Exercises for Artificial Intelligence

Dr Sean B Holden, 2010-20

1 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 - `VioletScroot`
   - 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

   $$\begin{array}{c} \neg\texttt{At}(B,H), \neg\texttt{At}(L,B) \\ \boxed{\texttt{Move}(B,H)} \\ \texttt{At}(B,H) \end{array} \qquad \begin{array}{c} \texttt{At}(B,H), \texttt{At}(L,B), \texttt{At}(V,B) \\ \boxed{\texttt{Move}(V,W)} \\ \texttt{At}(V,W) \end{array}$$

   $$\begin{array}{c} \neg\texttt{At}(L,B) \\ \boxed{\texttt{Move}(L,B)} \\ \texttt{At}(L,B) \end{array} \qquad \begin{array}{c} \neg\texttt{At}(V,B) \\ \boxed{\texttt{Move}(V,B)} \\ \texttt{At}(V,B) \end{array} \qquad \begin{array}{c} \texttt{At}(L,B) \\ \boxed{\texttt{Move}(L,D)} \\ \neg\texttt{At}(L,B) \end{array}$$

   - 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 Robot. Consider the problem involving the situation calculus and Prover9 that you addressed on the previous problem sheet.

- 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

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

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);
```