Exercises for Artificial Intelligence I

Dr Sean B Holden, 2010-17

1 Introduction

These notes provide some extra exercises for *Artificial Intelligence I* and, depending on when you download them, their solutions.

2 Introduction and Agents

It is notoriously difficult to predict what will be possible in the future, so your answers might well be amusing to you when you find them in twenty years time.

1. If you haven't seen it already, watch the film *A.I. Artificial Intelligence* paying particular attention to the character *"Teddy"*.

2. A large number of subjects are mentioned in the initial lectures in terms of how they've influenced AI: for example philosophy, mathematics, economics and so on. How do these show up in Teddy's design?

3. What aspects of Teddy are within our current capabilities to design?

4. What aspects of Teddy would you expect to be able to implement within the next fifteen years. How about the next fifty years?

5. Are there aspects of Teddy that you would expect to elude us for one hundred years or more?

6. To what extent does the "natural basic structure" for an agent, as described in the lectures, form a useful basis for implementing Teddy's internals? What is missing?

3 Search

1. Explain why breadth-first search is optimal if path-cost is a non-decreasing function of node-depth.

2. In the graph search algorithm, assume a node is taken from the fringe and found *not* to be a goal and *not* to be in `closed`. We then add it to `closed` and add its descendants to `fringe`. Why do we *not* check the descendants first to see if they are in `closed`?

3. Iterative deepening depends on the fact that *the vast majority of the nodes in a tree are in the bottom level*.

   - Denote by $f_1(b, d)$ the total number of nodes appearing in a tree with branching factor b and depth d. Find an expression for $f_1(b, d)$.
   - Denote by $f_2(b, d)$ the total number of nodes generated in a complete iterative deepening search to depth d of a tree having branching factor b. Find an expression for $f_2(b, d)$ in terms of $f_1(b, d)$.

- How do $f_1(b, d)$ and $f_2(b, d)$ compare when b is large?

4. The $A^\star$ algorithm does not perform a goal test on any state *until it has selected it for expansion*. We might consider a slightly different approach: namely, each time a node is expanded check all of its descendants to see if they include a goal.

   Give two reasons why this is a misguided idea, where possible illustrating your answer using a specific example of a search tree for which it would be problematic.

5. The f-cost is defined in the usual way as

$$f(n) = p(n) + h(n)$$

   where n is any node, p denotes path cost and h denotes the heuristic. An admissible heuristic is one for which, for any n

$$h(n) \leq \text{actual distance from } n \text{ to the goal}$$

   and a heuristic is monotonic if for consecutive nodes n and n' it is always the case that

$$f(n') \geq f(n).$$

   - Prove that h is monotonic if and only if it obeys the triangle inequality, which states that for any consecutive nodes n and n'

$$h(n) \leq c_{n \to n'} + h(n')$$

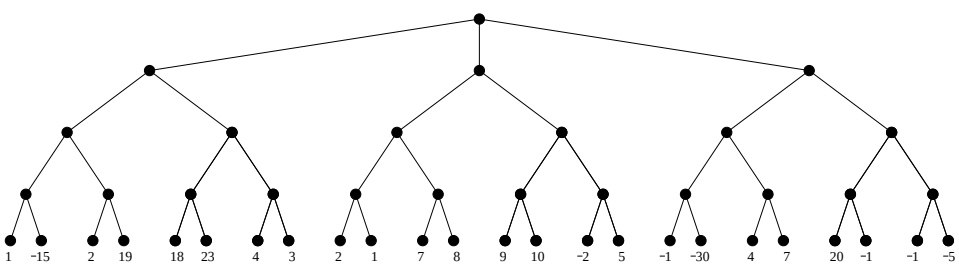
     where $c_{n \to n'}$ is the cost of moving from n to n'.
   - Prove that if a heuristic is monotonic then it is also admissible.
   - Is the converse true? (That is, are all admissible heuristics also monotonic?) Either prove that this is the case or provide a counterexample.

6. In RBFS we are replacing f values every time we backtrack to explore the current best alternative. This seems to imply a need to remember the new f values for all the nodes in the path we're discarding, and this in turn suggests a potentially exponential memory requirement. Why is this not the case?

7. Bi-directional search seems like a very good idea. Discuss how it might be problematic in practice.

4 Games

1. Consider the following game tree:



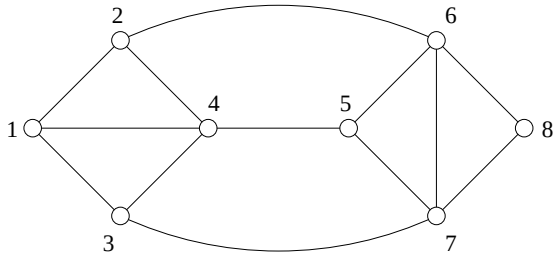
Large outcomes are beneficial for Max. How is this tree pruned by $\alpha - \beta$ minimax if Max moves first? (That is, Max is the root.) How is it pruned if Min is the root, and therefore moves first?

2. Implement the $\alpha - \beta$ pruning algorithm and use it to verify your answer to the previous problem.

3. Is the minimax approach to playing games optimal against an imperfect opponent? Either prove this is the case or give a counterexample.

5 Constraint satisfaction problems

1. Consider the following constraint satisfaction problem:



We want to colour the nodes using the colours red (R), cyan (C) and black (B) in such a way that connected nodes have different colours.

- Assume we attempt the assignments $1 = R$, $4 = C$, $5 = R$, $8 = C$, $6 = B$. Explain how *forward checking* operates in this example, and how it detects the need to backtrack.
- Will the AC-3 algorithm detect a problem earlier in this case? Explain the operation of the algorithm in this example.
- Implement the AC-3 algorithm and use it to verify your answer to the preceding problem.

6 Knowledge representation and reasoning

1. There were in fact *two* queries suggested in the notes for obtaining a sequence of actions. The details for

$$\exists a \; \exists s \; . \; \text{Sequence}(a, s_0, s) \wedge \text{Goal}(s)$$

were provided, but earlier in the notes the format

```
∃actionList.Goal(...  actionList ...)
```

was suggested. Explain how this alternative form of query might be made to work.

2. Making correct use of the situation calculus, write the sentences in FOL required to implement the `Shoot` action in Wumpus World. Write further sentences in FOL to allow movement and change of orientation.

3. Download and install a copy of *Prover9* from `www.cs.unm.edu/~mccune/prover9/`. (Hint: if you're Linux-based then you'll probably find it's already packaged. For instance, at the time of writing `yum install prover9.x86_64` works under Fedora 20.)

   Referring to exam question 2003, paper 9, question 8 assume that initially both owner and cat are in the living room. The cat can make its owner move to the kitchen by going to its food bowl in the kitchen and meeowing. It can then of course return to the living room and scratch something valuable.

   Implement sufficient knowledge in the situation calculus to allow an action sequence to be derived allowing the cat to achieve this, and use Prover9 to derive such an action sequence.

   In order to do this you need to know how to extract an answer from the theorem prover. Taking an easy example from the lecture notes:

```
formulas(assumptions).

wife(x,y) <-> (female(x) & married(x,y)).
female(Violet).
married(Violet,Bill).

end_of_list.

formulas(goals).

exists x wife(Violet,x).

end_of_list.
```

   Extracting the value of `x` requires two things: we need to move the goal into the assumptions (which is just like converting $\neg(A \rightarrow B)$ to $A \wedge \neg B$ when negating and converting to clauses) and we need to add a command to the knowledge base to get:

```
formulas(assumptions).

wife(x,y) <-> (female(x) & married(x,y)).
female(Violet).
married(Violet,Bill).

-wife(Violet,x) # answer(x).

end_of_list.

formulas(goals).
end_of_list.
```

   Here, the addition of `# answer(x)` causes the prover to output the value of `x` as part of the solution.

7 Planning

1. We've seen how heuristics can be used to speed up the process of searching. Planning has much in common with search. Can you devise any general heuristics that you might expect to speed up the planning process?

2. An undergraduate has turned up at this term's Big Party, only to find that it is in the home of her arch rival, who has turned her away. She spies in the driveway a large box and a ladder, and hatches a plan to gatecrash by getting in through a second floor window. Party on!

   Here is the planning problem. She needs to move the box to the house, the ladder onto the box, then climb onto the box herself and at that point she can climb the ladder to the window. Using the abbreviations

   - B - `Box`
   - L - `Ladder`
   - H - `House`
   - V - `Violet Scroot`
   - W - `Window`
   - D - `Driveway`

   The start state is $\neg\texttt{At}(B, H)$, $\neg\texttt{At}(L, B)$, $\neg\texttt{At}(V, W)$ and $\neg\texttt{At}(V, B)$. The goal is $\texttt{At}(V, W)$. The available actions are

   $\neg\texttt{At}(B, H), \neg\texttt{At}(L, B)$ — $\texttt{Move}(B, H)$ — $\texttt{At}(B, H)$

   $\texttt{At}(B, H), \texttt{At}(L, B), \texttt{At}(V, B)$ — $\texttt{Move}(V, W)$ — $\texttt{At}(V, W)$

   $\neg\texttt{At}(L, B)$ — $\texttt{Move}(L, B)$ — $\texttt{At}(L, B)$

   $\neg\texttt{At}(V, B)$ — $\texttt{Move}(V, B)$ — $\texttt{At}(V, B)$

   $\texttt{At}(L, B)$ — $\texttt{Move}(L, D)$ — $\neg\texttt{At}(L, B)$

   - Construct a solution to the problem using the partial order planning algorithm.
   - Construct the planning graph for this problem (you should probably start by finding a nice big piece of paper) and use the Graphplan algorithm to obtain a plan.

     If you are feeling keen, implement the algorithm for constructing the planning graph and use it to check your answer.

3. Return of the Evil Cat! Consider the problem involving the situation calculus and Prover9 above.

   - Represent this problem in the STRIPS format so that it could be given as input to the partial order planning algorithm.
   - Construct a solution to the problem using the partial order planning algorithm. How many specific plans can be extracted from the result?

4. Beginning with the domains

$$D_1 = \{\texttt{climber}\}$$
$$D_2 = \{\texttt{home}, \texttt{jokeShop}, \texttt{hardwareStore}, \texttt{spire}\}$$
$$D_3 = \{\texttt{rope}, \texttt{gorilla}, \texttt{firstAidKit}\}$$

and adding whatever actions, relations and so on you feel are appropriate, explain how the problem of purchasing and attaching a gorilla to a famous spire can be encoded as a constraint satisfaction problem (CSP).

If you are feeling keen, find a CSP solver and use it to find a plan. The course text book has a code archive including various CSP solvers at:

`http://aima.cs.berkeley.edu/code.html`

The following is an example of how to set up and solve a very simple CSP.

```
import java.io.*;
import java.util.*;
import aima.core.search.csp.*;

public class simpleCSP {
    public static void main(String[] args) {

        Variable v1 = new Variable("v1");
        Variable v2 = new Variable("v2");
        Variable v3 = new Variable("v3");

        List<String> domain1 = new LinkedList<String>();
        domain1.add("red");
        domain1.add("green");
        domain1.add("blue");

        Domain d1 = new Domain(domain1);

        List<Variable> vars = new ArrayList<Variable>();
        vars.add(v1);
        vars.add(v2);
        vars.add(v3);

        CSP csp = new CSP(vars);

        csp.setDomain(v1,d1);
        csp.setDomain(v2, d1);
        csp.setDomain(v3, d1);

        Constraint c1 = new NotEqualConstraint(v1,v2);
        Constraint c2 = new NotEqualConstraint(v1,v3);
        Constraint c3 = new NotEqualConstraint(v2,v3);
        csp.addConstraint(c1);
        csp.addConstraint(c2);
        csp.addConstraint(c3);
```

```
        ImprovedBacktrackingStrategy solver =
            new ImprovedBacktrackingStrategy();
        Assignment solution = new Assignment();
        solution = solver.solve(csp);

        System.out.println(solution);
    }
}
```

8 Learning

1. The purpose of this exercise is to gain some insight into the way in which the parameters of a basic, linear perceptron affect the position and orientation of its decision boundary. Recall that a linear perceptron is based on the function

$$f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b$$

where $\mathbf{x} \in \mathbb{R}^n$, $\mathbf{w} \in \mathbb{R}^n$ and $b \in \mathbb{R}$. The perceptron decides that a new input $\mathbf{x}$ is in class 1 if $f(\mathbf{x}) \geq 0$ and decides that the input is in class 2 otherwise. The decision boundary is therefore the collection of all points where $f(\mathbf{x}) = 0$.

It's always easy to find n distinct points where $f(\mathbf{x}) = 0$ because for any $\mathbf{w}$ and b we just need to solve

$$\mathbf{w}^T\mathbf{x}' = -b$$

which is easy using

$$\mathbf{x}^T = (\ (-b/w_1) \quad 0 \quad \cdots \quad 0\ )$$
$$\mathbf{x}^T = (\ 0 \quad (-b/w_2) \quad \cdots \quad 0\ )$$

and so on. If any of the weights is 0 this is problematic but easy to fix. (I leave it as a warm-up exercise to work out how.) Let $\mathbf{x}'$ and $\mathbf{x}''$ be two points where $f(\mathbf{x}') = 0$ and $f(\mathbf{x}'') = 0$. Let's concentrate on the case where $n = 2$. Consider the vector

$$\mathbf{y} = \mathbf{x}' - \mathbf{x}''.$$

Now take any number $a \in \mathbb{R}$ and look at what happens if we evaluate

$$f(\mathbf{x}' + a\mathbf{y}).$$

We obtain

$$\begin{aligned} f(\mathbf{x}' + a\mathbf{y}) &= \mathbf{w}^T(\mathbf{x}' + a\mathbf{y}) + b \\ &= \mathbf{w}^T\mathbf{x}' + a\mathbf{w}^T\mathbf{y} + b \\ &= f(\mathbf{x}') + a\mathbf{w}^T(\mathbf{x}' - \mathbf{x}'') \\ &= a(\mathbf{w}^T\mathbf{x}' - \mathbf{w}^T\mathbf{x}'') \\ &= a(-b - (-b)) \\ &= 0. \end{aligned}$$

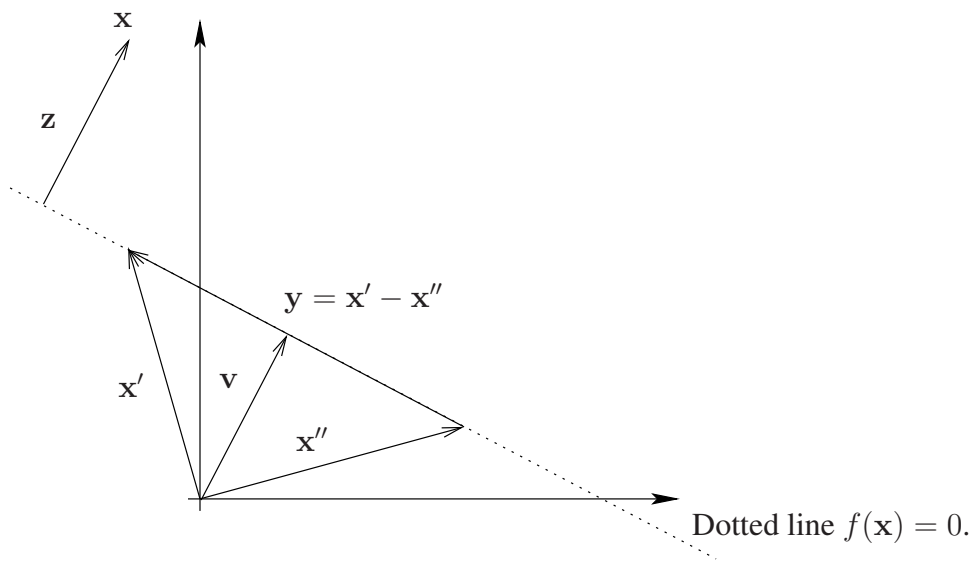


Figure 1: The decision boundary appears to be a straight line.

This works for *any* value $a \in \mathbb{R}$, and suggests that the decision boundary is a straight line in $\mathbb{R}^2$ as illustrated in figure 1. (Note however that we haven't yet demonstrated that $f(\mathbf{x}) \neq 0$ if $\mathbf{x}$ is not of the form $\mathbf{x} = \mathbf{x}' + a\mathbf{y}$.)

(a) Prove that the weight vector $\mathbf{w}$ is perpendicular to the line described by $\mathbf{x}' + a\mathbf{y}$; that is, the line corresponding to the set

$$\{\mathbf{x} | \mathbf{x} = \mathbf{x}' + a\mathbf{y} \text{ where } a \in \mathbb{R}\}.$$

(Hint: remember that vectors are perpendicular if their inner product is 0.) Note that this tells us that $\mathbf{w}$ describes the orientation of the decision boundary.

(b) Let $\mathbf{v}$ be the vector from the origin to the line described by $\mathbf{x}' + a\mathbf{y}$ and perpendicular to it as illustrated in figure 1. Prove that

$$||\mathbf{v}|| = \frac{|b|}{||\mathbf{w}||}.$$

Note that this tells us the following: if $||\mathbf{w}|| = 1$ then $|b|$ tells us the distance from the origin to the decision boundary.

(c) Let $\mathbf{x}$ be any point *not* on the line described by $\mathbf{x}' + a\mathbf{y}$. Let $\mathbf{z}$ be the vector from the line to $\mathbf{x}$ and perpendicular to the line as illustrated in figure 1. Prove that

$$||\mathbf{z}|| = \frac{|f(\mathbf{x})|}{||\mathbf{w}||}.$$

This tells us that points not on the line do not obey $f(\mathbf{x}) = 0$ and that the value of $f(\mathbf{x})$ tells us the distance from the decision boundary to $\mathbf{x}$.

(d) Prove that replacing $\mathbf{w}$ with $\mathbf{w}/||\mathbf{w}||$ and b with $b/||\mathbf{w}||$ does not alter the decision boundary.

2. In the application of neural networks to *pattern classification*—where we wish to assign any input vector $\mathbf{x}$ to membership in a specific *class*—it makes sense to attempt to interpret network outputs as probabilities of class membership.

   For example, in the medical diagnosis scenario presented in the lectures, where we try to map an input $\mathbf{x}$ to either class A (patient has the disease) or class B (patient is free of the disease) it makes sense to use a network with a single output producing values constrained between 0 and 1 such that the output $h(\mathbf{w};\mathbf{x})$ of a network using weights $\mathbf{w}$ is interpreted as

   $$h(\mathbf{w};\mathbf{x}) = \Pr(\mathbf{x} \text{ is in class } A).$$

   Clearly we also have

   $$\Pr(\mathbf{x} \text{ is in class } B) = 1 - h(\mathbf{w};\mathbf{x})$$

   and it follows that training examples should be labelled 1 and 0 for classes A and B respectively.

   Say we have a specific training example $(\mathbf{x}', 0)$. What does it tell us about how to choose a good $\mathbf{w}$? Clearly we might want to choose $\mathbf{w}$ to maximise[1]

   $$\begin{aligned}\Pr(\text{We see the example } (\mathbf{x}', 0)|\mathbf{w}) &= \Pr(\text{We see the label } 0|\mathbf{w}, \mathbf{x}') \times \Pr(\mathbf{x}') \\ &= \left\{1 - h(\mathbf{w};\mathbf{x}')\right\} \times \Pr(\mathbf{x}')\end{aligned}$$

   where the second step incorporates the assumption that $\mathbf{x}'$ and $\mathbf{w}$ are independent. This quantity is called the *likelihood* of $\mathbf{w}$. Given an entire training sequence

   $$\mathbf{s} = ((\mathbf{x}_1, c_1), (\mathbf{x}_2, c_2), \ldots, (\mathbf{x}_m, c_m))$$

   where the labels c_i take values 0 or 1 we can also consider choosing $\mathbf{w}$ to maximise the probability that the entire collection of m input vectors is labelled in the specified manner (the likelihood $\Pr(\mathbf{s}|\mathbf{w})$ of $\mathbf{w}$).

   Assuming that the examples in $\mathbf{s}$ are independent, show that in order to achieve this we should choose $\mathbf{w}$ to maximise the expression

   $$\sum_{i=1}^{m} c_i \log h(\mathbf{w};\mathbf{x}_i) + (1 - c_i) \log(1 - h(\mathbf{w};\mathbf{x}_i)).$$

   (Hint: When independence is assumed, $\Pr(A, B) = \Pr(A)\Pr(B)$, and you can maximise an expression equally well by maximising its $\log$.) What does this allow you to conclude about the version of the backpropagation algorithm presented in the lectures?

3. We now return to the case of regression. As in the previous question, the *likelihood* of a hypothesis h can be thought of as the probability of obtaining a training sequence $\mathbf{s}$ given that h is a perfect mapping from attribute vectors to classifications. Assume that $\mathcal{H}$ contains functions $h : X \to \mathbb{R}$ and examples are labelled using a specific target function $f \in \mathcal{H}$ but corrupted by noise, so

   $$\mathbf{s} = ((\mathbf{x}_1, o_1), (\mathbf{x}_2, o_2), \ldots, (\mathbf{x}_m, o_m))$$

   and

   $$o_i = f(\mathbf{x}_i) + e_i$$

[1] The basic result in probability theory being used here is that $\Pr(A, B|C) = \Pr(A|B, C)\Pr(B|C)$. You might want at this point to review the relevant notes.

for $i = 1, 2, \ldots, m$ where e_i denotes noise. If the attribute vectors are fixed, and the e_i are independent and identically distributed with the Gaussian distribution

$$p(e_i) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(\frac{-(e_i - \mu)^2}{2\sigma^2}\right)$$

where μ is the noise mean and σ^2 the noise variance, then the likelihood of any hypothesis is

$$p(\mathbf{s}|h) = p((o_1, o_2, \ldots, o_m)|h) = \prod_{i=1}^{m} p(o_i|h)$$

where the last step follows because the e_i are independent. Assume in the following that $\mu = 0$.

(a) Show that the mean of o_i is $f(\mathbf{x}_i)$ and the variance of o_i is σ^2.

(b) Show that

$$p(o_i|h) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(\frac{-(o_i - h(\mathbf{x}_i))^2}{2\sigma^2}\right).$$

(Hint: what happens to data having a Gaussian density if you linearly transform it?)

(c) Show that any hypothesis that *maximises* the likelihood is also one that *minimises* the quantity

$$\sum_{i=1}^{m} (o_i - h(\mathbf{x}_i))^2.$$

(d) What does this tell you about the specific example of the backpropagation algorithm given in the lectures?

4. The demonstration of the backpropagation algorithm given in the lectures can be improved. In solving the parity problem what we really want to know is the *probability* that an example should be placed in class one, exactly as described above. Probabilities lie in the interval $[0, 1]$, but the output of the network used in the lectures is unbounded.

- Derive the modification required to the algorithm if the activation function on the output node is changed from $g(x) = x$ to $g(x) = 1/(1+\exp(-x))$. (This is a function commonly used to produce probabilities as outputs, as it has range lying between 0 and 1.)
- Implement the modified algorithm. (Matlab is probably a good language to use.) Apply it to the parity data described in the lectures, and plot the results you obtain.