Computer and Information Sciences
University of Strathclyde

Week 4 – The MIU System, part 3 (worth 6% of your overall grade)

Aim

This this lab is a continuation of your previous lab, optimising your breadth first search, and implementing the A* algorithm. If you have not completed the previous lab, you should complete those first. All the rules from previous weeks apply to this week too. This is the most challenging lab, and there is not a simple solution.

The MIU System

Last week you implemented your first uninformed search algorithms, breadth first, depth limited, and iterative deepening search. In this practical you will improve your breadth first search by implementing a hash map to store states, and you will also perform heuristic search using the A* algorithm.

Breadth-first search using a dictionary – `breadth_first_dictionarysearch` (p) (worth 2%)

You will have noticed that breadth-first search keeps a lot of paths on the agenda, and that the size of the agenda quickly grows as the search progresses. If the solution path contains many steps, then breadth-first search can fill the memory up with incomplete paths stored on the agenda before it finds a solution.

- However, we notice that the first time breadth-first search encounters a new node, it will be at the smallest possible depth. This means we don't have to store the entire path: when we encounter a new node, we only need to remember its ancestor. When we find the goal node we can then extract the solution by following the ancestors all the way up the tree from the goal node to the initial state. If you aren't sure what this means, then ask!

- Implement a version of breadth-first search based on this idea, keeping only the next state on the agenda rather than whole paths (i..e you do not need extend_paths(), and compare the performance of this implementation with your original breadth-first search where you stored paths. It might be useful to keep this in a separate function.

- Each time a new node is encountered, you should store it in a hashmap. The best way of doing that is with a dictionary (https://www.w3schools.com/python/python_dictionaries.asp). You should then only add the next state to the agenda, rather than the entire path.

- When a solution is found, you should use the dictionary to reconstruct the path given the solution string, and then return this path

- Using a dictionary, you can also create a visited list, so that you no longer expand nodes that have already been expanded. Implement this.

```
breadthfirstdictionarysearch("MIUUIUUII")
Number of Expansions: 2227
Max Agenda: 17542
Solution Length: 9
```

```
Solution: ['MI', 'MII', 'MIIII', 'MIIIIU', 'MIUU', 'MIUUIUU',
'MIUUIUUIUUIUU', 'MIUUIUUIIUU', 'MIUUIUUII']
```

Heuristic Search A* – astarsearch (goalString) (worth 2%)

You have previously developed uninformed solvers for MIU. Here, we can add a heuristic function and use best first and A* search

- First, create a function which will return a rough estimate of the number of steps needed to transform the current string into the goal string. Use the header:
  `def estimate_steps(current, goal):`

- Initially, keep it very simple. Implement a stub (i.e. a placeholder) to begin with - just return 1 if current is different from goal and 0 if they are the same.

- You can return and improve this function after your first implementation of A*

- Next, implement A* search with visited list, following the guidance in the lectures. Use agenda-based search as you have done so far, but instead of picking the first item from the agenda, you pick the one with the lowest value of the sum of the heuristic function. Your implementation will be very similar to the previous search you implemented.

- When we refer to cost, we refer to the length of the path (i.e. how many moves to go from MI to current state)

- For A* search, pick the state from the agenda which has the lowest value of pathSoFar(initial,current) + estimateSteps(current,goal). In the case of a tie (i.e. two identical values on the agenda), you can choose the highest item on the agenda, i.e. the leftmost value.

- Compare your values to BFS values. Would you expect to see big differences?:

```
breadthfirstdictionarysearch("MIUUIUUII")
Number of Expansions: 1731
Max Agenda: 14155,
Solution Length: 9
Solution: ['MI', 'MII', 'MIIII', 'MIIIIU', 'MIUU', 'MIUUIUU',
'MIUUIUUIUUIUU', 'MIUUIUUIIUU', 'MIUUIUUII']

breadthfirstdictionarysearch("MUIU")
Number of Expansions: 11
Max Agenda: 14,
Solution Length: 9
Solution: ['MI', 'MII', 'MIIII', 'MIIIIU', 'MUIU']
```

Improving Steps Estimate – estimateSteps (current, goal) (worth 2%)

Try to implement a better version of estimateSteps(current,goal). It needs to be fast to compute, but as accurate as possible. If you want A* to return the optimal plan, it also has to be admissible.

- The challenge is to come up with a heuristic such that A* can find a length 11 derivation of the string "MUUUIIIII" with less than 10000 expansion steps.
- Sadly there is no "good" heuristic for the MIU system but I have added the description of a simple MIU algorithm at the end of this sheet, which might help with designing a heuristic that will somewhat improve the search.

- Important: Your heuristic function should be a function that leads generally to improved results, over you're a* value, not a hard-wired, constant function that is tailored to a specific string

- There is no "right" answer here, it is a challenge for you to think about!

- Any well explained heuristic that doesn't solve the above problem, but can solve other problems more quickly will still be awarded marks.

Submission Instructions

- This practical is worth 6% of the mark for the class, and will be assessed by demonstration in weeks 4 and 5. When you are ready, call a lab demonstrator over, and they will check your code and ask you questions.

- You should ensure that you understand your code before you demonstrate it. To earn the marks, you should not just have functional code, but also be able to explain it to the satisfaction of the demonstrators.

Computer and Information Sciences
University of Strathclyde

Algorithm Guidance Instructions: Comparison with a simple linear time algorithm

You may have already noticed that there is a way to construct strings of MIU, without doing any searching. The method is as follows:

1. Count the number of I’s in your target string, counting each U as 3 I’s.

2. If the target number of I’s is of the form 3n, where n is an integer, then stop – you cannot make this string.

3. If the target number of I’s is of the form 3n+1, then use the doubling rule to make a string of I’s which is of the form 3m+1, where m is an integer and 3m+1 ⩾ 3n+1. In other words, keep applying the doubling rule, starting from MI, until the length of the string of I’s is greater than or equal to the target string AND the constructed string of I’s produced by the doubling rule is of the form 3m+1.

4. If the target number of I’s is of the form 3n+2, then use the doubling rule to make a string of I’s which is of the form 3m+2, where m is an integer and 3m+2 ⩾ 3n+2. In other words, do the same as above, but test for the form of the constructed string being 3m+2.

5. Trim the constructed string to the required length using Rules 1 and 4.

6. Apply Rule 3 where necessary to create the target string. Here is an example of the algorithm creating the string MUUI:

MUUI contains 7 I’s. 7 is of the form 3n+1, so use the doubling rule until we’ve made a string at least as long as this, which is of the form 3n+1. We make: 1, 2, 4, 8, 16 and then stop as 16 ⩾ 7 and 16 is of the form 3m+1. We need to trim 9 I’s from this string, so add one U to it (adding 3 I’s) and then apply Rule 4 twice (which subtracts 12). Then apply Rule 3 twice to get MUUI, as required.

MUUI = MIIIIIII = 3n + 1

Start with MI, apply doubling:
MI : MII : MIIII : MIIIIIIII : MIIIIIIIIIIIIIIII = 16 I’s = 3n +1 and also >7

Need to trim 9 I’s, so add a U: MIIIIIIIIIIIIIIIIU
Can replace III with U: MIIIIIIIUUUU
Subtract UU: MIIIIIII
Add U’s: **MUUI – GOAL!**