wumpus.”

**Answer a** \(\exists e \in Is([\exists w Wumpus(w)]) \land During(Now,e)\)

**Answer b** \(\exists e \in Dead([\exists w Wumpus(w)]) \land During(Now,e)\)

**Answer c** \(\exists e \in Is([\exists x Wumpus(w)],x) \land In(x,[2,2]) \land During(Now,e)\)

3.2 Answer 2

22.8 This exercise concerns grammars for very simple languages.

a. Write a context-free grammar for the language \(a^n b^n\).

b. Write a context-free grammar for the palindrome language: the set of all strings second half is the reverse of the first half.

A context-free grammar is a 4-tuple: \((V, \sum, R, S)\) where

- \(V\) set of variables
- \(\sum\) alphabet of terminals
- \(R\) set of productions/rewrite rules
- \(S\) start symbol

**Answer a** Context-free grammar for the language \(a^n b^n\):

\[
\begin{align*}
V & = \{a, b\} \\
\sum & = S \\
R & = S \rightarrow \varepsilon \\
S & \rightarrow aSb
\end{align*}
\]

This results in \(S \rightarrow aSb\varepsilon\) since we, probably, want it to hold for \(n \geq 0\).
Answer b Context-free grammar for the palindrome language.

\[ V = \{ a,b,c \} \]
\[ \Sigma = S \]
\[ R = S \rightarrow \varepsilon \]
\[ S \rightarrow a|b|c \]
\[ S \rightarrow aSa|bSb|cSc \]

Basis: \( \varepsilon, a, b \) and \( c \) are palindromes.

Induction: If \( S \) is a palindrome, the \( aSa, bSb \) and \( cSc \) are also palindromes.

Therefore the answer is \( S \rightarrow \varepsilon|a|b|c|aSa|bSb|cSc \)

3.3 Answer 3

<table>
<thead>
<tr>
<th>Scrambled text</th>
<th>Unscrambled text</th>
</tr>
</thead>
<tbody>
<tr>
<td>have programming a seen never I language better</td>
<td>I have never seen a better programming language</td>
</tr>
<tr>
<td>loves john mary</td>
<td>john loves mary OR mary loves john</td>
</tr>
<tr>
<td>is the communication exchange of intentional information brought by about the production and perception of signs drawn from a system of conventional shared</td>
<td>communication is the intentional exchange of information brought by about the production and perception of signs drawn from a shared system of conventional</td>
</tr>
</tbody>
</table>

The first two text are quite easy to unscramble, but the third one is harder. The unscrambled text can be found in the second paragraph, right at the beginning of chapter 22 in the book.

To have a machine unscramble the strings of word we could train a n-gram on a training corpus, POS-tag the words and put them in a Markov chain with probabilities between all pairs. After that we could run an EM-algorithm over it to extract the highest probable permutation.

4 Planning

4.1 Answer 1

- \( FlyPrecond(p, f, to, s) = At(p, f, s) \land Plane(p) \land Airport(f) \land Airport(to) \)
- \( At(p, x, Result(a, s)) \iff \)
  \[ (a = Fly(p, f, x) \land FlyPrecond(p, f, x, s)) \lor \]
  \[ At(p, x, s) \land (\neg (a = Fly(p, f, x)) \lor \neg FlyPrecond(p, f, x, s)) \]
- \( At(p, x, Result(a, s)) \iff \)
  \[ (a = (Fly(p, f, x) \land FlyPrecond(p, f, x, s)) \lor (Teleport(p, f, x) \land \neg Warped(p))) \lor \]
  \[ (At(p, x, s) \land (\neg (a = Fly(p, f, x)) \lor \neg FlyPrecond(p, f, x, s) \lor \neg Teleport(p, x', x))) \]
4.2 Answer 2

A) A literal must appear in all levels, because it will always have some state. If a literal does not appear in the final level it will by that definition not be in any other level. Thus it will not be achievable.

B) Because the graph is serial, there will always be only one next action for each state. Thus, the first time a state $S_i$ is achieved, the cost $i$ will be the cheapest to get to that state; by definition there can not be any cheaper way to that same state.

4.3 Question 3

4.3.1 Forward state planing

A forward state planning algorithm maintains a partial-plan as a linear tree of actions with the initial-state as it’s root. It refines the plan by growing the tree with applicable actions. Applicable actions are all actions who’s PRECONDITIONS are met.

Thus we can say that this is a form of a partial-plan since we start with a root-node/state and grow the tree with actions.

4.3.2 Backward state planing

A backward state plan is the reverse of forward state. Instead of the initial-state at the root, it uses the goal-state. Thus the tree will be a reverse tree of actions, using the inverse of each action to build the tree. The refinement operator is to grow the tree with actions that are compatible with the root-node, that is, the goal.

Thus we can say that this is a form of a partial-plan since we start with a root-node/state and grow the tree with actions.