Kefan Yang
Kefany@sfu.ca

1. PEAS Model

(1) Performance Measure: efficiency, robustness, immediacy
Environment: road status in Vancouver, bus schedules
Actuators: mobile APP, website,
Sensors: GPS, satellites, input box (to get user input)
Environment Type: observable, nondeterministic, episodic, dynamic, continuous, single-agent

(2) Performance Measure: correctness of answers, amount of money you get
Environment: questions, other competitors
Actuators: your mouth and brains
Sensors: eyes and eyes to see and hear questions
Environment Type: partially observable, deterministic, episodic, static, discrete, multi-agent

2. Problem Formulation

(1) State: an arrangement of the colors (may have conflicts)
Action: change the color in a certain area into another color
Transition model: an action changes the current state into another state because the arrangement of colors is changed.
Goal test: the arrangement of colors has no conflict
Path cost: 1 for each action (optional)

(2) State: a set contains all measures we have
Action: add or subtract two measures in the set to produce a new measure
Transition model: an action adds a new measure into the set to get a new state
Goal test: 1 gallon is in the set
Path cost: 1 for each action (optional)

3. Searching Using Composite Actions

Super-composite actions do not speed up problem solving. Composite actions help us to ignore unimportant details, such as releasing the brake in the Go(Sibiu) action (this is unimportant because the problem doesn't care when the brake is released). In contrast, a super-composite action decreases the steps we need to take to get the goal, but increases the number of actions we can choose in each step. In this way, the cost of searching remains the same.

4. Lights Out Puzzle

4.1

Please see the source code 4.5.py3

4.2

No, the order doesn't matter. For instance, if a cell is toggled for 3 times, its current state is different from the original state, no matter in which order you toggle it.

It's no use to run perform_move() twice on a cell. Since the order of operation doesn't matter, running perform_move() twice on a cell is equivalent to not running perform_move() at all. The state of the adjacent cells remain the same.

4.3

H(x) = (#ON cells)/5

This heuristic function is admissive because each perform_move() operation can increase at most 5 OFF cells, so the number of operations needed is always greater than the heuristic function.

4..4 Please see the source code 4.5.py3

4.5

Algorithm	Total Runtime	Total Solution Length	Total Steps
DFS	17.927 sec	1000	217778
BFS	22.512 sec	440	754114
AStar	521.187 sec	440	68960

4.6

Input Size	Total Runtime	Total Solution Length	Total Steps
5	0.008 sec	1510	1902
8	1.654 sec	3116	394472

For the 5*5 puzzle, we use an algorithm known as ***Chasing the Light***. This algorithm is fast against the previous searching algorithms, but it's not optimal and it doesn't apply to 8*8 puzzles (in an 8*8 puzzle, the number of cases is very large after you perform the first round of light chasing, whereas there are only 7 cases for a 5*5 puzzle)

For the 8*8 puzzle, we use a more general version of the previous algorithm. Instead of explicitly specify how to toggle the first row in each case, we use brute force DFS to toggle the first row and see if the light chasing can give us a correct result. This gives a time complexity of $O(n^2 \cdot 2^n)$ instead of $O(n^2)$

5. Feedback

(1) The most interesting experience for me is to solve puzzles with what I've learned in class.
(2) It's hard to understand the pseudocode in the slides. I think some simple algorithms are more difficult to understand in this way.
(3) I hope we can have more opportunities to apply the algorithms we learned in the class.

6. Time

More than 5 hours