CS322 Fall 1999
Module 5 (Search Issues)
Assignment 5

Due: 1:30pm, Wednesday 13 October 1999.

The aim of this assignment is to learn more about search, both advanced search techniques as well as how to represent an abstract problem as a search problem.

Question 1

The graph *mod*5*gr*1999, available from the web site in both CILog format as well as for the graph-drawing applet, is meant to be part of the road network for a city. For this graph, the aim is find a path from node *mi* to the location *cp* that can only be reached by round-about methods.

(a) Which of the following methods will find a path from *mi* to *cp* without loop detection or multiple-path pruning: depth-first search, A* search, breadth-first search, best-first search.

(b) For A* search, how much saving (in the number of nodes expanded) is obtained by using loop detection and using Multiple-path pruning? (Give the number of nodes selected from the frontier with and without each of the two pruning methods).

(c) Is a backward search more efficient than a forward search for breadth-first search or A*? Explain why.

(d) How could a bi-directional search help? Explain. What forward and backward searches would be useful?

(e) Give the distance table created by dynamic programming to find a path from *mi* to *cp*.

Question 2

Publish-on demand for textbooks and online courses is becoming more commonplace. We want to be able to deliver custom versions of cs322 for use at other places who may only want to use a subset of the modules, perhaps in different orders. Here we consider the problem of delivering a course to suit the goals of an instructor as a search problem.

Suppose *mod*(*Mod*, *Prereqs*, *Teaches*) is true if module *Mod* covers the elements of the list *Teaches* and requires that the students have already covered the elements of the list *Prereqs*.

```
mod(m1,[],[intro]).
mod(m2,[intro],[semantics,symbols]).
mod(m3,[symbols,semantics],[proofs]).
```

```
mod(m4,[intro],[search]).
mod(m5,[search,symbols],[advanced_search]).
mod(m6,[search,semantics],[csp]).
mod(m7,[symbols,semantics],[kr]).
mod(m8,[proofs],[metainterpreters]).
mod(m9,[semantics],[actions]).
mod(m10,[actions,metainterpreters,search],[planning]).
mod(m11,[search,metainterpreters],[dtlearning]).
mod(m12,[csp],[nnlearning]).
```

Suppose we decide to represent the problem of designing custom courses as a search problem where

- the nodes are lists of topics that have been covered and
- the arcs are labelled with modules. Suppose L is a node, and M is a module such that $mod(M, P, T)$, where P is a subset of L (every element of P is in L), and T is not a subset of L, then $L \cup T$ is a neighbour of L, with the arc labelled with M.

The start node is labelled with the empty list []. A goal node is a node that includes all of the topics the instructor wants to cover.

For example, suppose an instructor wants to cover *nnlearning*, and *proofs*, then any node that contains both *nnlearning* and *proofs* is a goal node. Then a solution is the path that starts with [], then has arc labelled with $m1$ to node [*intro*], then has arc $m4$ to node[1] [*intro*, *search*], then has arc $m2$ to node [*intro*, *search*, *semantics*, *symbols*], then has arc $m6$ to node [*csp*, *intro*, *search*, *semantics*, *symbols*], then has arc $m12$ to [*csp*, *intro*, *nnlearning*, *search*, *semantics*, *symbols*], then has arc $m3$ to node [*csp*, *intro*, *nnlearning*, *proofs*, *search*, *semantics*, *symbols*], which is a goal node.

(a) Draw the search graph to depth four. This should include all paths from the start node that contain three or fewer arcs.

(b) Is loop checking useful? Explain.

(c) Is multiple path pruning useful? Explain.

(d) Is backward search better than forward search for this problem? Explain.

(e) Suppose we want to use A^* search. Give a non-trivial heuristic function that is an underestimate of the actual distance from a node to a goal.

Question 3

For each question in this assignment, say how long you spent on it. Was this reasonable? What did you learn?

[1] Note that here we are representing sets of nodes as lists sorted alphabetically.