UNIVERSITY OF MICHIGAN

ROB422/EECS465
Introduction to Algorithm Robotics

HW3 - Motion Planning

Junhao TU

October 26, 2023

Question 1

a.

- **State space**: pair of possible cities (i, j).
- **Successor function**: the successor of (i, j) are all (m, n), which are adjacent to (m, i) and adjacent to (n, j).
- **Action**: each of people move to adjacent cities.
- **Action cost**: the cost from (i, j) to (m, n) can be regarded as $max(d(i, m), d(j, m))$.
- **Goal**: achieve the state space (i, i), which is the city i.

b. The heuristic $D(i, j)/2$ is admissible since it doesn't overestimate the true cost when two individuals, starting an odd number of steps apart, move towards each other in equal steps, effectively halving the distance between them.

c. It is possible. Imagine a scenario with just two nodes connected by a single edge, where two friends, starting from these nodes, repeatedly swap positions. Such a situation arises when they start an odd number of steps away from each other.

d. Certainly. By selecting an unsolvable scenario from question c and introducing a self-loop where one friend moves in a way that alters the distance by 1, we can render the situation solvable.

Question 2

a. Hill Climbing Search. This variant focuses on the single best state at each step, akin to hill climbing, which evaluates and selects the best neighboring state.

b. Breath-first Search. This approach explores all possible states emanating from a single state, uniformly at each level, similar to breadth-first search which expands all child nodes at a particular depth before moving to the next.

c. Hill Climbing Search. With a temperature of zero, the algorithm rejects all downward steps, effectively mirroring the behavior of hill climbing where only the same or better states are chosen.

d. Random search. Infinite temperature leads to the acceptance of all proposed new states, regardless of their quality. This behavior aligns with a random search, where the selection of the next state is entirely random and uninfluenced by solution quality.

e. Random search. With only a single individual in the population, genetic variation through crossover is non-existent. Variation can only arise through mutation, which, depending on its nature and frequency, essentially turns the process into a random search with a mutation mechanism.

Question 3

The configuration space (C-space) for a cylindrical rod that can translate and rotate in $\mathbb{R}^3$, disregarding rotation about its own axis, is $\mathbb{R}^3 \times \mathbb{RP}^2$. Here, $\mathbb{R}^3$ accounts for 3D translations and $\mathbb{RP}^2$ represents the rod's orientation excluding rotations about its longitudinal axis. The total dimension of this C-space is 5.

Question 4

The configuration space (C-space) for five independent polyhedral bodies each capable of translating and rotating in 3D space is $(\mathbb{R}^3)^5 \times (\mathbb{SO}(3))^5$. The dimension of this C-space is 30, with each body contributing 3 dimensions for translation and 3 dimensions for rotation, totaling $5 \times (3 + 3) = 30$.

Question 5

- **Implicit, High-Resolution Grid:** C-space is discretized into a grid, much like dividing a physical space into tiny cells. Examples include BFS, DFS, Dijkstra, A*, Visibility graph. In this approach, the grid can be stored as an array or matrix, making access and modifications fast. Also, the solution's optimality and completeness depend on the grid's resolution. It is simple but may suffer from memory and computational inefficiencies, especially in higher-dimensional spaces.
- **Growing Search Trees Directly on the C-space:** Instead of discretizing the C-space, this method explores the C-space by growing trees. Examples include the RRT and its variants. It can handle higher-dimensional spaces without incurring the same computational penalties as grid-based methods. However, the paths produced may not be the most optimal and it might require more complex data structures and algorithms.

Implementation 4

i. `astar_template.py`

Table 1: Comparison between "4-connected" and "8-connected" neighbors

	"4-connected" neighbors	**"8-connected" neighbors**
Computation time [s]	224.574	144.201
Action cost	12.521	10.74

In terms of computation time, the '8-connected' neighbors are faster than the '4-connected' one because it allows diagonal movements, leading to shorter paths and fewer nodes to explore to reach the goal. This efficiency comes from being able to move directly towards the goal and cover more area per step.

In terms of path cost, the path cost in the '8-connected' neighbors is also shorter because it allows diagonal movements, enabling more direct paths to the goal. This reduces the total

number of steps compared to ‘4-connected‘, which is restricted to horizontal and vertical movements only, often resulting in longer, less direct paths. Another possible reason for the shorter path cost, beyond the allowance of diagonal movements, could be the heuristic used in the A* algorithm. If the heuristic closely matches the actual movement options allowed (including diagonals), it tends to guide the search more efficiently towards the goal. This alignment between the heuristic and the movement capabilities can result in finding shorter, more optimal paths.

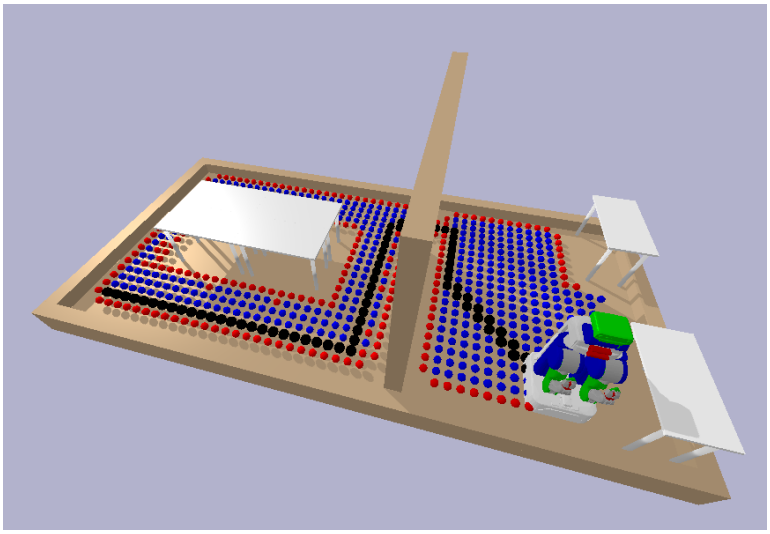

Figure 1: A* algorithm with “4-connected” neighbors

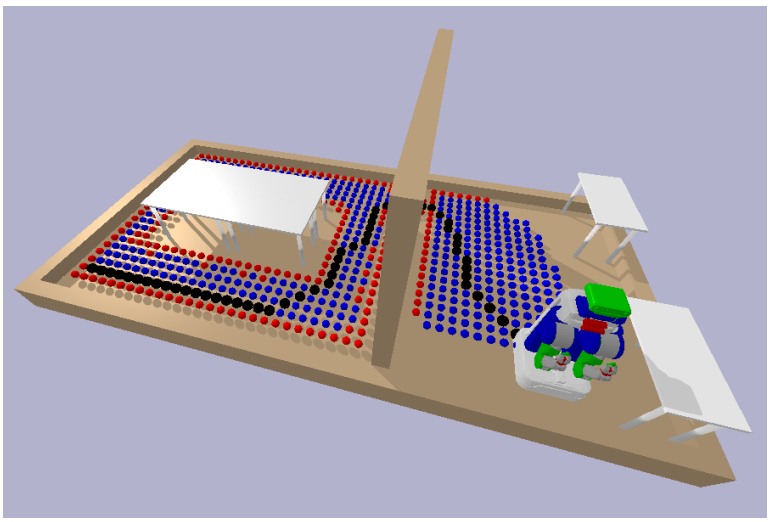

Figure 2: A* algorithm with “8-connected” neighbors

Implementation 5

i. `rrt_templete.py`

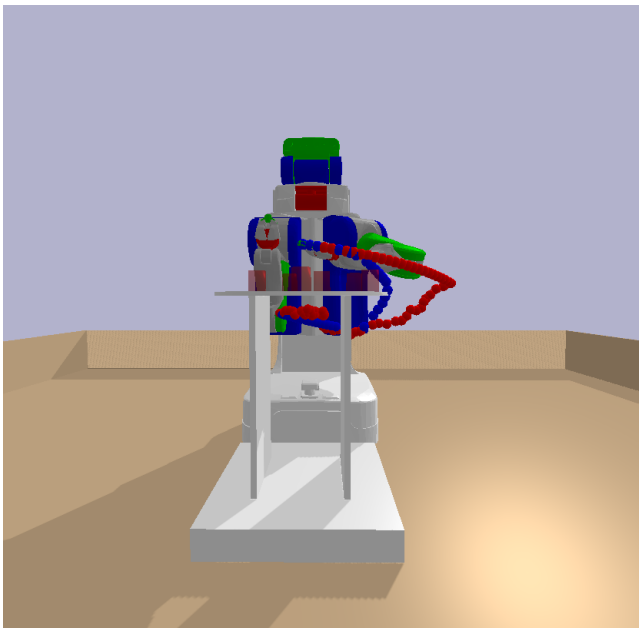

Figure 3: RRT-Connect algorithm (red) and shortcut smoothing algorithm (blue)