Experimental report for the 2021 COM1005 Assignment: The Rambler's Problem*

YiLong Jia

In this report, we firstly describe the implementation of Branch and Bound (BB) and A* search. Then, we will evaluate the efficiency of these two approaches in different maps and evaluation functions based on the presented results.

1 Description of Branch-and-Bound and A* implementation

My BB search will explore the node with the lowest accumulated cost, it uses an **open list** to keep track of nodes that yet to explore, and it is sorted based on the accumulated cost. There is also a **closed list** that keeps track of nodes are already visited. Then, BB search will continue to iterate all nodes in the given procedure till it reaches the goal.

In addition, A* search algorithm builds on a heuristic function that is based on the sum of $g(n)$, which is known cost of getting from start to the current node (n), and $f(n)$, which is the estimated of remaining cost from the current node (n) to the goal.

Therefore, in RamblersState.java, the RamblerSearcher will find all valid successors around the current node, that will be iterating 4 different directions $\{\{1,0\},\{0,1\},\{-1,0\},\{0,-1\}\}$, and we will ignore all invalid nodes that meet one of 4 conditions that checks whether this node has exceed the boundary of this map.

*https://github.com/aronjyl/COM1005

```java
public ArrayList<SearchState> getSuccessors(Search searcher){
    RamblersSearch ramSearcher = (RamblersSearch) searcher;
    TerrainMap terr_m = ramSearcher.getMap();
    int[][] tMap = terr_m.getTmap();

    ArrayList<SearchState> successors = new ArrayList<SearchState>();
    // iterate 4 around directions and collect all valid steps
    for (int i=0; i<4; ++i) {
        int x = curr_coords.getx() + DIRECTIONS[i][0];
        int y = curr_coords.gety() + DIRECTIONS[i][1];

        // ignore invalid steps
        if (x < 0 || y < 0 || x >= terr_m.getWidth() || y >= terr_m.getDepth()) continue;

        Coords nextCoords = new Coords(x, y);
        int temp_est;
        // 0 for eucledian, 1 for manhatten, 2 for height and 3 for all
        switch (eval_flag) {
            case 0:
                temp_est = terr_m.eucledianEst( curr_coords, ramSearcher.getGoal() );
                break;
            case 1:
                temp_est = terr_m.manhattenEst( curr_coords, ramSearcher.getGoal() );
                break;
            case 2:
                temp_est = terr_m.heightEst( curr_coords, ramSearcher.getGoal() );
                break;
            default:
                temp_est = terr_m.eucledianEst( curr_coords, ramSearcher.getGoal() ) +
                           terr_m.manhattenEst( curr_coords, ramSearcher.getGoal() ) +
                           terr_m.heightEst( curr_coords, ramSearcher.getGoal() );
                break;
        }
        RamblersState nextSuccessor = new RamblersState(nextCoords,
                                                        tMap[nextCoords.getx()][nextCoords.gety()],
                                                        temp_est, eval_flag);
        successors.add(nextSuccessor);
    }
    return successors;
}
```

Figure 1: getSuccessors() for RamblersState

Moreover, we have evaluated 4 different distances for the A* search in TerrainMap.java as displayed in Fig 2, so RamblersState will use an instance of TerrainMap to evaluate the cost and decide the next successors, as displayed in the Fig 1.

```java
public int eucledianEst(Coords start, Coords end){
  double deltaY = start.gety() - end.gety();
  double deltaX = start.getx() - end.getx();
  double diff_res = Math.sqrt(deltaX * deltaX + deltaY * deltaY);

  return (int) diff_res;
}

public int manhattenEst(Coords start, Coords end){
  int est = Math.abs(start.getx() - end.getx()) + Math.abs(start.gety() - end.gety());
  return est;
}

public int heightEst(Coords start, Coords end){
  int est = Math.abs(start.gety() - end.gety());
  return est;
}
```

Figure 2: Evaluation functions in TerrainMap

- Manhattan distance, $d_m(x, y) = \sum_{i=1}^{n} |x_i - y_i|$
- Euclidean distance, $d_e(x, y) = \sum_{i=1}^{n} \sqrt{x_i^2 - y_i^2}$
- Height distance, $d_h(y_i, y_j) = |y_i - y_j|$
- Combinations of all above three, $D = d_m + d_e + d_h$

2 Assessing efficiency

We evaluated the A* search in 100 random locations in two different maps by 4 different costs (Euclidean, Manhattan, Height and the combined all), and the result is displayed in Fig 3, the y-axis is the efficiency that means A* search is more efficient than B&B in general and A* search is affected by the choice of evaluation distance, and the combined all costs together has the most advantage in the efficiency performance.

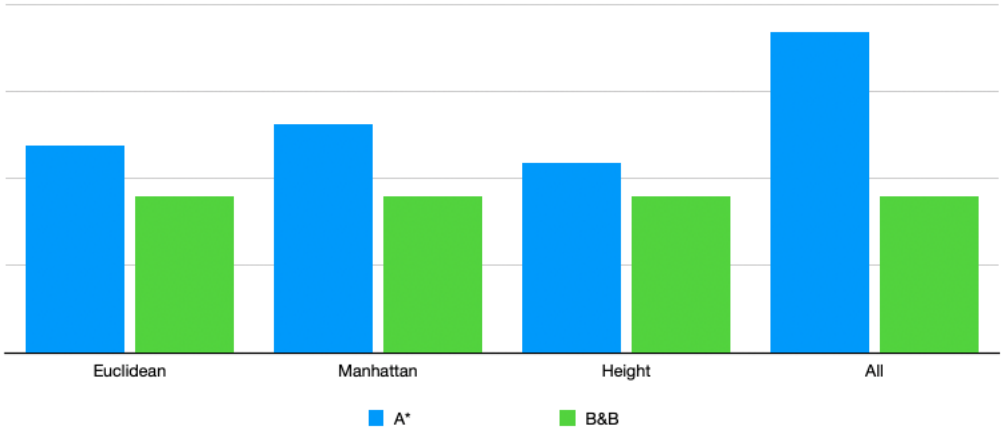

Figure 3: The average performance evaluation on A* at 4 different costs

2.1 Assessing the efficiency of my branch-and-bound and A* search algorithm

We generated 1000 random start and end locations in the target map, and evaluate BB search algorithm. The average efficiency result for 1000 times is 0.13 and the standard deviation is 0.12.

	Branch & Bound		A* Star	
	avg. efficiency	std	avg. efficiency	std
tmc.pgm	0.117	0.108	0.124	0.084
diablo.pgm	0.125	0.106	0.162	0.077

Table 1: Evaluation B&B and A* star in 1000 random locations

3 Conclusions

Based on our evaluation on two maps (tmc.pgm and diablo.pgm), we can see A* star search is better than BB in terms of efficiency performance, and its performance is affected by the choice of evaluation function.