Midterm Solution

CS510, Fall 2012, Drexel University

Ariel Stolerman

1)

a.

It is possible to construct a knowledge-based agent that is a pure reflex agent: ***false***

(I found a definition of knowledge based agents in the beginning of chapter 7, on which I relay my answer).

A pure reflex agent works with a lookup table, like a DFA, and as such it has no memory. A knowledge based agent has to have memory – the knowledge base, on which it may rely to infer the action it should take, and that knowledge base develops in time, which means the lookup table cannot be closed as in a simple reflex agent.

b.

BFS is complete if the state space has infinite depth but finite branching factor: ***true***

BFS never expands to a lower depth of a previously expanded node, so the path lengths are monotonically increasing. Therefore if the branching factor is finite, eventually we will get to a goal node – the finite branching factor guarantees we finish every level we reach, and the properties of BFS mentioned above guarantee we never get in a loop and so bound to eventually get to a goal node (given one exists).

c.

It is sometimes useful to use A^* *for bidirectional search:* ***true***

I cannot reason why not; apply A^* from the initial state and from the goal state(s) until the fringes overlap (in the "meeting" node). We then can earn A^*'s efficiency compared to other search methods.

d.

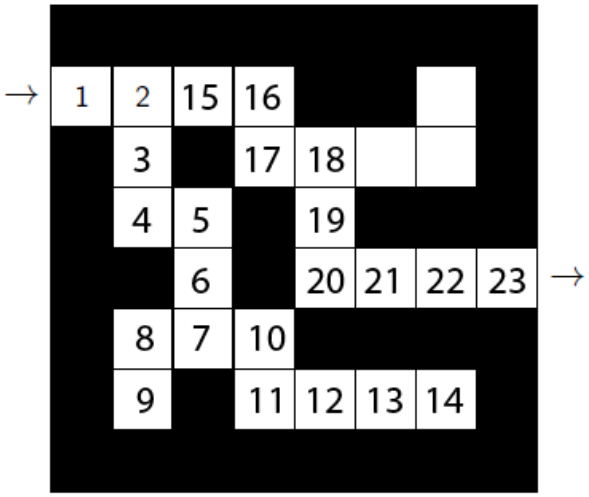
It is possible to write an exact heuristic function for the 8-puzzle (An exact heuristic function is one that always returns the exact minimum cost of getting from the current node to a goal.): ***false***

If it was exact, it wouldn't be a heuristic function, but the true cost function (and there was no point for a heuristic in that case). It is possible to enumerate all possibilities for all initial states in the 8-puzzle game and create a "heuristic" that relies on a lookup table generated from the enumeration, but I believe that doesn't qualify as a heuristic.

2)

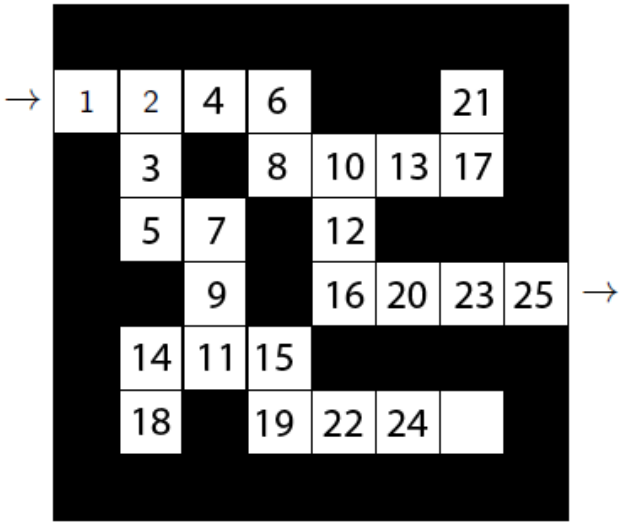
a.

Using DFS:



b.

Using BFS:



c.

The heuristic $h(n)$ presented in this question is **admissible**, since it never overestimates. The reason for that is that the real path cost (number of cells) from any cell in the maze to the exit cell is always greater than or equals to $h(n)$. Let $h^T(n)$ be a heuristic defined like $h(n)$ only on the number of *columns* between n and the exit cell, then:

$$\text{Real path cost from } n = h^*(n) \geq \text{Manhattan distance cost from } n = h(n) + h^T(n) \geq h(n)$$

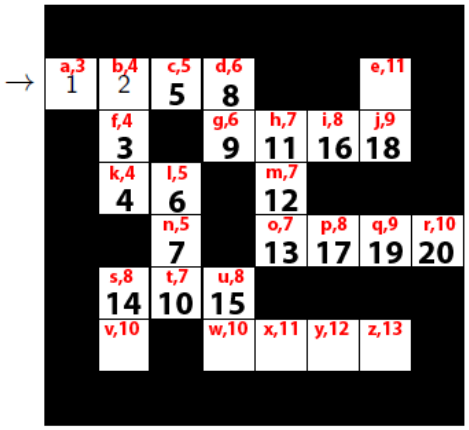
Therefore $h(n) \leq h^*(n)$, thus $h(n)$ it is admissible by definition.

d.

The red labels indicate the cell identifier (letter) and the f-value, where $f(n) = g(n) + h(n)$, $g(n)$ being the path cost thus far (with a cost of 1 to the first cell) and $h(n)$ being the heuristic from the previous section.

Comments:

- The exit cell (r) is extracted eventually not since it has the lowest f-value, but since it is the goal state.
- The precedence defined in the question harms the efficiency of the process. It seems it would have been more beneficial to sort nodes with the same value by taking the **right**most first, and if there's still a tie, take the one with the smallest h-value. Then again, it could be only in this case.




```
(a,3) (b,4) (f,4) (c,5)
(k,4) (c,5)
(c,5) (l,5) // c before l since it is older in the queue
(l,5) (d,6)
(n,5) (d,6)
(d,6) (t,7)
(g,6) (t,7)
(t,7) (h,7) // t first since it is leftmost on the grid
(h,7) (s,8) (u,8) // s before u since it is leftmost
(m,7) (s,8) (u,8) (i,8) // s,u before i since they are leftmost
(o,7) (s,8) (u,8) (i,8)
(s,8) (u,8) (i,8) (p,8) // i before p since it is older in the queue
(u,8) (i,8) (p,8) (v,10)
(i,8) (p,8) (v,10) (w,10)
(p,8) (j,9) (v,10) (w,10)
(j,9) (q,9) (v,10) (w,10) // j before q since it is older in the queue
(q,9) (v,10) (w,10) (e,11)
when q is extracted and searched, we immediately get to r, the exit node
```

3)

a.

It might not make sense to encode this problem as a COP since naturally we would have the quantities we take from each of the k items as our variables. If so, the finite discrete domains for any variable v_i would be $\left\{0,1,2,\dots,\left\lfloor\frac{cart\ weight\ limit}{weight\ of\ item\ i}\right\rfloor\right\}$, which may potentially be very large and cause the sum function we want to optimize over all functions $f_{ij}: D_i \times D_j \to \mathbb{N}$ have a very large domain with a potentially large range as well. Therefore we might end up with a very large search tree for the problem, which can be infeasible for a practical solution.

b.

One heuristic that can be used for selecting the next item to be added to the cart is the one that takes the most relatively profitable item that can still fit in the cart without passing the weight limit, i.e. the item i that has the highest $\frac{value\ of\ i}{weight\ of\ i}$ ratio where $current\ cart\ weight + weight\ of\ i \leq max\ weight\ limit\ of\ cart$.

Note: the heuristic is simply the value-per-pound; the backtracking takes care of the cases we pass the weight limit by taking us back one step and on to the next best choice.

The reason this heuristic is good is that it is a "greedy-best-value" heuristic, meaning it locally maximizes the "value-per-pound" of the current content of our cart.

The reason this heuristic does not guarantee optimality is that it may be the case that it would be better for us to choose some item that fills our cart completely with not the best value-per-pound, but total absolute value better than the greedy solution, that might take us to a better value-per-pound for the total contents of our cart, but a smaller absolute value, which is eventually what we care about.

c.

(To my understanding the chromosome addresses the string representation of the population.)

Using a messy genetic algorithm for the problem in the question can be useful, if the strings are a sparse representation of the states, i.e. encoded something like $\{(item\ i_1, quantity), \dots, (item\ i_l, quantity)\}$ where $i_1, \dots, i_l$ are the indices of the l of k items taken in this state and the corresponding quantities taken from them. This way combining two states together can create a larger chromosome (overlapping items should be accumulated in the resulting chromosome).

This is different from, say, the n-queen problem, as in the n-queen problem all n queens should be assigned a position in every state, resulting with a single-length chromosome whose length cannot be changed. Here since there might be items we will not take at all, a messy genetic algorithm can work.

The maintenance of the above is a bit harder than in a non-messy genetic algorithm, however it can increase space efficiency and may be preferable over the non-messy version (depending mostly on the size of k).

4)

This search problem may not be a good candidate for simulated annealing search because the large branching factor and the small-range heuristic mean that $\Delta E = value(next) - value(current)$ is relatively limited. The distribution of heuristic evaluations of nodes imposed by h is coarse, so values that would have been differentiated by a finer heuristic are now placed in the same bin. This means the probability a bad $next$ (worse than $current$) node is chosen, which is $e^{\frac{\Delta E}{T}}$, is potentially higher for bad nodes than it would have been if h had a broader range. Taking the Ping-Pong ball in the bumpy surface analogy from the book, this means we shake the surface "less accurately" or "less considerable of the current position of the ball" trying to get the ball in the deepest spot on the surface.

5)

a.

There is only one pure-strategy Nash equilibrium in the game given in the question: (A, D) (i.e. player 1 always chooses A, and player 2 always chooses D).

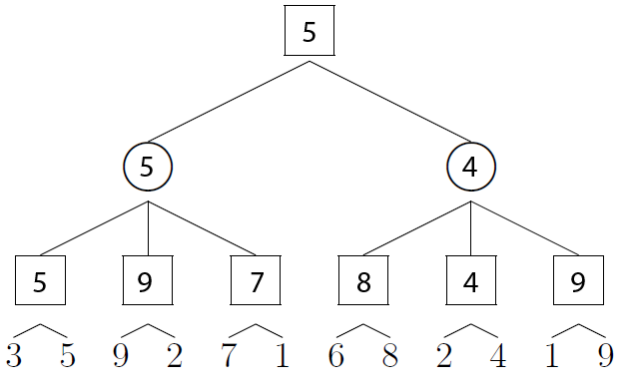
b.

The (only) Nash equilibrium in the question is **not** Pareto optimal, since there exist strategies that provide all of the players a higher profit, for instance the following pure strategies: (B,E), (B,F), (C,E) and (C,F) (and there are more non-pure strategies.)

6)

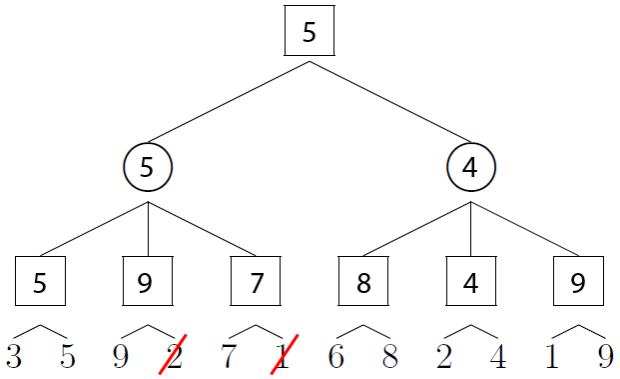
a.

Following is the tree filled out where boxes represent the MAX nodes and circles represent the MIN nodes:



b. **Correct answer:** should also cross out 2 rightmost leafs (1 and 9)

Assuming the nodes are scanned left-to-right, following is an illustration of the leaf nodes that would not be evaluated (crossed-out):



7)

a.

The circumference is $2\pi r$ and the diameter is $2r$, so the answer is (pumpkin) π!

b.

The following LISP code:

```
((lambda (x) (list x (list 'quote x))) '(lambda (x) (list x (list 'quote x))))
```

returns a list which is represented as a string that is identical to the code (i.e. a self-printing code; That's actually an implementation of the recursion theorem we learnt about in CS525).

It does it by applying a function (using the lambda notation) that lists the given parameter and a quote of the given parameter on a quote of itself. Since the argument is a quote, the ` sign is removed when printed out making the output identical to the code. Pretty neat!

c.

That's an oldie but a goodie: because DEC 25 == OCT 31 ☺ (i.e. $2 \cdot 10^1 + 5 \cdot 10^0 = 3 \cdot 8^1 + 1 \cdot 8^0$)