Instituto Superior Técnico

Artificial intelligence and decision systems

Mini-project 1

Filipe Novais 75352

Pedro Martins 75565

2^{st} of November of 2016

2.

(a) Problem Representation

After reading the data file, all the variables, including *casks*, *stacks* and *edges*, are stored in lists of classes. We also interpret the data and create a graph that is stored in a class.

To represent the state, we use a string with three elements separated by spaces. The three elements constituing the state are the current localization of the CTS in the graph, the cask that CTS is carrying at that moment and the *stackstate*. If the CTS is not carrying any cask the value of the variable *cask* in the string is $'0'$. The *stackstate* consists in a substring that incorporates all the stacks and the casks stored in each stack. Is represented by: $state = 'localization\ cask\ stackstate'$.

The operators of our problem are $Move$, $Load$ and $Unload$. The $Move$ is an operation that can be done in all circunstances. The $Load$ is an operation that can be performed when the *statelocation* is a *stack* that has casks stored in and the state *cask* is '0'. After the $Load$ operation the state is updated changing the *state cask* to the id of the *cask* loaded and the *stackstate* is also updated. The $Unload$ is an operation that can happen when the *statelocation* is a *stack* that has enough free space for the *cask* that is in the CTS and the CTS has to be carrying a *cask*. After this operation the state cask changes to $'0'$ and the stackstate is updated.

In the initial state, the state location is $'EXIT'$, the state cask is $'0'$ and the *stackstate* is the original one: $Initial\ state = EXIT\ 0\ stackstate$. In the goal state, the state location is also $'EXIT'$, the *state cask* is the cask goal and the *stackstate* is the final one: $Goal\ state = EXIT\ caskgoal\ stackstate$.

To calculate the cost we did an update to it in every operation, being that, the cost of the $Move$ is the length between the current node and the child to which the CTS is moving when the CTS is not carrying a cask and $length \times (1 + caskweight)$ when the CTS is loaded. The cost of the $Load$ and $Unload$ operations is $1 + caskweight$.

(b) Search algorithms

For the uninformed search, we chose the uniform cost search because is the only one that guarantees optimality. For example, the breadth-first search is only optimal if all the step costs are identical. In this algorithm we expand the lowest cost node, having the node highest priority in our fringe that is implemented using the python function *priority queue*.

For the informed search, we chose the A* algorithm because this algorithm keeps searching for the solution with lowest until it finds the optimal solution. None of the other informed search algorithms could do it better. The performance of this search depends on the quality of the heuristics function used.

(c) Heuristics

To implement our heuristic function, we, first, run the uniform cost search, being the start every location in the graph and the goal the stack where the cask goal is stored and we save the costs obtained, that correspond to the distance, divided by two, in an auxiliary list, *heuristics_no_cask_goal*. Then, we run again the uniform cost search, being the start every location in the graph and the goal the $'EXIT'$ and we save the costs obtained, that correspond to the distance, divided by two, in another auxiliary list, *heuristics_cask_goal*.

In the heuristics function, we use the lists of distances, mentioned before, to calculate the heuristic value for each child of the current state. To do that we use the following expression:

$$g[node\_index].heuristics[i] =$$

$$0.1 \times num1 \times heuristics\_cask\_goal[node\_index][i] + num2 \times heuristics\_no\_cask\_goal[node\_index][i]$$

Where the *node_index* is the index, in the graph class, of the current node in the graph represented in the stack. i is the index of the childs of that node in the graph. $num1$ and $num2$ are two variables that depend on the current state, more specifically, when the *state cask* is the *cask goal*, $num1 = 1$ and $num2 = 0$ and when the *state cask* is not the *cask goal*, $num1 = 1$ and $num2 = 2$.

(d) Comparison between search methods

		s1.dat Cc	s3.dat C0	g-s10d4-L1.dat C0	g-s10d4-L2.dat C1
Uniform cost	Nodes explored	262	1451	631968	7621
	Time (s)	0.0074	0.0691	17.8522	0.1842
A*	Nodes explored	259	1435	335878	6960
	Time(s)	0.0130	1.7045	13.3156	0.4114

Table 1: Comparison between uniform cost and A* searches

We did a comparison between two methods by the running time of the program and the number of nodes explored. As we can see in the table, the number of nodes is always less using the A* method. In some less complex cases the reduction is not that noticed but in the most complex case, *g-s10d4-L1.dat C0*, the number of nodes explored is almost half. As for the time, it was only reduced in the most complex case. We also verified that the solutions for this cases are the same for both search methods.