Exercises for Artificial Intelligence II
Solution Notes

Sean B. Holden, 2013-16

1 Introduction

1. Evaluate the integral

$$\int_{-\infty}^{\infty} \exp(-x^2)\, dx.$$

See the handout *How to evaluate Gaussian integrals* available on the course web site.

2. Evaluate the integral

$$\int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} \exp\left(-\frac{1}{2}\left(\mathbf{x}^T\mathbf{\Sigma}\mathbf{x} + \mathbf{x}^T\boldsymbol{\alpha} + \beta\right)\right)\, dx_1 \cdots dx_n$$

where $\mathbf{\Sigma} \in \mathbb{R}^{n \times n}$ is a real, symmetric $n \times n$ matrix, $\boldsymbol{\alpha} \in \mathbb{R}^n$ is a real vector, $\beta \in \mathbb{R}$ and

$$\mathbf{x}^T = \left[\begin{array}{cccc} x_1 & x_2 & \cdots & x_n \end{array}\right] \in \mathbb{R}^n.$$

See the handout *How to evaluate Gaussian integrals* available on the course web site.

2 Planning

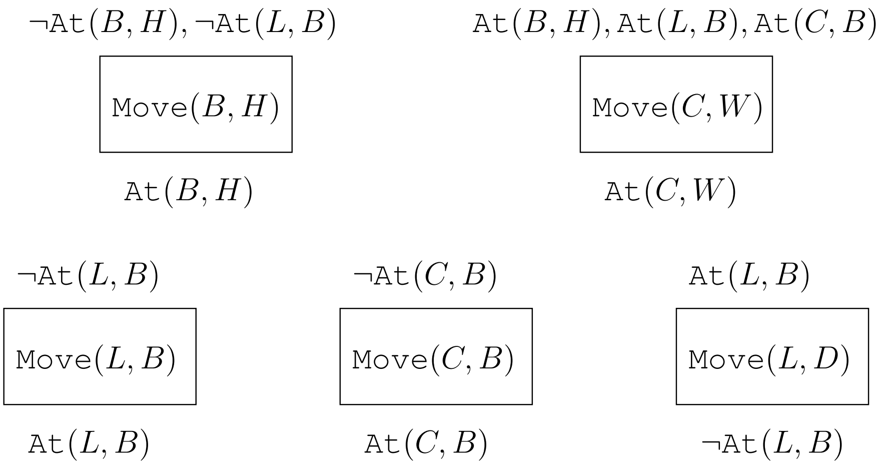
1. An undergraduate, eager to meet some new friends, has turned up at the term's Big Party, only to find that it is in the home of her arch-rival, who has turned her away. She notices in the driveway a large box and a ladder, and hatches a plan to gatecrash by getting in through a second-floor window. 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`
- C - `Ms Compsci`
- W - `Window`

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



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.

The first action and state levels are quite straightforward; after that, you will probably find it as hard to construct the graph by hand as to write a basic implementation. Appendix A contains an implementation in ML which can be used to verify that action level 1 is:

```
 [persist (not (proposition "at(B,H)")),
    persist (not (proposition "at(L,B)")),
    persist (not (proposition "at(C,W)")),
    persist (not (proposition "at(C,B)")),
    action
     (name "move(B,H)",
      [not (proposition "at(B,H)"), not (proposition "at(L,B)")],
      [proposition "at(B,H)"]),
    action
     (name "move(L,B)", [not (proposition "at(L,B)")],
      [proposition "at(L,B)"]),
    action
     (name "move(C,B)", [not (proposition "at(C,B)")],
      [proposition "at(C,B)"])]
```

with mutexes

```
[aMutex
     (action
       (name "move(B,H)",
        [not (proposition "at(B,H)"), not (proposition "at(L,B)")],
        [proposition "at(B,H)"]), persist (not (proposition "at(B,H)")),
      actionsInt),
    aMutex
     (action
       (name "move(L,B)", [not (proposition "at(L,B)")],
        [proposition "at(L,B)"]), persist (not (proposition "at(L,B)")),
      actionsInt),
    aMutex
     (action
       (name "move(L,B)", [not (proposition "at(L,B)")],
        [proposition "at(L,B)"]),
      action
       (name "move(B,H)",
        [not (proposition "at(B,H)"), not (proposition "at(L,B)")],
        [proposition "at(B,H)"]), actionsInt),
    aMutex
     (action
       (name "move(C,B)", [not (proposition "at(C,B)")],
        [proposition "at(C,B)"]), persist (not (proposition "at(C,B)")),
      actionsInt)]
```

and state level 1 is:

```
[not (proposition "at(C,B)"), not (proposition "at(C,W)"),
    not (proposition "at(L,B)"), not (proposition "at(B,H)"),
    proposition "at(B,H)", proposition "at(L,B)", proposition "at(C,B)"]
```

with mutexes

```
[cMutex
     (proposition "at(B,H)", not (proposition "at(B,H)"), mutexPreconditions),
    cMutex
     (proposition "at(L,B)", not (proposition "at(L,B)"), mutexPreconditions),
    cMutex (proposition "at(L,B)", proposition "at(B,H)", mutexPreconditions),
    cMutex
     (proposition "at(C,B)", not (proposition "at(C,B)"), mutexPreconditions)]
```

If you wish to continue then action level 2 is

```
[persist (not (proposition "at(C,B)")),
    persist (not (proposition "at(C,W)")),
    persist (not (proposition "at(L,B)")),
    persist (not (proposition "at(B,H)")), persist (proposition "at(B,H)"),
    persist (proposition "at(L,B)"), persist (proposition "at(C,B)"),
    action
     (name "move(B,H)",
      [not (proposition "at(B,H)"), not (proposition "at(L,B)")],
      [proposition "at(B,H)"]),
    action
     (name "move(C,W)",
      [proposition "at(B,H)", proposition "at(L,B)", proposition "at(C,B)"],
      [proposition "at(C,W)"]),
    action
     (name "move(L,B)", [not (proposition "at(L,B)")],
      [proposition "at(L,B)"]),
    action
     (name "move(C,B)", [not (proposition "at(C,B)")],
      [proposition "at(C,B)"]),
    action
     (name "move(L,D)", [proposition "at(L,B)"],
      [not (proposition "at(L,B)")])]
```

and the corresponding list of mutexes is BIG!!! After a little more work you should find it is possible to extract a plan.

2. 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 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);
    }
}
```

This is asking for a completion of the example started in the lectures, and there are many correct ways of achieving a successful plan. The question is included mostly as a basis for discussion.

3. Exam question: 2008, paper 7, question 6.

4. Exam question: 2009, paper 7, question 4.

5. Exam question: 2011, paper 7, question 2.

6. Exam question: 2012, paper 8, question 2.

3 Uncertainty

1. Prove that conditional independence, defined in the lectures notes as

$$\Pr(A, B|C) = \Pr(A|C)\Pr(B|C)$$

can equivalently be defined as

$$\Pr(A|B,C) = \Pr(A|C).$$

By definition

$$\Pr(A,B|C) = \frac{\Pr(A,B,C)}{\Pr(C)}.$$

I want to get $\Pr(A|B,C)$ on the left so

$$\Pr(A,B|C) = \frac{\Pr(A,B,C)}{\Pr(C)}\frac{\Pr(B,C)}{\Pr(B,C)} = \Pr(A|B,C)\frac{\Pr(B,C)}{\Pr(C)} = \Pr(A|B,C)\Pr(B|C).$$

Equating this with the first line in the question we have

$$\Pr(A|B,C) = \Pr(A|C).$$

2. Derive, from first principles, the general form of Bayes rule

$$\Pr(A|B,C) = \frac{\Pr(B|A,C)\Pr(A|C)}{\Pr(B|C)}.$$

By definition

$$\Pr(A|B,C) = \frac{\Pr(A,B,C)}{\Pr(B,C)}\frac{\Pr(A,C)}{\Pr(A,C)} = \Pr(B|A,C)\frac{\Pr(A,C)}{\Pr(B,C)}$$

and

$$\frac{\Pr(A,C)}{\Pr(B,C)} = \frac{\Pr(A|C)\Pr(C)}{\Pr(B|C)\Pr(C)}.$$

Cancelling the $\Pr(C)$ terms gives the result.

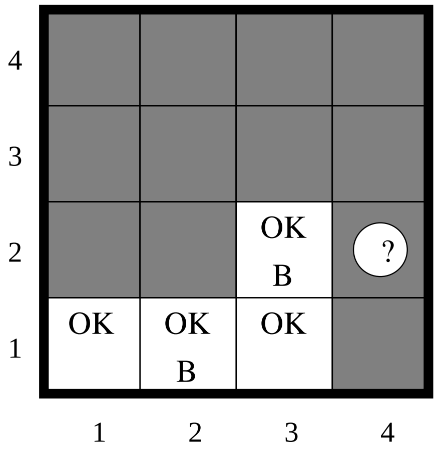
3. This question revisits the Wumpus World, but now our hero, having learned some probability by attending *Artificial Intelligence II*, will use probabilistic reasoning instead of situation calculus.

   Our hero, through careful consideration of the available knowledge on Wumpus caves, has established that each square contains a pit with prior probability 0.3, and pits are independent of one-another. Let $\texttt{Pit}_{i,j}$ be a Boolean random variable (RV) denoting the presence of a pit at row i, column j. So for all i, j

$$\Pr(\texttt{Pit}_{i,j} = \top) = 0.3 \qquad (1)$$

$$\Pr(\texttt{Pit}_{i,j} = \bot) = 0.7 \qquad (2)$$

   In addition, after some careful exploration of the current cave, our hero has discovered the following.



$$\texttt{Pit}_{1,1} = \perp$$

$$\texttt{Pit}_{1,2} = \perp$$

$$\texttt{Pit}_{1,3} = \perp$$

$$\texttt{Pit}_{2,3} = \perp$$

B denotes squares where a breeze is perceived. Let $\texttt{Breeze}_{i,j}$ be a Boolean RV denoting the presence of a breeze at i, j

$$\texttt{Breeze}_{1,2} = \texttt{Breeze}_{2,3} = \top \tag{3}$$

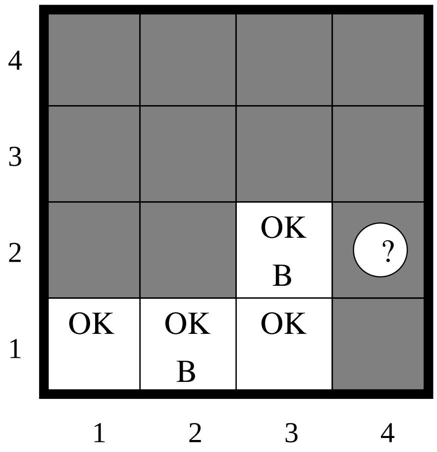
$$\texttt{Breeze}_{1,1} = \texttt{Breeze}_{1,3} = \perp \tag{4}$$

He is considering whether to explore the square at $2, 4$. He will do so if the probability that it contains a pit is less than 0.4. Should he?

Hint: The RVs involved are $\texttt{Breeze}_{1,2}, \texttt{Breeze}_{2,3}, \texttt{Breeze}_{1,1}, \texttt{Breeze}_{1,3}$ and $\texttt{Pit}_{i,j}$ for all the i, j. You need to calculate

$$\Pr(\texttt{Pit}_{2,4} | \text{all the evidence you have so far})$$

Here is the known situation



$$\texttt{Pit}_{1,1} = \perp$$

$$\texttt{Pit}_{1,2} = \perp$$

$$\texttt{Pit}_{1,3} = \perp$$

$$\texttt{Pit}_{2,3} = \perp$$

In addition we know that

$$\Pr(\texttt{Pit}_{i,j} = \top) = 0.3 \tag{5}$$

$$\Pr(\texttt{Pit}_{i,j} = \perp) = 0.7 \tag{6}$$

and pits are independent. Introduce the abbreviations

$$\texttt{safe} = (\texttt{Pit}_{1,1} = \perp) \wedge (\texttt{Pit}_{1,2} = \perp) \wedge (\texttt{Pit}_{1,3} = \perp) \wedge (\texttt{Pit}_{2,3} = \perp)$$

and

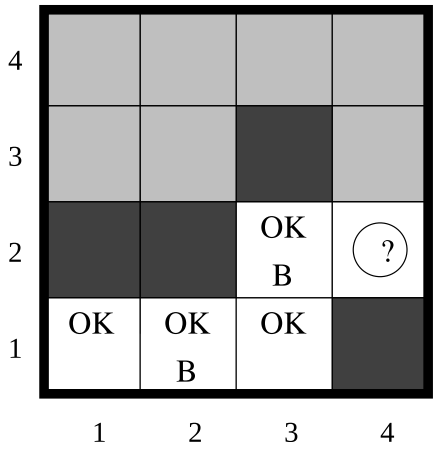
$$\texttt{breeze} = (\texttt{Breeze}_{1,1} = \bot) \wedge (\texttt{Breeze}_{1,2} = \top) \wedge (\texttt{Breeze}_{1,3} = \bot) \wedge (\texttt{Breeze}_{2,3} = \top)$$

Let $\texttt{Others}$ be a random variable (RV) collecting together all the $\texttt{Pit}_{i,j}$ RVs with the exception of $\texttt{Pit}_{2,4}$ and all those included in $\texttt{safe}$. So we have by the definition of conditional probability and the usual rule for marginalization

$$\begin{aligned}\Pr(\texttt{Pit}_{2,4}|\texttt{safe},\texttt{breeze}) &= c\Pr(\texttt{Pit}_{2,4},\texttt{safe},\texttt{breeze}) \\ &= c\sum_{\texttt{others}} \Pr(\texttt{Pit}_{2,4},\texttt{safe},\texttt{breeze},\texttt{others})\end{aligned} \tag{7}$$

which is a somewhat tricky sum containing 2^{11} terms.

We are rescued by the fact that the known rules for Wumpus caves present a conditional independence assumption. Namely, because the breezes depend only on *adjacent* squares we concentrate attention only on those. Split $\texttt{Others}$ into $\texttt{Adjacent}$ and $\texttt{Rest}$ where in the following diagram $\texttt{Adjacent}$ consists of the $\texttt{Pit}_{i,j}$ RVs for the dark grey squares and $\texttt{Rest}$ consists of the $\texttt{Pit}_{i,j}$ RVs for the light grey squares.



Now we can write

$$\Pr(\texttt{breeze}|\texttt{safe},\texttt{Pit}_{2,4},\texttt{Adjacent},\texttt{Rest}) = \Pr(\texttt{breeze}|\texttt{safe},\texttt{Pit}_{2,4},\texttt{Adjacent}) \tag{8}$$

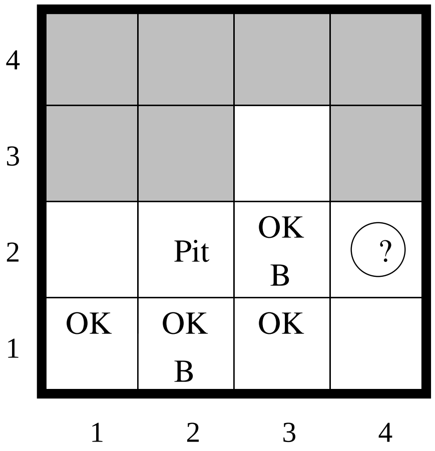
Combining equations (7) and (8) we have

$$\begin{aligned}&\Pr(\texttt{Pit}_{2,4}|\texttt{safe},\texttt{breeze}) \\ &\quad= c\sum_{\texttt{adjacent}}\sum_{\texttt{rest}} \Pr(\texttt{breeze}|\texttt{safe},\texttt{Pit}_{2,4},\texttt{adjacent})\Pr(\texttt{safe},\texttt{Pit}_{2,4},\texttt{adjacent},\texttt{rest}) \\ &\quad= c\sum_{\texttt{adjacent}} \Pr(\texttt{breeze}|\texttt{safe},\texttt{Pit}_{2,4},\texttt{adjacent})\sum_{\texttt{rest}}\Pr(\texttt{safe},\texttt{Pit}_{2,4},\texttt{adjacent},\texttt{rest}) \\ &\quad= c\Pr(\texttt{safe})\Pr(\texttt{Pit}_{2,4})\sum_{\texttt{adjacent}} \Pr(\texttt{breeze}|\texttt{safe},\texttt{Pit}_{2,4},\texttt{adjacent})\Pr(\texttt{adjacent})\end{aligned}$$

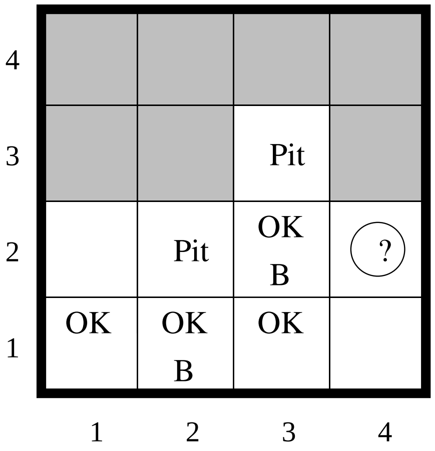
Now

$$\Pr(\texttt{breeze}|\texttt{safe},\texttt{Pit}_{2,4},\texttt{adjacent}) = \begin{cases} 1 & \text{if the configuration would produce the observed breezes} \\ 0 & \text{otherwise}\end{cases}$$

and only two values for $\texttt{adjacent}$ would give a value of 1.



$\Pr(\texttt{adjacent}) = 0.7^3 \times 0.3$



$\Pr(\texttt{adjacent}) = 0.7^2 \times 0.3^2$

At this point all that is left is to plug in some numbers.

$$\Pr(\texttt{Pit}_{2,4} = \top | \texttt{safe}, \texttt{breeze}) = c \times 0.7^4 \times 0.3 \times ((0.7^3 \times 0.3) + (0.7^2 \times 0.3^2))$$

and

$$\Pr(\texttt{Pit}_{2,4} = \bot | \texttt{safe}, \texttt{breeze}) = c \times 0.7^4 \times 0.7 \times ((0.7^3 \times 0.3) + (0.7^2 \times 0.3^2))$$

Obtaining the value of c as usual gives

$$\Pr(\texttt{Pit}_{2,4} = \top | \texttt{safe}, \texttt{breeze}) = 0.3$$

and

$$\Pr(\texttt{Pit}_{2,4} = \bot | \texttt{safe}, \texttt{breeze}) = 0.7$$

4. Continuing with the running example of the roof-climber alarm...

The porter in lodge 1 has left and been replaced by a somewhat more relaxed sort of chap, who doesn't really care about roof-climbers and therefore acts according to the probabilities

$$\begin{array}{rr} \Pr(l1|a) = 0.3 & \Pr(\neg l1|a) = 0.7 \\ \Pr(l1|\neg a) = 0.001 & \Pr(\neg l1|\neg a) = 0.999 \end{array}$$

Your intrepid roof-climbing buddy is on the roof. What is the probability that lodge 1 will report him? Use the variable elimination algorithm to obtain the relevant probability. Do you learn anything interesting about the variable $L2$ in the process?

For this query the natural way to factorize the joint distribution is

$$\Pr(L1|c) = \frac{1}{Z}\Pr(c) \sum_A \sum_G \Pr(G)\Pr(A|c,G)\Pr(L1|A) \sum_{L2} \Pr(L2|c)$$

and as the sum over $L2$ always equals 1 we have discovered that the term for $L2$ is irrelevant. We can further factorize in two ways

$$\begin{aligned} \Pr(L1|c) &= \frac{1}{Z}\Pr(c) \sum_G \Pr(G) \sum_A \Pr(A|c,G)\Pr(L1|A) \\ &= \frac{1}{Z}\Pr(c) \sum_A \Pr(L1|A) \sum_G \Pr(G)\Pr(A|c,G) \end{aligned}$$

and you should get the same answer whichever factorization you use. (You did, of course, try it both ways didn't you?)

Using the first alternative, and working from right to left, we combine the factors $F_A(A, G)$ and $F_{L1}(L1, A)$ to get

$F_{A,L1}(A, L1, G) =$

A	$L1$	G	
T	T	T	$.98 \times .3$
T	T	F	$.96 \times .3$
T	F	T	$.98 \times .7$
T	F	F	$.96 \times .7$
F	T	T	$.02 \times .001$
F	T	F	$.04 \times .001$
F	F	T	$.02 \times .999$
F	F	F	$.04 \times .999$

then sum out A to get

$F_{\overline{A},L1}(L1, G) =$

$L1$	G	
T	T	$.98 \times .3 + .02 \times .001 = .29402$
T	F	$.96 \times .3 + .04 \times .001 = .28804$
F	T	$.98 \times .7 + .02 \times .999 = .70598$
F	F	$.96 \times .7 + .04 \times .999 = .71196$

Next, we incorporate the factor $F_G(G)$ to get

$F_{G,\overline{A},L1}(L1, G) =$

$L1$	G	
T	T	$.29402 \times .2$
T	F	$.28804 \times .8$
F	T	$.70598 \times .2$
F	F	$.71196 \times .8$

and sum out the G to get

$F_{\overline{G},\overline{A},L1}(L1) =$

$L1$	
T	$.289236$
F	$.710764$

At this point, we note that as we know that $C = T$ the remaining term $\text{Pr}(c)/Z$ serves only to normalize the distribution $\text{Pr}(L1|c)$, and performing the normalization yields the probability $.289$.

5. In the lecture notes, an example was given for which we would expect $\text{Pr}(A \rightarrow B)$ to be (relatively) much larger than $\text{Pr}(B|A)$. Suggest a situation where the converse would be true.

After a little experimentation you might believe that this is a little tricky. So, let's consider the general problem. We're asking for a situation where

$$\text{Pr}(\neg A \vee B) = \text{Pr}(\neg A) + \text{Pr}(B) - \text{Pr}(\neg A \wedge B) < \text{Pr}(B|A) = \frac{\text{Pr}(A \wedge B)}{\text{Pr}(A)}.$$

Noting that

$$\text{Pr}(\neg A \wedge B) = \text{Pr}(B) - \text{Pr}(A \wedge B)$$

(draw a Venn diagram if you need convincing) and re-arranging we have

$$1 - \text{Pr}(A) + \text{Pr}(A \wedge B) < \frac{\text{Pr}(A \wedge B)}{\text{Pr}(A)}.$$

Re-arranging this we have

$$\frac{1 - \text{Pr}(A)}{\text{Pr}(B|A)} + \text{Pr}(A) < 1.$$

Now attempt to make the first term small by adjusting B to make $\Pr(B|A)$ large. The maximum value it can take is 1, which leaves us with

$$1 < 1.$$

So: *there is no situation where the converse is true.*

6. Later in the course it is shown that in constructing a two-class classifier (such as a multilayer perceptron) the optimal approach involves computing $\Pr(\texttt{class}|\texttt{features})$. Suggest an approach to performing this calculation in practice. (Hint: apply Bayes' theorem and estimate some probabilities.) What problems might this present in practice, and what assumption(s) might you introduce to overcome them?

   The aim of this question is to prompt you to derive the *Naive Bayes Classifier*.

   Taking the approach suggested I have using Bayes' theorem

   $$\Pr(\texttt{class}|\texttt{features}) = \frac{1}{Z}\Pr(\texttt{features}|\texttt{class})\Pr(\texttt{class})$$

   where as usual Z just normalizes to insure we have a probability distribution. Call the two classes C_1 and C_2, so $\Pr(\texttt{class} = C_1) = 1 - \Pr(\texttt{class} = C_2)$. It's going to be pretty straightforward to estimate $\Pr(\texttt{class} = C_1)$ as it's exactly analogous to the ubiquitous example of estimating the probability that tossing a biased coin results in a head. Specifically, say you have a training sequence $\mathbf{s}$ with m examples, and m_1 of these are labelled as C_1. Then

   $$\Pr(\texttt{class} = C_1) \simeq \frac{m_1}{m}.$$

   Unfortunately estimating $\Pr(\texttt{features}|\texttt{class})$ is going to be more tricky. Say each example in the training set has n features. Even if these features are all binary we need to estimate about 2^n different numbers. To get a good estimate of just one of those numbers, say $\Pr(\texttt{features} = \mathbf{x}|\texttt{class})$ for some $\mathbf{x}$, means counting how many times $\mathbf{x}$ appears in the training set and so we should expect to need m to be at least a multiple of 2^n. Ouch. (And if features have more than two values, or are continuous, then the situation is even worse.)

   The solution lies in the lecture notes under the title of *idiot's Bayes*. We make the extremely brave conditional independence assumption that, if $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ then

   $$\Pr(\texttt{features} = \mathbf{x}|\texttt{class}) = \prod_{i=1}^{n} \Pr(x_i|\texttt{class}).$$

   For binary features we now have to estimate n numbers instead of 2^n, and each is essentially equivalent, again, to the coin flipping case.

   Surprisingly, this can be a very effective method; see *Machine Learning* by Tom Mitchell, McGraw Hill, 1997 for an example involving classification of news stories.

7. In designing a Bayesian network you wish to include a node representing the value reported by a sensor. The quantity being sensed is real-valued, and if the sensor is working correctly it provides a value close to the correct value, but with some noise present. The correct value is provided by its first parent. A second parent is a boolean random variable that indicates whether the sensor is faulty. When faulty, the sensor flips between providing the correct value, although with increased noise, and a known, fixed incorrect value, again with some added noise. Suggest a conditional distribution that could be used for this node.

   Denote the parents using the real-valued RV V and the boolean RV F. Denote the value reported by the sensor using the real RV V'. When the sensor is working we might reasonably use a normal density

$N(V,\sigma)$ where σ is a parameter denoting the noise variance. When the sensor is faulty, let p be a parameter denoting the probability that the sensor reports an incorrect value, denoted by w and having variance σ', and let ϵ denote the increase in variance when the faulty sensor reports a (noisy) correct value. Combining these, a possible conditional distribution is

$$p(V'|V,F) = \begin{cases} N(V,\sigma) & \text{if } F = \texttt{false} \\ (1-p)N(V,\sigma+\epsilon) + pN(w,\sigma') & \text{if } F = \texttt{true} \end{cases} .$$

8. Exam question: 2005, paper 8, question 2.

9. Exam question: 2006, paper 8, question 9.

10. Exam question: 2009, paper 8, question 1.

4 Making decisions

1. Prove the result mentioned on slide 161:

$$\texttt{VPI}_E(E',E'') = \texttt{VPI}_E(E') + \texttt{VPI}_{E,E'}(E'').$$

Let's start by being a little more careful with the notation. The evidence E denotes what we already know, implying it's been measured and thus that we know $E = e$ for some value e. So we *should* write

$$\texttt{VPI}_e(E',E'') = \texttt{VPI}_e(E') + \texttt{VPI}_{e,e'}(E'').$$

This should set alarm bells ringing. The term $\texttt{VPI}_{e,e'}(E'')$, and hence the right hand side of the equation, now depends on the value of e' whereas the left hand side does not! *We have just established that the expression in the textbook is incorrect.*

This leaves us with the task of correcting it. Writing the left hand side in full

$$\begin{aligned}\texttt{VPI}_e(E',E'') &= \left\{\sum_{e',e''} \Pr(e',e''|e)\texttt{EU}(\text{action}|e,e',e'')\right\} - \texttt{EU}(\text{action}|e) \\ &= \left\{\sum_{e',e''} \Pr(e''|e,e')\Pr(e'|e)\texttt{EU}(\text{action}|e,e',e'')\right\} - \texttt{EU}(\text{action}|e) \\ &= \left\{\sum_{e'} \Pr(e'|e)\sum_{e''}\Pr(e''|e,e')\texttt{EU}(\text{action}|e,e',e'')\right\} - \texttt{EU}(\text{action}|e).\end{aligned}$$

Now,

$$\texttt{VPI}_{e,e'}(E'') = \left\{\sum_{e''} \Pr(e''|e,e')\texttt{EU}(\text{action}|e,e',e'')\right\} - \texttt{EU}(\text{action}|e,e').$$

Combining these expressions gives

$$\begin{aligned}\texttt{VPI}_e(E',E'') &= \left\{\sum_{e'} \Pr(e'|e)\left[\texttt{VPI}_{e,e'}(E'') + \texttt{EU}(\text{action}|e,e')\right]\right\} - \texttt{EU}(\text{action}|e) \\ &= \mathbb{E}\left[\texttt{VPI}_{e,E'}(E'')|e\right] + \sum_{e'}\Pr(e'|e)\texttt{EU}(\text{action}|e,e') - \texttt{EU}(\text{action}|e).\end{aligned}$$

Finally,

$$\mathtt{VPI}_e(E') = \left\{ \sum_{e'} \Pr(e'|e)\mathtt{EU}(\text{action}|e, e') \right\} - \mathtt{EU}(\text{action}|e)$$

and thus

$$\mathtt{VPI}_e(E', E'') = \mathtt{VPI}_e(E') + \mathbb{E}\left[\mathtt{VPI}_{e,E'}(E'')|e\right].$$

2. Evil Robot is teaching himself surgery. He believes that there are two treatments, t_1 and t_2 suitable for his first patient, each having three possible outcomes: `cure`, `death` and `amputation`. These have utilities of 100, -1000 and -250 respectively. Evil robot thinks that t_1 has probabilities 0.8, 0.1 and 0.1 respectively for the three outcomes and treatment t_2 has probabilities 0.75, 0.05 and 0.2. Compute the expected utility of each treatment.

   Evil Robot has been studying hard, and has learned that an unpleasant test T is available that might help him choose the better treatment. The test has a cost to the patient of -50, while the cost of not performing it is -2. (Evil robot will nonetheless conduct some slightly unpleasant tests.) He estimates that the probability of the test being positive is 0.7. He also thinks that, armed with a positive test he can give t_1 outcome probabilities of 0.9, 0.01 and 0.09 respectively, and t_2 outcome probabilities of 0.85, 0.02 and 0.13. If test T is negative then the outcome probabilities are unchanged. (He does the other tests just for fun.)

   In the interest of the patient, should Evil Robot use test T?

   The expected utility of action t_1 is

   $$(.8 \times 100) + (.1 \times -1000) + (.1 \times -250) = -45$$

   and for t_2

   $$(.75 \times 100) + (.05 \times -1000) + (.2 \times -250) = -25$$

   so

   $$\text{EU}(\mathtt{action}|E) = -25.$$

   If test T is positive the utility of t_1 is

   $$(.9 \times 100) + (.01 \times -1000) + (.09 \times -250) = 57.5$$

   and for t_2

   $$(.85 \times 100) + (.02 \times -1000) + (.13 \times -250) = 32.5$$

   giving

   $$\text{EU}(\mathtt{action}|E, T = \mathtt{true}) = 57.5$$

   and if T is false we still have

   $$\text{EU}(\mathtt{action}|E, T = \mathtt{false}) = -25$$

   so

   $$\text{VPI}_E(T) = ((.7 \times 57.5) + (.3 \times -25)) - (-25) = 57.75.$$

   Taking the cost into accout we have a remaining positive utility of 7.75, so Evil Robot should indeed conduct the test.

3. Exam question: 2007, paper 8, question 9.

4. Exam question: 2011, paper 8, question 8.

5. Exam question: 2013, paper 8, question 2.

5 HMMs

1. Derive the equation
$$b_{t+1:T} = \mathbf{SE}_{t+1} b_{t+2:T}$$
for the backward message in a hidden Markov model (lecture slide 208).

   From slide 186 the general equation for the backward message is
$$b_{t+1:T} = \sum_{s_{t+1}} \underbrace{\Pr(e_{t+1}|s_{t+1})}_{\text{Sensor model}} \underbrace{\Pr(e_{t+2:T}|s_{t+1})}_{b_{t+2:T}} \underbrace{\Pr(s_{t+1}|S_t)}_{\text{Transition model}} \tag{9}$$
and using the definitions of $\mathbf{E}_{t+1}$ and $b_{t+2:T}$ we have
$$\begin{aligned}
&\mathbf{E}_{t+1} b_{t+2:T} \\
&= \begin{pmatrix} \Pr(e_t+1|S_{t+1}=s_1) & \cdots & 0 \\ 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \Pr(e_t+1|S_{t+1}=s_n) \end{pmatrix} \begin{pmatrix} \Pr(e_{t+2:T}|S_{t+1}=s_1) \\ \Pr(e_{t+2:T}|S_{t+1}=s_2) \\ \vdots \\ \Pr(e_{t+2:T}|S_{t+1}=s_n) \end{pmatrix} \\
&= \begin{pmatrix} \Pr(e_t+1|S_{t+1}=s_1)\Pr(e_{t+2:T}|S_{t+1}=s_1) \\ \Pr(e_t+1|S_{t+1}=s_2)\Pr(e_{t+2:T}|S_{t+1}=s_2) \\ \vdots \\ \Pr(e_t+1|S_{t+1}=s_n)\Pr(e_{t+2:T}|S_{t+1}=s_n) \end{pmatrix}.
\end{aligned}$$
Finally using the definition of **S**
$$\begin{aligned}
&\mathbf{SE}_{t+1} b_{t+2:T} \\
&= \begin{pmatrix} \Pr(S_{t+1}=s_1|S_t=s_1) & \cdots & \Pr(S_{t+1}=s_n|S_t=s_1) \\ \Pr(S_{t+1}=s_1|S_t=s_2) & \cdots & \Pr(S_{t+1}=s_n|S_t=s_2) \\ \vdots & \ddots & \vdots \\ \Pr(S_{t+1}=s_1|S_t=s_n) & \cdots & \Pr(S_{t+1}=s_n|S_t=s_n) \end{pmatrix} \begin{pmatrix} \Pr(e_{t+1}|S_{t+1}=s_1)\Pr(e_{t+2:T}|S_{t+1}=s_1) \\ \Pr(e_{t+1}|S_{t+1}=s_2)\Pr(e_{t+2:T}|S_{t+1}=s_2) \\ \vdots \\ \Pr(e_{t+1}|S_{t+1}=s_n)\Pr(e_{t+2:T}|S_{t+1}=s_n) \end{pmatrix} \\
&= \begin{pmatrix} \sum_{s_{t+1}} \Pr(s_{t+1}|S_t=s_1)\Pr(e_{t+1}|s_{t+1})\Pr(e_{t+2:T}|s_{t+1}) \\ \sum_{s_{t+1}} \Pr(s_{t+1}|S_t=s_2)\Pr(e_{t+1}|s_{t+1})\Pr(e_{t+2:T}|s_{t+1}) \\ \vdots \\ \sum_{s_{t+1}} \Pr(s_{t+1}|S_t=s_n)\Pr(e_{t+1}|s_{t+1})\Pr(e_{t+2:T}|s_{t+1}) \end{pmatrix}
\end{aligned}$$
which is identical to (9).

2. Explain why the backward message update should be initialized with the vector $(1, \dots, 1)$.

   At the first iteration of the backward update we have $t = T - 1$ and so we are aiming to compute the quantity
$$b_{T:T} = \Pr(e_T|S_{T-1}).$$
Using the usual trick to expand this
$$\begin{aligned}
b_{T:T} = \Pr(e_T|S_{T-1}) &= \sum_{s_T} \Pr(e_T, s_T|S_{T-1}) \\
&= \sum_{s_T} \Pr(e_T|s_T, S_{T-1})\Pr(s_T|S_{T-1}) \\
&= \sum_{s_T} \Pr(e_T|s_T)\Pr(s_T|S_{T-1})
\end{aligned} \tag{10}$$

where we've used the conditional independence assumption in the last line. Comparing (10) and (9) we see that to make them match a vector of 1s should be used at the first step.

3. Establish how the prior $\Pr(S_0)$ should be included in the derivation of the Viterbi algorithm. (This is mentioned on slide 192, but no detail is given.)

   There are a couple of ways to answer this, depending on exactly how you interpret the prior. If we follow strictly the presentation given in the notes, then the prior is used to choose an initiel state S_0 *which produces no observation*. The state transition probabilities are then used to generate the first state S_1 for which an observation E_1 is produced. You might reasonably argue that it makes more sense to choose S_1 directly using a prior; a little thought however shows that the two possibilities are essentially equivalent. To see this note that in the first version

   $$\Pr(S_1) = \sum_{s_0} \Pr(s_0, S_1) = \sum_{s_0} \Pr(S_1|s_0)\Pr(s_0)$$

   which allows us to compute a new prior for use in the second version.

   Once that has been done, the Viterbi algorithm proceeds simply by using the prior $\Pr(S_1)$ to label the first column of the lattice instead of the transition probabilities. This is illustrated in the answer to the next problem.

4. A hidden Markov model has transition matrix $S_{ij} = \Pr(S_{t+1} = s_j | S_t = s_i)$ where

   $$\mathbf{S} = \begin{pmatrix} 0.2 & 0.4 & 0.4 \\ 0.1 & 0.6 & 0.3 \\ 0.8 & 0.1 & 0.1 \end{pmatrix}.$$

   In any state we observe one of the symbols △, ▽, ○, □ with the following probabilities:

	△	▽	○	□
s_1	0.7	0.1	0.1	0.1
s_2	0.3	0.2	0.4	0.1
s_3	0.4	0.2	0.2	0.2

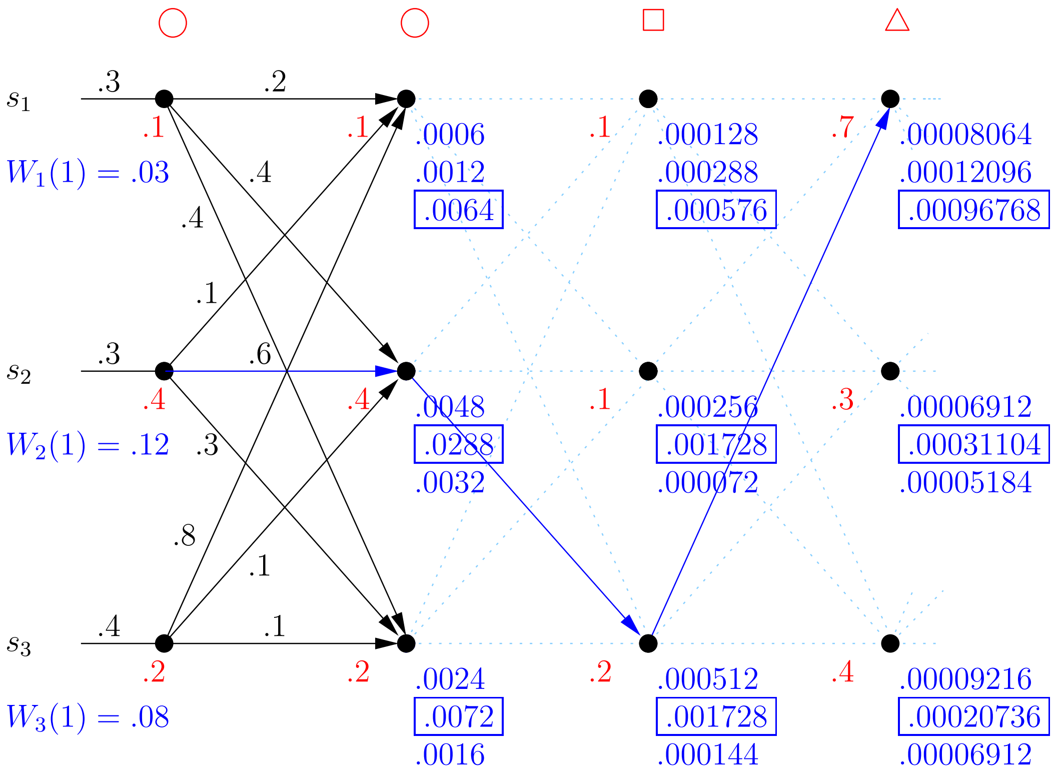
   .

   Prior probabilites for the states are $\Pr(s_1) = 0.3$, $\Pr(s_2) = 0.3$ and $\Pr(s_3) = 0.4$. We observe the sequence of symbols

   ○ ○ □ △ △□ ▽ □.

   Use the Viterbi algorithm to infer the most probable sequence of states generating this sequence.

   Here is the lattice for the first four observations. I have used the prior directly to label the column for S_1, rather than starting with S_0. See the answer to the previous problem for a discussion of this.



Numbers in black denote prior or transition probabilities, numbers in red denote probabilities for the observations, and numbers in blue denote values used in computing $W_i(j)$. Where three blue values are associated with a node, they are ordered according to which previous state they correspond to, and the maximum is boxed, denoting the actual $W_i(j)$ value for that node. Arrows in blue show the maximum probability path.

At this point the pattern should be clear. It is of course sensible to use something like Matlab or Octave to check your answer. Unfortunately both implementations assume that the HMM starts in state 1, which doesn't really help much. Luckily there is another free system called R with a more correct implementation. The code looks like this:

```
# R code to check the solution to the Viterbi algorithm problem
# in the AI II problem sheet.
#
# Sean Holden 2014.

.libPaths("~/R/")
library(HMM)

# Set up the HMM.

stateNames <- c("s1", "s2", "s3")
symbolNames <- c("^", "v", "O", "|_|")
prior <- c(0.3, 0.3, 0.4)
S <- matrix(c(0.2, 0.2, 0.8, 0.4, 0.6, 0.1, 0.4, 0.3, 0.1), 3, 3)
E <- matrix(c(0.7, 0.3, 0.4, 0.1, 0.2, 0.2, 0.1, 0.4, 0.2, 0.1, 0.1, 0.2), 3, 4)

model <- initHMM(stateNames, symbolNames, prior, S, E)

# Run the Viterbi algorithm for the given observations.
```

```
observed <- c("O", "O", "|_|", "^", "^", "|_|", "v", "|_|")
answer <- viterbi(model, observed)
```

and should be self-explanatory. It tells me that the most probable sequence of states is $s_2, s_2, s_3, s_1, s_3, s_1, s_3, s_1$.

5. Exam question: 2005, paper 9, question 8.

6. Exam question: 2008, paper 9, question 5.

7. Exam question: 2010, paper 7, question 4.

8. Exam question: 2013, paper 7, question 2.

6 Bayesian learning

1. Derive the *weight decay* training algorithm

$$\mathbf{w}_{\text{MAP}} = \underset{\mathbf{w}}{\operatorname{argmin}} \frac{\alpha}{2}||\mathbf{w}||^2 + \frac{\beta}{2}\sum_{i=1}^{m}(y_i - f(\mathbf{w};\mathbf{x}_i))^2$$

given on slide 268.

We have

$$\begin{aligned} h_{\text{MAP}} &= \underset{\mathbf{w}}{\operatorname{argmax}} \frac{p(\mathbf{y}|\mathbf{w})p(\mathbf{w})}{p(\mathbf{y})} \\ &= \underset{\mathbf{w}}{\operatorname{argmax}} (\log p(\mathbf{w}) + \log p(\mathbf{y}|\mathbf{w})) \\ &= \underset{\mathbf{w}}{\operatorname{argmax}} \left(\log \left[\frac{1}{Z_W(\alpha)} \exp\left(-\frac{\alpha}{2}||\mathbf{w}||^2\right)\right] + \log\left[\frac{1}{Z_{\mathbf{y}}(\beta)} \exp\left(-\beta E_{\mathbf{y}}(\mathbf{w})\right)\right]\right) \\ &= \underset{\mathbf{w}}{\operatorname{argmin}} \left(\frac{\alpha}{2}||\mathbf{w}||^2 + \beta E_{\mathbf{y}}(\mathbf{w})\right) \\ &= \underset{\mathbf{w}}{\operatorname{argmin}} \frac{\alpha}{2}||\mathbf{w}||^2 + \frac{\beta}{2}\sum_{i=1}^{m}(y_i - f(\mathbf{w};\mathbf{x}_i))^2 \end{aligned}$$

2. Use the standard Gaussian integral to derive the final equation for Bayesian regression

$$p(Y|\mathbf{y},\mathbf{x}) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(y - f(\mathbf{w}_{\text{MAP}};\mathbf{x}))^2}{2\sigma_y^2}\right)$$

where

$$\sigma_y^2 = \frac{1}{\beta} + \mathbf{g}^T\mathbf{A}^{-1}\mathbf{g}$$

given on slide 284.

Collecting the relevant expressions, the aim is to evaluate the (slightly re-arranged) integral

$$p(Y|\mathbf{y},\mathbf{x}) = \frac{1}{Z}\int_{\mathbb{R}^W} \exp\left(-\frac{1}{2}\left[\beta\left(y - f(\mathbf{w}_{\text{MAP}};\mathbf{x}) - \mathbf{g}^T\Delta\mathbf{w}\right)^2 + \Delta\mathbf{w}^T\mathbf{A}\Delta\mathbf{w}\right]\right) d\mathbf{w} \qquad (11)$$

where

$$Z = (2\pi)^{W/2}|\mathbf{A}|^{-1/2}\sqrt{2\pi\sigma^2}$$

using the fact that

$$\int_{\mathbb{R}^W} \exp\left(-\frac{1}{2}\left(\mathbf{w}^T\mathbf{X}\mathbf{w} + \mathbf{w}^T\mathbf{y} + z\right)\right) d\mathbf{w} = (2\pi)^{W/2}|\mathbf{X}|^{-1/2} \exp\left(-\frac{1}{2}\left(z - \frac{\mathbf{y}^T\mathbf{X}^{-1}\mathbf{y}}{4}\right)\right). \quad (12)$$

The first thing to notice is that because

$$\Delta\mathbf{w} = \mathbf{w} - \mathbf{w}_{\text{MAP}}$$

we can treat the integral (11) as being written with a $d\Delta\mathbf{w}$ rather than a $d\mathbf{w}$.[1]

If we abbreviate using $f' = f(\mathbf{w}_{\text{MAP}}; \mathbf{x})$ we can multiply out the relevant part of the integrand and re-arrange to get

$$\begin{aligned}
&\beta\left[((y - f') - \mathbf{g}^T\Delta\mathbf{w})^2\right] + \Delta\mathbf{w}^T\mathbf{A}\Delta\mathbf{w} \\
&\quad = \beta\left[((y - f')^2 - 2(y - f')\mathbf{g}^T\Delta\mathbf{w} + (\mathbf{g}^T\Delta\mathbf{w})^2\right] + \Delta\mathbf{w}^T\mathbf{A}\Delta\mathbf{w} \\
&\quad = \beta(y - f')^2 - 2\beta(y - f')\mathbf{g}^T\Delta\mathbf{w} + \beta\Delta\mathbf{w}^T\mathbf{g}\mathbf{g}^T\Delta\mathbf{w} + \Delta\mathbf{w}^T\mathbf{A}\Delta\mathbf{w} \\
&\quad = \underbrace{\beta(y - f')^2}_{z}\underbrace{-2\beta(y - f')\mathbf{g}^T}_{\mathbf{y}}\Delta\mathbf{w} + \Delta\mathbf{w}^T\underbrace{(\beta\mathbf{g}\mathbf{g}^T + \mathbf{A})}_{\mathbf{X}}\Delta\mathbf{w}
\end{aligned}$$

where the underbraces show how the terms relate to those in (12). Using (12) we can now evaluate the integral in (11) as

$$\begin{aligned}
p(Y|\mathbf{y}, \mathbf{x}) &\propto \exp\left(-\frac{1}{2}\left(\beta(y - f')^2 - \frac{4\beta^2(y - f')^2\mathbf{g}^T(\beta\mathbf{g}\mathbf{g}^T + \mathbf{A})^{-1}\mathbf{g}}{4}\right)\right) \\
&= \exp\left(-\frac{1}{2}(y - f')^2(\beta - \beta^2\mathbf{g}^T(\beta\mathbf{g}\mathbf{g}^T + \mathbf{A})^{-1}\mathbf{g})\right).
\end{aligned}$$

Here we have supressed the leading constant for reasons that will shortly become clear. Comparing this expression with the usual expression for a Normal density in one dimension with mean μ and variance σ^2

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

we see that $p(Y|\mathbf{y}, \mathbf{x})$ is a normal density with mean f' and variance

$$\sigma_Y^2 = \frac{1}{\beta - \beta^2\mathbf{g}^T(\beta\mathbf{g}\mathbf{g}^T + \mathbf{A})^{-1}\mathbf{g}}. \quad (13)$$

The leading constant must be $1/\sqrt{2\pi\sigma_Y^2}$, which is why we were able to suppress the more complex constant above.

Getting from here to the final expression is a little tricky. We have as part of the expression in (13) a tricky-looking matrix inversion. However, all good machine learners should recognize the form as one that can be tackled using the *matrix inversion lemma*, also sometimes known as the *Woodbury formula*, or to be specific a special case known as the *Sherman-Morrison* formula. This says that

$$(\mathbf{X} + \mathbf{y}\mathbf{z}^T)^{-1} = \mathbf{X}^{-1} - \frac{\mathbf{X}^{-1}\mathbf{y}\mathbf{z}^T\mathbf{X}^{-1}}{1 + \mathbf{z}^T\mathbf{X}^{-1}\mathbf{y}}.$$

[1] To see this, think of it as an area. As the integral is over the entire space the fact that you shift the variable by a constant $\mathbf{w}_{\text{MAP}}$ makes no difference. If you still doubt it, try something similar in one dimension and actually change the variable.

Applied to the inverse in (13) this gives

$$\begin{aligned}(\mathbf{A}+\beta\mathbf{g}\mathbf{g}^T)^{-1} &= \mathbf{A}^{-1} - \frac{\mathbf{A}^{-1}\beta\mathbf{g}\mathbf{g}^T\mathbf{A}^{-1}}{1+\mathbf{g}^T\mathbf{A}^{-1}\beta\mathbf{g}} \\ &= \mathbf{A}^{-1} - \frac{\beta\mathbf{A}^{-1}\mathbf{g}\mathbf{g}^T\mathbf{A}^{-1}}{1+\beta\mathbf{g}^T\mathbf{A}^{-1}\mathbf{g}}.\end{aligned}$$

So

$$\begin{aligned}\mathbf{g}^T(\mathbf{A}+\beta\mathbf{g}\mathbf{g}^T)^{-1}\mathbf{g} &= \mathbf{g}^T\mathbf{A}^{-1}\mathbf{g} - \frac{\beta(\mathbf{g}^T\mathbf{A}^{-1}\mathbf{g})(\mathbf{g}^T\mathbf{A}^{-1}\mathbf{g})}{1+\beta\mathbf{g}^T\mathbf{A}^{-1}\mathbf{g}} \\ &= \frac{\mathbf{g}^T\mathbf{A}^{-1}\mathbf{g}}{1+\beta\mathbf{g}^T\mathbf{A}^{-1}\mathbf{g}}.\end{aligned}$$

We thus have

$$\begin{aligned}\sigma_t^2 &= \frac{1}{\beta - \beta^2\left(\frac{\mathbf{g}^T\mathbf{A}^{-1}\mathbf{g}}{1+\beta\mathbf{g}^T\mathbf{A}^{-1}\mathbf{g}}\right)} \\ &= \frac{1+\beta\mathbf{g}^T\mathbf{A}^{-1}\mathbf{g}}{\beta(1+\beta\mathbf{g}^T\mathbf{A}^{-1}\mathbf{g}) - \beta^2\mathbf{g}^T\mathbf{A}^{-1}\mathbf{g}} \\ &= \frac{1}{\beta} + \mathbf{g}^T\mathbf{A}^{-1}\mathbf{g}.\end{aligned}$$

3. This question asks you to produce a version of the graph on slide 286, but using the Metropolis algorithm instead of the solution obtained by approximating the integral. Any programming language is fine, although Matlab is probably the most straightforward.

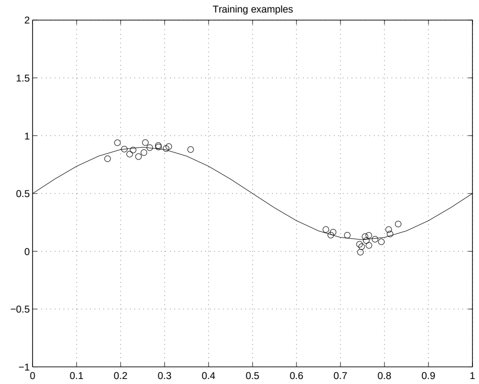
   The data is simple artificial data for a one-input regression problem. Use the target function

$$f(x) = \frac{1}{2} + 0.4\sin 2\pi x$$

   and generate 30 examples clustered around $x = 0.25$ and $x = 0.75$. Then label these examples

$$y(x) = f(x) + n$$

   where n is Gaussian noise of standard deviation 0.05. Plot the data as follows:



Let $\mathbf{w} \in \mathbb{R}^W$ be the vector of all the weights in a network. Your supervised learner should be based on a prior density

$$p(\mathbf{w}) = \left(\frac{2\pi}{\alpha}\right)^{-W/2} \exp\left(-\frac{\alpha}{2}||\mathbf{w}||^2\right)$$

on the weights. A value of $\alpha = 1$ is reasonable. The likelihood used should be

$$p(\mathbf{y}|\mathbf{w}) = \left(\frac{2\pi}{\beta}\right)^{-m/2} \exp\left(-\frac{\beta}{2}\sum_{i=1}^{m}(y(x_i) - h(\mathbf{w}; x_i))^2\right)$$

where m is the number of examples and $h(\mathbf{w}; x)$ is the function computed by the neural network with weights $\mathbf{w}$. A value of $\beta = 1/(0.05)^2$ is appropriate. Note that we are assuming that hyperparameters α and β are known, and the prior and likelihood used are the same as those used in the lectures.

Complete the following steps:

(a) Write a function `simpleNetwork` function implementing a multilayer perceptron with a single hidden layer, a basic feedforward structure as illustrated in the AI I lectures, and a single output node. The network should use sigmoid activation functions for the hidden units and a linear activation function for its output. You should use a network having 4 hidden units.

The following Matlab code implements a simple multilayer perceptron.

```
function output = mynetwork(input, w)

%
% Multilayer perceptron with a single input, four hidden units with
% tanh activation functions, and a single linear output unit.
%
% w(1:4) is weights from input to hidden units.
% w(5:8) is biases for hidden units.
% w(9:12) is weights from hidden units to output unit.
% w(13) is bias for output unit.
%

hidden_inputs = w(1:4) * input;
hidden_outputs = tanh(hidden_inputs + w(5:8));

output = (w(9:12)' * hidden_outputs) + w(13);
```

(b) Starting with a weight vector chosen at random, use the Metropolis algorithm to sample the posterior distribution $p(\mathbf{w}|\mathbf{y})$. You should generate a sequence of 100 weight vectors $\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_{100}$.

For this we need a prior:

```
function p = prior(w, alpha)

%
% Gaussian prior with specified alpha.
%

p = (((2 * pi) / alpha)^(-length(w) / 2)) * exp ((-alpha/2) * (norm(w)^2));
```

We also need to be able to compute the likelihood:

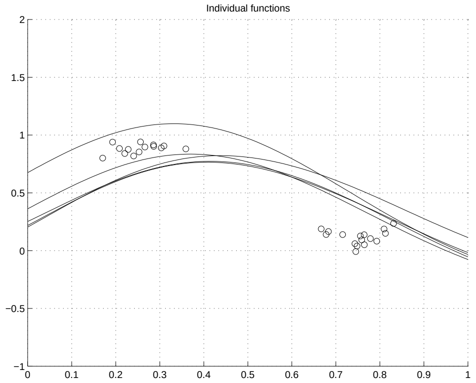
```
function l = likelihood(w, inputs, targets, beta)

%
% Likelihood calculated for Gaussian noise with beta specified.
%

N = length(targets);

error = 0;
```

```
for n = 1:N
  error = error + ((mynetwork(inputs(n), w) - targets(n))^2);
end

l = (((2 * pi) / beta)^(-N/2)) * exp((-beta/2) * error);
```

We can then implement the metropolis algorithm:

```
w = randn(13,1) * 0.01;
p_old = likelihood(w,inputs,targets,beta) * prior(w,alpha);

while counter < terms_in_sum
  old_w = w;
  w = w + (randn(13,1) * 0.04);
  total_tries = total_tries + 1;
  p_new = likelihood(w,inputs,targets,beta) * prior(w,alpha);
  if p_new > p_old
    weights(:,counter) = w;
    counter = counter + 1
  else
    if p_old ~= 0                         % Needed to avoid division by 0.
      if rand < (p_new/p_old)
        weights(:,counter) = w;
        counter = counter + 1
      else
        w = old_w;
      end
    end
  end
end
```

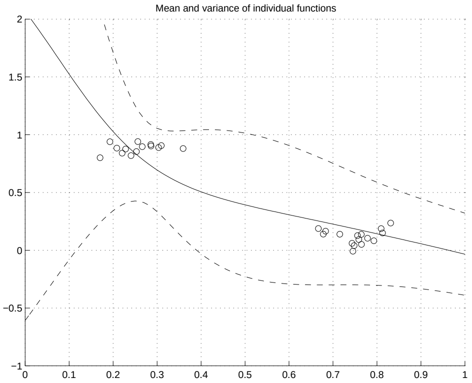
(c) Plot the function $h(\mathbf{w}_i; x)$ computed by the neural network for a few of the weight vectors obtained.



(d) Discard the first 50 weight vectors generated. Using the remainder, calculate the mean and variance of the corresponding functions using

$$\text{mean}(x) = \frac{1}{50} \sum_{i=51}^{100} h(\mathbf{w}_i; x)$$

and a similar expression for the variance. Plot the mean function along with error bars provided by the variance.



4. Can you incorporate hyperparameter estimation into your solution to the previous problem? If so, do the results make sense?

   This is straightforward using the equations derived in the lectures for estimating hyperparameters. This question represents discussion material: do the estimated values match the correct ones?

5. Explain how the Gibbs algorithm might be applied to the Bayesian network developed earlier for the *roof-climber alarm* problem.

   This is essentially straightforward as we know the joint distribution, and all the conditional distributions needed for sampling can therefore be computed directly. There is room here for discussion of how one might generate the samples, although this is beyond the syllabus for the course.

6. Slide 314 uses the following estimate for the variance of a random variable:

$$\sigma^2 \simeq \hat{\sigma}^2 = \frac{1}{n-1}\left[\sum_{i=1}^{n}(X_i - \hat{X}_n)^2\right].$$

   Show that this estimate is unbiased; that is,

$$\mathbb{E}\left[\hat{\sigma}^2\right] = \sigma^2.$$

   We have

$$\begin{aligned}
\mathbb{E}\left[\hat{\sigma}^2\right] &= \frac{1}{n-1}\mathbb{E}\left[\sum_{i=1}^{n}(X_i - \hat{X}_n)^2\right] \\
&= \frac{1}{n-1}\sum_{i=1}^{n}\mathbb{E}\left[(X_i - \hat{X}_n)^2\right] \\
&= \frac{1}{n-1}\sum_{i=1}^{n}\left(\mathbb{E}\left[X_i^2\right] - 2\mathbb{E}\left[X_i\hat{X}_n\right] + \mathbb{E}\left[\hat{X}_n^2\right]\right)
\end{aligned}$$

   Using the definition of $\hat{X}_n$ the term T_i inside the summation is

$$\begin{aligned}
T_i &= \mathbb{E}\left[X_i^2\right] - \frac{2}{n}\mathbb{E}\left[X_i\sum_{j=1}^{n}X_j\right] + \frac{1}{n^2}\sum_{j=1}^{n}\sum_{k=1}^{n}\mathbb{E}\left[X_jX_k\right] \\
&= \mathbb{E}\left[X_i^2\right] - \frac{2}{n}\sum_{j=1}^{n}\mathbb{E}\left[X_iX_j\right] + \frac{1}{n^2}\sum_{j=1}^{n}\sum_{k=1}^{n}\mathbb{E}\left[X_jX_k\right]
\end{aligned}$$

If $i = j$ then $\mathbb{E}[X_i X_j] = \mathbb{E}\left[X_i^2\right]$ and if $i \neq j$ then $\mathbb{E}[X_i X_j] = \mathbb{E}[X_i]\,\mathbb{E}[X_j] = \mu^2$ where $\mathbb{E}[X] = \mu$ as we are assuming the RVs are independent. Thus

$$\begin{aligned} T_i &= \mathbb{E}\left[X_i^2\right] - \frac{2}{n}\left(\mathbb{E}\left[X_i^2\right] + (n-1)\mu^2\right) + \frac{1}{n^2}\left(\sum_{j=1}^{n}\mathbb{E}\left[X_j^2\right] + n(n-1)\mu^2\right) \\ &= \mathbb{E}\left[X_i^2\right] - \frac{2}{n}\left(\mathbb{E}\left[X_i^2\right] + (n-1)\mu^2\right) + \frac{1}{n^2}\left(n\mathbb{E}\left[X_i^2\right] + n(n-1)\mu^2\right) \end{aligned}$$

where the last line follows because the RVs are identically distributed. Placing T_i back into the summation and re-arranging

$$\begin{aligned} \mathbb{E}\left[\hat{\sigma}^2\right] &= \frac{1}{n-1}\sum_{i=1}^{n}\left(\frac{n-1}{n}\mathbb{E}\left[X_i^2\right] - \frac{n-1}{n}\mu^2\right) \\ &= n\left(\frac{\mathbb{E}\left[X^2\right]}{n} - \frac{\mu^2}{n}\right) \\ &= \mathbb{E}\left[X^2\right] - \mu^2 \\ &= \sigma^2. \end{aligned}$$

7. Show that if a random variable has zero mean then dividing it by its standard deviation σ results in a new random variable having variance 1.

   We have a random variable X with

   $$\mathbb{E}(X) = 0$$

   and

   $$\text{var}(X) = \mathbb{E}(X^2) - [\mathbb{E}(X)]^2 = \mathbb{E}(X^2) = \sigma^2.$$

   Let $Y = X/\sigma$. Then

   $$\mathbb{E}(Y) = \frac{1}{\sigma}\mathbb{E}(X) = 0$$

   and

   $$\text{var}(Y) = \mathbb{E}(Y^2) - [\mathbb{E}(Y)]^2 = \mathbb{E}(X^2/\sigma^2) = \frac{1}{\sigma^2}\mathbb{E}(X^2) = 1.$$

8. Verify the expression in point 4 on slide 317.

   We're considering an RV X having mean μ and variance σ^2. We know that the mean of $\hat{X}_n$ is

   $$\mathbb{E}\left[\hat{X}_n\right] = \mathbb{E}\left[\frac{1}{n}\sum_{i=1}^{n} X_i\right] = \frac{1}{n}\sum_{i=1}^{n}\mathbb{E}[X_i] = \mu.$$

   and we want to know the variance of $\hat{X}_n$. So

   $$\begin{aligned} \text{var}(\hat{X}_n) &= \mathbb{E}\left[(\hat{X}_n^2)\right] - \mu^2 \\ &= \frac{1}{n^2}\mathbb{E}\left[\left(\sum_{i=1}^{n} X_i\right)^2\right] - \mu^2 \\ &= \frac{1}{n^2}\mathbb{E}\left[\sum_{i=1}^{n} X_i^2 + \sum_{i\neq j} X_i X_j\right] - \mu^2 \\ &= \frac{1}{n^2}\left(n\mathbb{E}\left[X^2\right] + n(n-1)(\mathbb{E}[X])^2\right) - \mu^2 \end{aligned}$$

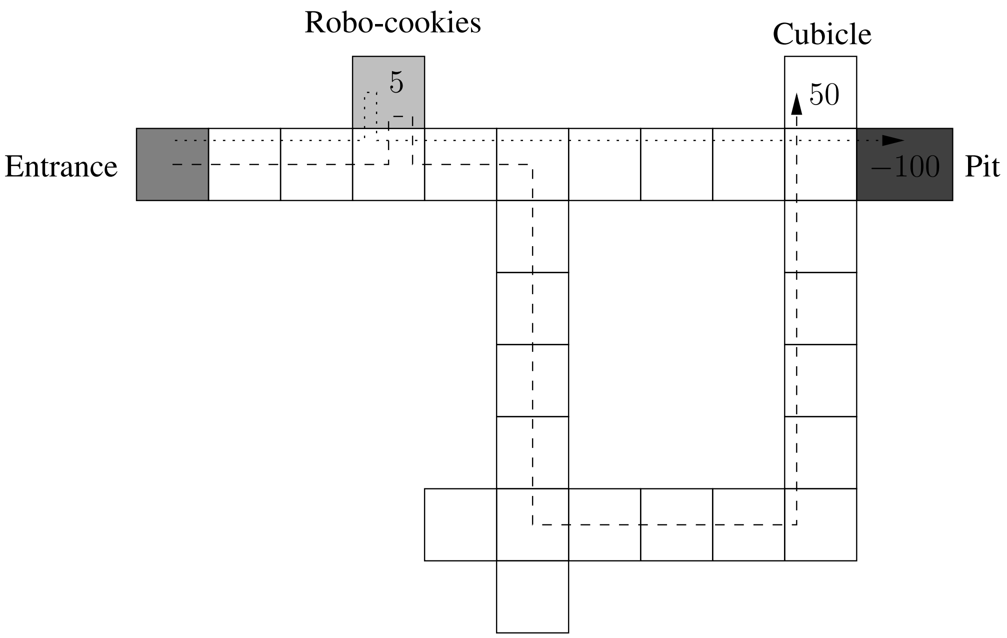
where in the last line we've used the fact that the X are independent. Now using the fact that $\mathbb{E}\left[X^2\right] - \mu^2 = \sigma^2$

$$\begin{aligned}\text{var}(\hat{X}_n) &= \frac{1}{n}\left(\sigma^2 + \mu^2 + (n-1)\mu^2\right) - \mu^2 \\ &= \frac{\sigma^2}{n} + \frac{\mu^2(1 + (n-1) - n)}{n} \\ &= \frac{\sigma^2}{n} \\ &= \frac{\text{var}(X)}{n}.\end{aligned}$$

9. Exam question: 2010, paper 8, question 2.

7 Reinforcement learning

1. Evil Robot's Dungeon of Darkness was constructed in such a way that the Pit of Endless Disgruntlement is very close to the Cubicle of Inventive Punishment. Evil Robot likes going to the Cubicle of Inventive Punishment as he gets to be nasty to a human. He does not however like falling into the Pit of Endless Disgruntlement, because that makes him very disgruntled. Between these locations and the entrance is a table with robo-cookies.



Unfortunately the part of his memory related to navigating the Dungeon has accidentally been wiped, so he can't find his way to punish the human responsible. He is running a simulation of the Q-learning algorithm to re-learn it. His actions are to move left, right, up or down one square. Assuming all Q values are initialised to 0, explain how the Q-learning algorithm operates if the dotted route is followed once, then the dashed route is followed, then the dotted route is followed again. Where no reward is indicated assume the reward is 0.

2. Exam question: 2007, paper 9, question 9.

3. Exam question: 2012, paper 7, question 2.

Appendices

A ML code for constructing planning graphs

The following is a simple ML implementation of planning graph construction. For the specific example in these problems the graph can be constructed using the following code:

```
\end{tiny}
PolyML.use "sets.sml";
PolyML.use "lists.sml";
PolyML.use "graphplan.sml";

(*---------------------------------------------------------------------------*)
(* Exercises for Artificial Intelligence II.                                 *)
(* Part 2, Question 1.                                                       *)
(* Sean Holden 2013/14.                                                      *)
(*---------------------------------------------------------------------------*)

val f1 = not (proposition "at(B,H)");
val f2 = not (proposition "at(L,B)");
val f3 = not (proposition "at(C,W)");
val f4 = not (proposition "at(C,B)");

val st0 = [f1, f2, f3, f4];

val f1' = proposition "at(B,H)";
val f2' = proposition "at(L,B)";
val f3' = proposition "at(C,W)";
val f4' = proposition "at(C,B)";

val a1 = action ((name "move(B,H)"), [f1, f2], [f1']);
val a2 = action ((name "move(C,W)"), [f1', f2', f4'], [f3']);
val a3 = action ((name "move(L,B)"), [f2], [f2']);
val a4 = action ((name "move(C,B)"), [f4], [f4']);
val a5 = action ((name "move(L,D)"), [f2'], [f2]);

val act = [a1, a2, a3, a4, a5];

val problem1 = problem (st0, act, goal [proposition "at(C,W)"]);

(* Make first expansion. *)
val stm0 = findMutex1 st0;

val (ac1,acm1,st1,stm1) = expandOnce act st0 stm0;

val (ac2,acm2,st2,stm2) = expandOnce act st1 stm1;
```

The code needs some simple functions for manipulating sets

```
(*-------------------------------------------------------------------------*)
(* Basic set operations. Set operations - assume list has no duplicates.   *)
(* Sean Holden 2013/14.                                                    *)
(*-------------------------------------------------------------------------*)

fun member x [] = false
  | member x (y::ys) = (x=y) orelse member x ys;

fun subset [] _ = true
  | subset (x::xs) ys = member x ys andalso subset xs ys;

fun setAdd x [] = [x]
  | setAdd x ys = if (member x ys) then ys else x::ys;

fun union [] ys = ys
  | union (x::xs) ys = if (member x ys)
                       then union xs ys
                       else union xs (x::ys);

fun unionConcat [] = []
  | unionConcat (x::xs) = union x (unionConcat xs);
```

and lists

```
(*-------------------------------------------------------------------------*)
(* General simple stuff for manipulating lists.                            *)
(* Sean Holden 2013/14.                                                    *)
(*-------------------------------------------------------------------------*)

(* nth element of a list                                                   *)

exception noSuchElement;
fun nth [] _ = raise noSuchElement
  | nth (x::xs) y = if (y=0) then x else nth xs (y-1);
```

```
(* Turn a list of lists into a list.                                       *)

fun collapse x = rev (foldl (fn(y, ys) =>
                        foldl (fn(z, zs) => (z::zs)) ys y) [] x);

(* Standard filter function.                                               *)

fun filter p [] = []
  | filter p (x::xs) =
    let val y = filter p xs;
    in
if (p x) then x::y else y
    end;

(* Remove duplicates from a list.                                          *)

fun removeDuplicates [] acc = acc
  | removeDuplicates (x::xs) acc =
    if (member x xs) then removeDuplicates xs acc
    else removeDuplicates xs (x::acc);

(* Remove duplicates from a list using a specified equality function.      *)

fun removeDuplicatesEq eq l =
    let fun find x [] = false
  | find x (y::ys) = (eq x y) orelse find x ys;
fun rDE [] acc = acc
  | rDE (x::xs) acc =
            if (find x xs) then rDE xs acc
            else rDE xs (x::acc);
    in
rDE l []
    end;

(* Pair an element with each element of a list.                            *)

fun pairUp x [] acc = acc
  | pairUp x (y::ys) acc = if (x=y) then pairUp x ys acc
                           else pairUp x ys ((x,y)::acc);

(* Generation of all pairs, *not* including pairs of equal elements.       *)

fun allDifferentPairs x y =
    let fun allDifferentPairs' [] _ acc = acc
  | allDifferentPairs' (x::xs) [] acc = acc
          | allDifferentPairs' (x::xs) y acc =
            allDifferentPairs' xs y (pairUp x y acc);
    in
allDifferentPairs' (removeDuplicates x []) (removeDuplicates y []) []
    end;

(* Generation of all pairs from a single list, *not* including             *)
(* pairs of equal elements. (a,b) and (b,a) can occur.                     *)

fun differentPairs x =
    let val x' = removeDuplicates x [];
    in
allDifferentPairs x' x'
    end;

(* Generation of all pairs from a single list, *not* including             *)
(* pairs of equal elements. (a,b) and (b,a) can *not* occur.               *)

fun differentPairs' x =
    let val x' = removeDuplicates x [];
fun dP [] acc = acc
  | dP (x::xs) acc = dP xs (pairUp x xs acc)
    in
dP x' []
    end;

(* Is a predicate true for at least one thing in a list?                   *)

fun trueSomewhere p [] = false
  | trueSomewhere p (x::xs) = (p x) orelse trueSomewhere p xs;

(* Is a predicate true for all things in a list?                           *)

fun trueEverywhere p x =
    let fun tE [] acc = acc
  | tE (y::ys) acc = tE ys ((p y) andalso acc);
    in
tE x true
    end;
```

The actual code is as follows:

```
(*-------------------------------------------------------------------------*)
(* Simple implementation of GraphPlan.                                     *)
(* Sean Holden 2013/14.                                                    *)
(*-------------------------------------------------------------------------*)

(*-------------------------------------------------------------------------*)
(* Types                                                                   *)
(*-------------------------------------------------------------------------*)

datatype formula = proposition of string
                 | not of formula;

fun fToString (proposition s) = s
  | fToString (not(proposition(s))) = concat ["not-",s]
  | fToString _ = "";

datatype actionName = name of string;

datatype action = persist of formula
                | action of (actionName * formula list * formula list);

datatype goal = goal of formula list;

datatype problem = problem of formula list * action list * goal;

datatype mutexType = inconsistentEff
                   | actionsInt
                   | competeForPre
                   | conflictingPair
                   | mutexPreconditions
                   | unspecified;

datatype mutex = aMutex of action * action * mutexType
               | cMutex of formula * formula * mutexType;

(*-------------------------------------------------------------------------*)
(* Basic functions                                                         *)
(*-------------------------------------------------------------------------*)

(*-------------------------------------------------------------------------*)
(* Does one formula negate another?                                        *)
(*-------------------------------------------------------------------------*)

fun negates (f1, (not f2)) = (f1=f2)
  | negates ((not f1), f2) = (f1=f2)
  | negates (_, _) = false;

(*-------------------------------------------------------------------------*)
(* In getting rid of duplicated mutexes which might result from different  *)
(* rules we use this as a definition of equality.                          *)
(*-------------------------------------------------------------------------*)

fun redundantMutexes (aMutex (a1,a2,t)) (aMutex (a1’,a2’,t’))
    = (a1=a1’ andalso a2=a2’) orelse (a1=a2’ andalso a2=a1’)
  | redundantMutexes (cMutex (f1, f2, t)) (cMutex (f1’, f2’, t’))
    = (f1=f1’ andalso f2=f2’) orelse (f1=f2’ andalso f2=f1’)
  | redundantMutexes _ _ = false;

(*-------------------------------------------------------------------------*)

fun getPres (persist(p)) = [p]
  | getPres (action(_,p,_)) = p;

(*-------------------------------------------------------------------------*)

fun getEffs (persist(e)) = [e]
  | getEffs (action(_,_,e)) = e;

(*-------------------------------------------------------------------------*)
(* Given a state level, generate the next action level and all the effects *)
(* for the next state level.                                               *)
(*-------------------------------------------------------------------------*)

(* Make sure you don’t apply persistence actions to anything.              *)
exception cantApplyPersistenceAction;

(* Does an action apply in a given state?                                  *)
fun actionApplies stateLevel (persist _) = raise cantApplyPersistenceAction
  | actionApplies [] (action _) = false
  | actionApplies stateLevel (action (n, preList, effList))
                                = subset preList stateLevel;

(*-------------------------------------------------------------------------*)
(* Apply an action if possible, getting a list of effects.                 *)
(*-------------------------------------------------------------------------*)

fun applyAction stateLevel (persist _) = raise cantApplyPersistenceAction
  | applyAction stateLevel (a as action (n, preList, effList)) =
     if actionApplies stateLevel a
     then effList
     else [];

(*-------------------------------------------------------------------------*)
(* Generate all the necessary persistence actions.                         *)
(*-------------------------------------------------------------------------*)

fun addPersistence stateLevel = map (fn x => persist x) stateLevel

(*-------------------------------------------------------------------------*)

(* Expand a state level into the next two levels.                          *)
fun expandState stateLevel actionList =
  let
    val persists = addPersistence stateLevel;
    val others = filter (actionApplies stateLevel) actionList;
    val allEffects =
        union stateLevel
              (unionConcat (map (applyAction stateLevel) actionList));
  in
    (collapse [persists, others], allEffects)
  end;

(*-------------------------------------------------------------------------*)
(* Given a state and action level, detect mutexes.                         *)
(*-------------------------------------------------------------------------*)

(*-------------------------------------------------------------------------*)
(* Do a pair of lists of formulas contain conflicts?                       *)
(*-------------------------------------------------------------------------*)

fun conflictExists x y = trueSomewhere negates (allDifferentPairs x y);

(*-------------------------------------------------------------------------*)
(* Do a pair of actions have inconsistent effects?                         *)
(*-------------------------------------------------------------------------*)

fun inconsistentEffects (a1, a2) = conflictExists (getEffs a1) (getEffs a2);

(*-------------------------------------------------------------------------*)
(* Do a pair of actions have conflicting precondition and effect?          *)
(*-------------------------------------------------------------------------*)

fun actionsInterfere (a1, a2) =
            conflictExists (getPres a1) (getEffs a2) orelse
            conflictExists (getEffs a1) (getPres a2);

(*-------------------------------------------------------------------------*)

(* Do a pair of preconditions need a mutex? *)
val preconditionsConflict = negates;

(*-------------------------------------------------------------------------*)
(* Take a list of (condition) mutexes for a state level and two actions.  *)
(* See if any pair of formulas (action preconditions) appears in a mutex. *)
(* In other words, do the pair of actions compete for preconditions?      *)
(*-------------------------------------------------------------------------*)

exception unexpectedActionMutex;
fun competeForPreconditions mList (x, y) =
    let
      val actionPairs = allDifferentPairs (getPres x) (getPres y);
      fun matchMutex (aMutex(_)) (f1’, f2’) = raise unexpectedActionMutex
| matchMutex (cMutex (f1, f2, _)) (f1’, f2’)
        = (f1=f1’ andalso f2=f2’) orelse (f1=f2’ andalso f2=f1’);
      fun cM _ [] = false
| cM [] y = false
| cM (x::xs) y = (trueSomewhere (matchMutex x) y)
                         orelse cM xs y;
    in
      cM mList actionPairs
    end;

(*-------------------------------------------------------------------------*)
(* Does an action achieve a precondition?                                  *)
(*-------------------------------------------------------------------------*)

fun achieves (persist(f)) f’ = (f=f’)
  | achieves (action(_,_,effs)) f’ = member f’ effs;

(*-------------------------------------------------------------------------*)
(* From a list of actions find all pairs that might satisfy a pair of      *)
(* preconditions.                                                          *)
(*-------------------------------------------------------------------------*)

fun allGoodPairs actions (f1,f2) =
let val allPairs = differentPairs’ actions;
in
   filter (fn (x,y) => (achieves x f1 andalso achieves y f2)
                        orelse (achieves x f2 andalso achieves y f1))
          allPairs
end;

(*-------------------------------------------------------------------------*)
(* Given a mutex and a pair of actions, are the actions mutex?             *)
(*-------------------------------------------------------------------------*)

exception unexpectedConditionMutex;
fun areMutex _ (cMutex(_)) = raise unexpectedConditionMutex
  | areMutex (a1,a2) (aMutex(a1',a2',_))
       = (a1 <> a2) andalso ((a1=a1' andalso a2=a2') orelse
                             (a1=a2' andalso a2=a1'));

(*-------------------------------------------------------------------------*)
(* Are all pairs of actions that can achieve the pair of preconditions     *)
(* mutex?                                                                  *)
(*-------------------------------------------------------------------------*)

fun actionsMutex mutexList actions (f1, f2) =
  let
    val allActionPairs = allGoodPairs actions (f1, f2);
    fun testOnePair x = trueSomewhere (areMutex x) mutexList;
  in
    trueEverywhere testOnePair allActionPairs
  end;

(*-------------------------------------------------------------------------*)
(* Having used expandState you have a start state, the possible actions    *)
(* and the potential next state. Collect together all the required mutexes.*)
(*-------------------------------------------------------------------------*)

fun findMutex1 preconditions =
let val allPairs = differentPairs' preconditions;
in
  map (fn (x,y) => cMutex(x, y, conflictingPair))
      (filter preconditionsConflict allPairs)
end;

(*-------------------------------------------------------------------------*)

fun findMutex2 actions =
let val allPairs = differentPairs' actions;
in
  map (fn (x,y) => aMutex(x, y, inconsistentEff))
  (filter inconsistentEffects allPairs)
end;

(*-------------------------------------------------------------------------*)

fun findMutex3 actions =
let val allPairs = differentPairs' actions;
in
  map (fn (x,y) => aMutex(x, y, actionsInt))
      (filter actionsInterfere allPairs)
end;

(*-------------------------------------------------------------------------*)

fun findMutex4 cMutexes actions =
let val allPairs = differentPairs' actions;
in
  map (fn (x,y) => aMutex(x, y, competeForPre))
      (filter (competeForPreconditions cMutexes) allPairs)
end;

(*-------------------------------------------------------------------------*)

fun findMutex5 aMutexes actions preconditions =
let val allPairs = differentPairs' preconditions;
in
  map (fn (x,y) => cMutex(x, y, mutexPreconditions))
      (filter (actionsMutex aMutexes actions) allPairs)
end;

(*-------------------------------------------------------------------------*)
(* Starting with a state level and its mutexes, generate the next action   *)
(* and state levels and their mutexes.                                     *)
(*                                                                         *)
(* The removal of duplicates takes care of the fact that we generally find *)
(* a mutex multiple times using different rules.                           *)
(*-------------------------------------------------------------------------*)

fun expandOnce allActions stateLevel stateMutexes =
  let
    val (actionLevel', stateLevel') = expandState stateLevel allActions;
    val actionMutexes'
          = unionConcat [findMutex2 actionLevel', findMutex3 actionLevel',
                         findMutex4 stateMutexes actionLevel'];
    val stateMutexes'
```

```
          = unionConcat [findMutex1 stateLevel',
                         findMutex5 actionMutexes' actionLevel' stateLevel'];
  in
    (actionLevel',
     removeDuplicatesEq redundantMutexes actionMutexes',
     stateLevel',
     removeDuplicatesEq redundantMutexes stateMutexes')
  end;

(*----------------------------------------------------------------------------*)
```