CSC384: Introduction to Artificial Intelligence, Fall 2014
Assignment 1 Clarifications

1 General Clarifications

If you are having performance issues and are using an old copy of cyclecheck.py, note that the latest version available corrects the issue.

1.1 Hashable state function

The hashable state function is used to check if two states are equal. This is needed for cycle checking. Your function should return a tuple that, when checked value by value, is equivalent to that returned by another state if and only if the states are in fact equivalent.

Note that lists are not hashable. Tuples are hashable however, and a list can be converted to a tuple using the following:

```
tuple(targetlist)
```

1.2 Heuristic performance

We can judge the performance of heursitics by the amount of nodes we must expand and the total time taken to solve a problem. If the heuristic is admissible, then we find an optimal solution and do not need to worry about the cost of the solution found. Nodes explored is a good indicator of how much work we skip, time taken moderates that by factoring in the computation time spent using the heuristic. For simple problems time taken may be too small to judge well.

Performance wise you will not be expected to produced exceptional results, but your heuristic should show improvement in some cases. If your code takes an excessive amount of time to run (far exceeding the necessary amount of time), your process will be killed and you will be penalized.

2 Question 1 Clarifications

The heuristic you produce does not have to be exceptional, but it should in some case outperform the Manhattan distance heuristic. Your heuristic should not be strictly worse than the Manhattan distance heuristic.

Let h_m be the Manhattan distance heuristic and $h_0 = 0$ be the uniform cost heuristic. Since $h_0(s) \leq h_m(s) \leq h^*(s)$ for any state s, h_m is strictly better than h_0. While the heuristic you create (h_c) does not have to be admissible (though it is certainly not a bad quality!), it should not be the case that the Manhattan distance heuristic is strictly better: $h_c(s) \leq h_m(s)$ for all states s.

The cost of all actions in question 1 is 1. This is not a cost in fuel, we are trying to minimize the number of actions taken. Fuel is a seperate concern that we must model to ensure that the plans we generate conform to the problem specification. Fuel does not directly factor into cost, but will often factor indirectly by forcing the use of more actions.

You do not consume fuel to refuel the robot. If you end up at a fuel station with 0 fuel you may still refuel the robot.

3 Question 2 Clarifications

There is a mistake in one section of the problem description when it mentions *earliest start time*. There should be no *earliest start time*, only *earliest completion time* associated with jobs.

Performing a job is an atomic action. When you peform a job, you go immediately from choosing a job to a state in which the job is finished. Ideally at no point should you have a state in which a job is processing.

The cost of action is the late penalty incurred by the action. Configuration actions advance time, but have no cost themselves. Advancing the time may change heuristic values though.

The goal state is to complete all jobs. A job may not be performed twice, once complete that action can no longer be taken.

3.1 Waiting

Since the cost of a configuration action is 0, with a bad heuristic the system could consider a plan which consists of nothing but cycling through different configurations. You may optionally stipulate that after performing a configuration action a perform job action must be taken before another configuration action is taken. This would need to be encoded in the state (and thus hashable state function).

Due to the earliest completion time constraints it may be the case that no job can be performed at the current point in time. If you make the above modification you can no longer pass time by switching through configurations. You are allowed to add a wait action if you want, either to compensate for the above issue or as a more elegant solution to the issue of passing time. The design of such an action is up to you.

All test cases used will assume that you do not have to wait to finish a job, but these modifcations and may make processing more efficient.