Nathan Mohapatra

Ian Davidson

ECS 170 001

5 February 2021

Programming Assignment 1: Report

Create an admissible heuristic, document the exact form of the heuristic and prove/show it is admissible.

AStarDiv:

```
private double getHeuristic(final TerrainMap map, final Point pt) {

    // Distance between current point and end point
    final Point endPoint = map.getEndPoint();
    final int dx1 = Math.abs(pt.x - endPoint.x);
    final int dy1 = Math.abs(pt.y - endPoint.y);
    // Distance between start point and end point
    final int dx2 = Math.abs(map.getStartPoint().x - endPoint.x);
    final int dy2 = Math.abs(map.getStartPoint().y - endPoint.y);

    // Cross product gives preference to paths along straight line between start point and end point
    int crossProduct = Math.abs((dx1 * dy2) - (dx2 * dy1));
    final double MULTIPLIER = 0.1; // constant to be adjusted

    // Calculate heuristic, vary between type of distance
    final double cost = map.getCost(pt, endPoint);
    double heuristic = cost * (dx1 + dy1) + (cost - (2 * cost)) * Math.min(dx1, dy1); // Cost estimate w/
Chebyshev distance

    return heuristic + (crossProduct * MULTIPLIER);
  }
```

To be completely honest, my AStarDiv heuristic is not always admissible (it is admissible for seeds 1-5, but not for the much larger grid). I believe that this is because there are *edge cases* in which my AStarDiv heuristic overestimates. Despite this, I will explain how it helps find an optimal path, efficiently and intelligently, for seeds 1-5.

The cost between the current node and the goal node is estimated with the Chebyshev distance (8-directional movement, with cardinal and non-cardinal movement potentially sharing the same cost):

```
final double cost = map.getCost(pt, endPoint);

double heuristic = cost * (dx1 + dy1) + (cost - (2 * cost)) * Math.min(dx1, dy1);
```

With a larger grid, there will be many paths with the same path cost; therefore, A* could potentially explore all the paths with the same $f(n) = g(n)$, instead of only one. To prevent this and "break ties" between paths, my AStarDiv heuristic computes the vector cross-product between the vector from the start node to the goal node and the vector from the current node to the goal node:

```
int crossProduct = Math.abs((dx1 * dy2) - (dx2 * dy1))
```

When these vectors do not align, preference will be given to paths along the straight line from the start node to the goal node, which is responsible for efficiency and "intelligent" behavior.

A constant factor is chosen such that it is < (minimum cost of taking one step) / (expected maximum path length).

AStarExp:

```
private double getHeuristic(final TerrainMap map, final Point pt) {

    // Distance between current point and end point
    final Point endPoint = map.getEndPoint();
    final int dx = Math.abs(pt.x - endPoint.x);
    final int dy = Math.abs(pt.y - endPoint.y);

    // Best-case path depends on difference in height, between current point and end point
    final double height1 = map.getTile(pt);
    final double height2 = map.getTile(endPoint);

    // Case 1: Current point level with end point, stay at same height
```

```
        double heuristic = Math.max(dx, dy); // Chebyshev distance

        // Case 2: Current point is below end point, take small steps up then remain at same height
        if (height1 < height2) {
            for (int i = 0; i < Math.abs(height1 - height2); i++) {
                heuristic -= 1;
                heuristic += Math.exp(1);
            }

        // Case 3: Current point is above end point, take large step down then remain at same height
        } else if (height1 > height2) {
            heuristic -= Math.abs(height1 - height2);
            heuristic += 1 / Math.exp(Math.abs(height1 - height2));
        }

        return heuristic;
    }
```

My AStarExp heuristic is admissible, because it *never overestimates* the cost to reach a goal. Furthermore, my AStarExp heuristic is consistent because, for every node *n* and every successor *n'* of *n* generated by an action *a*,

$$h(n) \leq c(n, a, n') + h(n').$$

My AStarExp heuristic obeys the *triangle inequality* by considering three different cases and estimating the cost of the *best-case* path (a guaranteed underestimate) from the current node to the goal node. These three cases are:

1. Case 1: The current node is level with the goal node

   → Remain at the same height, because the cost of remaining at the same height (for 2 steps) is less than the cost of stepping up then stepping down

   → $h(n) = D * e^0 \leq (D - 2) + e^1 + e^{-1} = c(n, a, n') + h(n')$

2. Case 2: The current node is below the goal node

   → Take small steps up then remain at the same height, because the cost of two small steps up (+1 height) is less than the cost of one larger step up (+2 height)

   → $h(n) = (D - 2) + e^1 + e^1 \leq (D - 2) + e^2 = c(n, a, n') + h(n')$

3. Case 3: The current node is above the goal node

   → Take large step down then remain at the same height, because the cost of one larger step down (-2 height) is less than the cost of two smaller steps down (-1 height)

   → $h(n) = (D - 2) + e^{-2} \leq (D - 2) + e^{-1} + e^{-1} = c(n, a, n') + h(n')$

A clear and concise description of your modified A*.

To modify my AStarDiv algorithm (MtStHelensDiv) such that it finds the optimal path in a new environment in *the least possible time*, I made two slight optimizations:

1. I implemented *Weighted A** by multiplying the heuristic by a constant factor, a weight $w > 1$ (I chose $w = 2.0$). This makes A* run faster because it magnifies the heuristic's effect.

   ```
   final double WEIGHT = 2.0;
   return (heuristic + (crossProduct * MULTIPLIER)) * WEIGHT;
   ```

2. I modified my implementation of Dijkstra's algorithm such that there are no duplicate entries added to the priority queue. When iterating through the neighbors of a node that has been popped off the priority queue, it is possible to encounter a node that has been reached previously, but at a higher cost. In my earlier implementation of Dijkstra's algorithm, a duplicate entry would be added to the priority queue, despite its cost being updated (and stored in a different data structure). With a conditional statement that checks for this, the higher-cost entry (the duplicate) is removed from the priority queue, and the runtime for Dijkstra's algorithm is decreased.

Together, these two optimizations lowered the average runtime of the algorithm from ~1200ms to ~400ms.

The cost of your shortest path, number of nodes expanded, and time to find it for seeds 1, 2, 3, 4, 5 for both the AStarDiv and AStarExp.

Note: TimeTaken values are from my laptop, not the CSIF computers...

Dijkstra's (Div):

	PathCost	**Uncovered**	**TimeTaken**
Seed 1	198.59165501141644	162340	4431
Seed 2	198.56141550095256	162364	3379
Seed 3	198.4864317411443	162247	3505
Seed 4	198.70066424826052	162263	4131
Seed 5	198.25499446810397	161946	3310

AStarDiv:

	PathCost	**Uncovered**	**TimeTaken**
Seed 1	198.59165501141644	1373	76
Seed 2	198.56141550095256	1301	39
Seed 3	198.4864317411443	1356	45
Seed 4	198.70066424826052	1144	47
Seed 5	198.25499446810397	1690	66

Dijkstra’s (Exp):

	PathCost	**Uncovered**	**TimeTaken**
Seed 1	533.4482191461119	226914	12407
Seed 2	549.5036346739352	237620	12420
Seed 3	510.97825243663607	228697	11165
Seed 4	560.6570436319696	216438	11344
Seed 5	479.5879215923168	220673	10594

AStarExp:

	PathCost	**Uncovered**	**TimeTaken**
Seed 1	533.4482191461119	72625	2849
Seed 2	549.5036346739352	82171	3797
Seed 3	510.97825243663607	75432	3323
Seed 4	560.6570436319696	68869	2386
Seed 5	479.5879215923168	69895	2523