Informatics 2D: Tutorial 2

Informed Search and Constraint Satisfaction

Week 3

1 Informed Search

We saw in the lectures a graph representing the road map of part of Romania, see Figure 1. The cost of a path is the distance via the road, as given on the graph. We also have a table of straight-line distances from each town to Bucharest.

(a) Show that using greedy best-first search with the straight-line heuristic function, h_{SLD}, does not give an optimal solution when looking for a path from Arad to Bucharest.

(b) Suppose that you have the following straight-line distances from Fagaras to: Neamt 140km, Iasi 175km, Vaslui 175km, Urziceni 180km, Hirsova 230km, Giurgiu 220km, Pitesti 50km, Rimnicu Vilcea 50km, Craiova 180km, Sibiu 60km. What happens when you try to use greedy best-first search to find a path from Iasi to Fagaras?

(c) We can use A^* search in this problem; h_{SLD} is an admissible heuristic that can be combined with the actual distance of the path so far to get a new heuristic f. Show that f finds an optimal solution in part (a) and solves the problem in part (b).

2 Heuristics

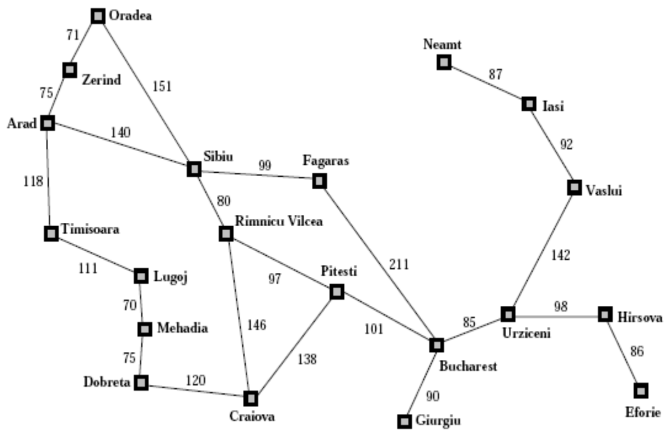
(Taken from R&N Chapter 3)

Sometimes there is no good evaluation function for a problem, but there is a good comparison method: a way to tell whether one node is better than another, without assigning numerical values to either. Show that this is enough to do a Best-First search. Is there an analog for A^*?

3 The Crop Allocation Problem

Consider the following problem in bio-dynamic farming (where some crops grow better next to particular crops)[1] for the specific land division shown in Figure 2.

[1]Adapted from an original problem set by Mellish & Fisher



Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Dobreta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

Figure 1: The Romania map from R&N with a table of straight-line distances to Bucharest

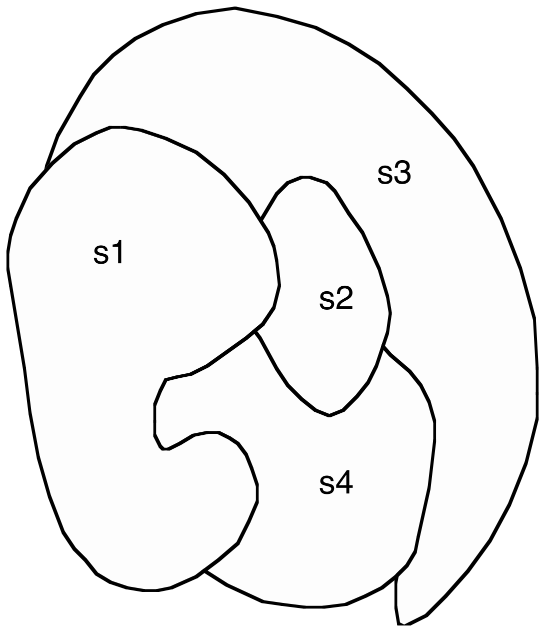


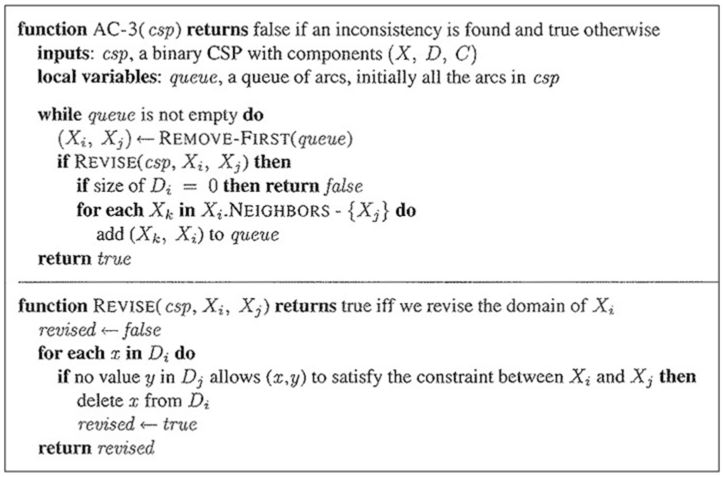
Figure 2: The Bio-Dynamic Farming Problem

The figure shows the allocation of a piece of land for planting four different crops using the constraints of bio-dynamic farming. In this kind of farming, the idea is that there are groups of crops that develop better if set in particular arrangements. Also the balance of nutrients in the soil is used to decide what to plant where. Here are the constraints according to the current levels of nutrients in the soil:

1. Sector 1 (s1) can be planted with one of the following crops: {cabbage, kale, broccoli, cauliflower }
2. Sector 2 (s2) can be planted with one of the following crops: {cabbage, kale, broccoli}
3. Sector 3 (s3) can be planted with one of the following crops: {kale}
4. Sector 4 (s4) can be planted with one of the following crops: {kale, broccoli}

The constraint here is that we do not want two sectors that are adjacent to each other to be planted with the same crops

How does this look when expressed as a constraint satisfaction problem (CSP)? What are the stages that the AC-3 algorithm goes through in obtaining arc consistency for this example? (see Figure 3 for the AC-3 algorithm)



```
function AC-3( $csp$) returns false if an inconsistency is found and true otherwise
  inputs: $csp$, a binary CSP with components $(X, D, C)$
  local variables: $queue$, a queue of arcs, initially all the arcs in $csp$

  while $queue$ is not empty do
    $(X_i, X_j) \leftarrow$ REMOVE-FIRST($queue$)
    if REVISE($csp$, $X_i$, $X_j$) then
      if size of $D_i = 0$ then return $false$
      for each $X_k$ in $X_i$.NEIGHBORS - $\{X_j\}$ do
        add $(X_k, X_i)$ to $queue$
  return $true$
```

```
function REVISE( $csp$, $X_i$, $X_j$) returns true iff we revise the domain of $X_i$
  $revised \leftarrow false$
  for each $x$ in $D_i$ do
    if no value $y$ in $D_j$ allows $(x,y)$ to satisfy the constraint between $X_i$ and $X_j$ then
      delete $x$ from $D_i$
      $revised \leftarrow true$
  return $revised$
```

Figure 3: The AC-3 algorithm

4 *More to learn[2]

1. What are the differences among A^*, greedy best-first search, and Dijkstra's algorithm?
2. Assume, an agent is to travel across a square grid from the left-top corner to the rightbottom corner. The agent can move left, right, up, down only. In terms of the number of steps, the path via the left-bottom corner (one turn) and a path that is zigzagging downwards near the diagonal of the square (2*n* - 3 turns) are equivalent. How can you modify the A^* algorithm such that a path with a smaller number of directional changes is preferred?
3. Not a recent paper, but still interesting: D. S. Nau (1983) Pathology on game trees revisited, and an alternative to minimaxing. Artificial Intelligence 21(1-2), 221–244.

[2]Starred *problems are outside the examinable course content. Feel free to ignore them completely