Ammanuel Mihrete (akm142)

Luke Schroeder (lhs52)

Project 1 Report

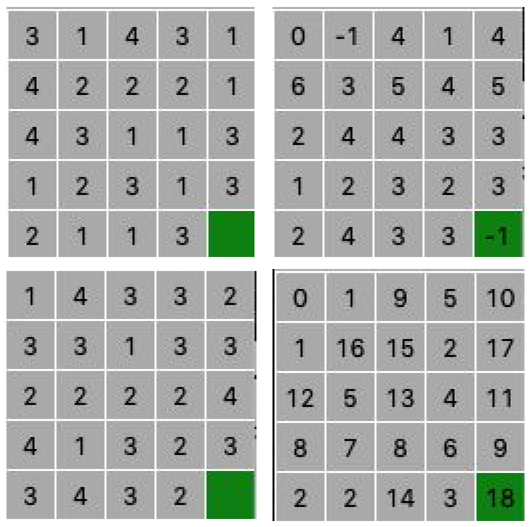
Task 1: Problem Formulation

- Difficult Puzzle Generation
    - Environment
        - Space of all nxn puzzles stored as indexed dictionaries in Python
    - State Space
        - The set of all specific puzzle configurations
        - Goal state is a maximal puzzle evaluation bounded by n * n
    - Actions
        - Hill Climbing: Randomly change single node in puzzle, keep result if evaluation improves
        - Genetic: Mutation, Crossover, Selection as explained below
    - Perception/Observations
        - Hill Climbing: Single randomly generated puzzle, iteratively manipulated
        - Genetic: Population of randomly generated puzzles
    - Transition Function
        - Hill climbing: Update nodes of puzzle upon evaluation improvement
        - Genetic: Applying mutation, crossover, selection to population and in the last iteration select the most fit offspring puzzle
    - Evaluation Metric
        - If goal node able to be reached: E(puzzle) = minimum steps to reach goal node
        - Otherwise: E(puzzle) = **-** (number of unreachable nodes)

- Puzzle Evaluation
    - Environment
        - Single nxn puzzle stored as indexed dictionary in Python
    - State Space
        - Set of all nodes in puzzle, starting at top left corner and ending in bottom right
        - Goal state is bottom right corner
    - Actions
        - Move k spaces up, right, down, left within puzzle bounds depending on the movenumber associated with each node
    - Perception/Observations
        - Bounds of puzzle and move numbers of neighboring nodes
    - Transition Function
        - Move from current node under consideration to one of its neighbors
    - Evaluation Metric
        - If reached goal node then success otherwise store steps taken from initial node

Task 3: Puzzle Evaluation

5x5



3	1	4	3	1
4	2	2	2	1
4	3	1	1	3
1	2	3	1	3
2	1	1	3	🟩

0	-1	4	1	4
6	3	5	4	5
2	4	4	3	3
1	2	3	2	3
2	4	3	3	-1

1	4	3	3	2
3	3	1	3	3
2	2	2	2	4
4	1	3	2	3
3	4	3	2	🟩

0	1	9	5	10
1	16	15	2	17
12	5	13	4	11
8	7	8	6	9
2	2	14	3	18

7x7

3	6	1	6	6	1	6
2	1	4	2	1	4	4
6	4	3	1	4	3	3
2	5	3	2	3	5	2
1	1	2	4	2	3	3
2	4	1	4	4	1	4
3	6	2	2	5	4	🟩

0	4	3	1	6	-1	-1
2	5	3	5	5	6	4
7	6	5	9	6	4	8
1	7	2	6	9	3	8
5	6	4	3	5	6	6
2	4	3	4	6	5	5
4	3	3	2	4	3	-1

4	6	2	2	4	1	5
2	5	4	1	3	4	6
1	5	2	3	4	4	1
6	1	3	2	1	1	6
4	4	1	3	3	5	2
4	1	4	4	3	2	5
1	6	3	4	4	2	🟩

0	4	32	23	1	24	10
6	4	7	22	3	25	5
14	15	33	20	13	17	16
7	30	31	29	28	27	8
1	3	34	35	2	4	36
11	10	8	21	12	26	9
15	5	32	19	14	18	37

9x9

1	3	2	5	1	3	7	2	6
8	2	6	3	2	2	6	3	2
5	1	3	3	5	1	1	4	3
3	2	6	2	2	2	2	7	3
2	5	6	3	2	2	1	2	3
3	4	4	5	2	5	1	6	2
1	7	2	6	6	4	1	4	5
2	7	4	5	4	5	5	6	5
5	6	2	8	8	7	1	5	🟩

0	1	4	3	2	3	-1	-1	4
1	3	4	4	3	5	3	6	2
6	-1	5	7	7	6	5	6	7
6	2	5	3	4	4	5	5	3
6	-1	7	5	-1	7	6	7	8
5	3	6	4	5	4	6	7	5
7	8	-1	5	-1	8	7	7	4
7	5	5	6	6	-1	4	-1	6
6	-1	-1	-1	-1	7	-1	-1	-1

4	4	3	8	8	3	7	7	7
6	6	3	6	6	6	3	3	8
5	7	2	4	1	3	1	7	1
8	4	5	5	3	2	5	4	4
6	6	6	3	1	3	5	5	5
6	4	4	5	1	4	4	7	5
6	7	3	4	6	4	6	5	8
7	4	6	7	3	3	5	4	6
8	6	1	4	8	5	1	3	🟩

0	41	43	9	1	42	63	-1	40
21	32	16	8	34	17	15	33	20
50	47	52	54	53	51	65	49	48
-1	28	44	25	27	29	66	24	26
1	3	5	7	36	19	2	4	6
13	31	15	38	37	30	14	12	39
61	59	57	55	28	58	62	56	60
22	29	19	8	35	18	64	23	19
69	46	45	10	2	68	67	11	70

11x11

3	1	9	2	9	7	3	7	2	5	10
3	6	5	3	7	5	2	4	4	3	6
5	2	1	6	4	5	1	3	3	6	4
8	6	6	3	4	6	3	6	1	2	1
2	7	7	2	2	3	6	2	1	5	9
6	6	3	6	1	3	5	5	3	2	7
9	9	6	7	4	3	5	7	4	9	2
10	2	3	4	5	7	3	5	3	5	6
4	6	2	4	5	8	1	6	3	8	6
7	5	4	3	8	8	9	4	9	7	4
4	5	3	5	9	10	9	10	6	10	🟩

0	2	3	1	6	2	6	6	-1	5	-1
6	3	7	5	6	6	5	4	6	4	7
5	6	6	2	6	4	6	5	3	3	5
1	4	7	6	8	8	6	3	2	3	-1
7	5	6	4	5	5	6	4	3	4	7
-1	5	5	-1	-1	5	-1	5	4	4	-1
6	8	7	5	6	7	7	5	7	7	6
6	4	7	5	6	3	-1	6	-1	5	7
7	5	6	3	7	6	-1	4	5	4	7
6	5	4	5	7	8	5	4	-1	5	-1
-1	-1	7	-1	7	8	7	6	8	-1	-1

6	5	7	10	10	4	9	10	1	9	7
9	5	4	4	7	4	7	6	7	4	2
3	1	6	3	6	4	7	5	8	9	6
8	1	5	2	3	6	1	5	1	6	4
7	8	8	7	5	6	6	5	7	5	10
10	7	5	3	3	3	6	7	6	2	7
6	5	5	2	2	6	5	1	5	6	5
1	7	5	3	4	6	2	5	5	9	2
6	4	5	8	7	3	6	5	4	8	8
5	8	6	8	3	2	9	9	9	7	3
2	10	6	9	5	4	4	6	8	2	🟩

0	29	33	10	81	52	1	26	38	34	87
-1	4	48	12	42	70	3	13	39	71	91
6	61	16	7	84	60	5	15	17	36	85
63	62	18	66	80	67	93	19	64	65	92
23	42	21	51	44	53	94	24	41	43	22
7	30	32	8	83	59	9	33	31	72	8
1	3	47	50	46	51	2	48	49	-1	90
74	75	15	77	79	69	78	14	76	73	88
5	57	17	9	41	58	4	18	40	56	86
-1	46	36	69	45	68	2	25	37	35	89
19	28	20	11	82	54	95	27	18	55	96

Task 7: Genetic Algorithm

The population-based local search approach that was implemented within this task consisted of selection with elitism, crossover and mutation. The parameters necessary to execute the algorithm are the size of the population, a selection rate, and the number of iterations the algorithm runs(up to the user to decide on this number). The algorithm begins with generating a population of size n and uses this information to create n randomly generated puzzles of a specified dimension. Once these puzzles are generated, the algorithm proceeds as following:

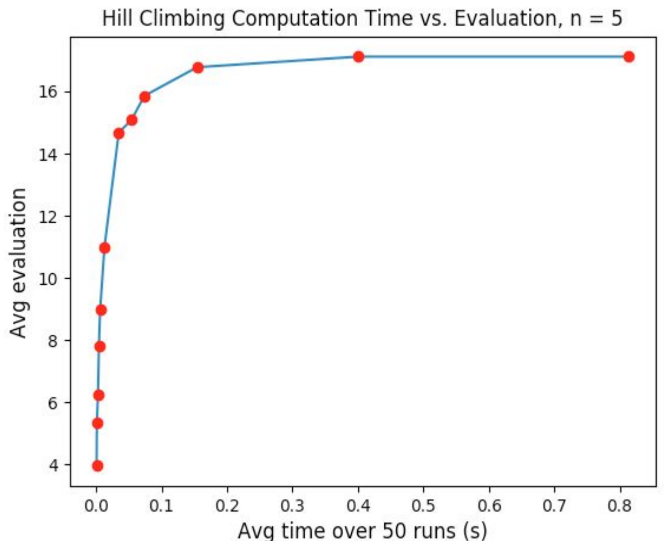
1. Selection
   The selection algorithm chooses optimal candidates from the population by first sorting the population by increasing evaluation of its members. It then reduces the population size by a rate specified by the user to save the top puzzles.

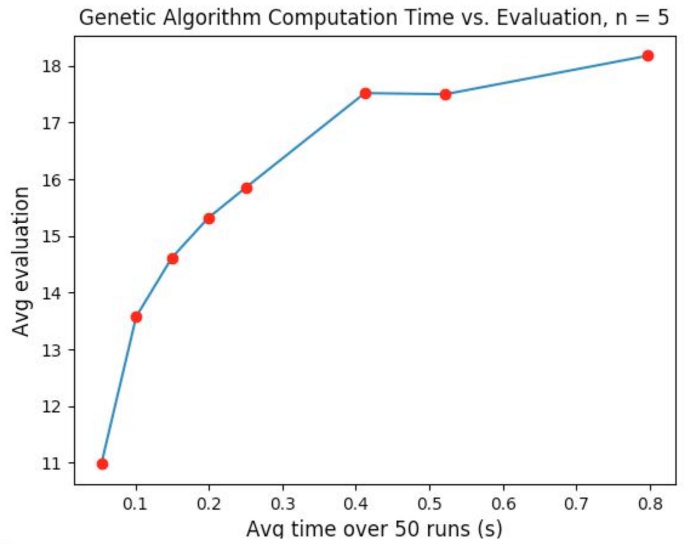
2. Mutation
   The mutation function takes in the population and the number of mutations that will occur. The mutation is implemented on the selected population by swapping a random node with a suitable value in such a way that is similar to hill climbing. The puzzle is then evaluated to see if the mutation improved its evaluation. If the mutation was not beneficial to the state of the puzzle, the move is reversed to its original position.

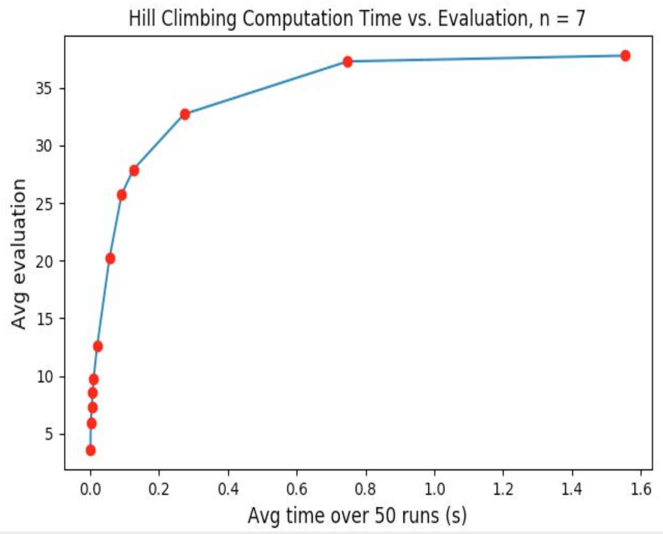
3. Crossover
   In the crossover stage, a population of n elements is mated with itself to generate n^2 babies. Each member of the population is mated with each of the other members to generate all of the babies. The crucial step is the mate function which takes two puzzles as an input and generates their baby. In the mate function, the node at specific coordinates of the graph will maintain the move number of one of the parents. This move number is chosen based on a comparison of the distances from the start node of the two parents to that point. The movenumber **m** at node **(x, y)** of the baby will be that of the parent with movenumber **m** with the greater distance from the initial node to the coordinates **(x, y).** We chose this crossover function in the hopes to preserve a long path from one of the parents. Say it takes 10 steps to reach a node in parent puzzle A and 3 steps to reach a node in parent puzzle B, then we want to preserve the movenumber in puzzle A to try and keep the path to that node alive in the offspring puzzle.

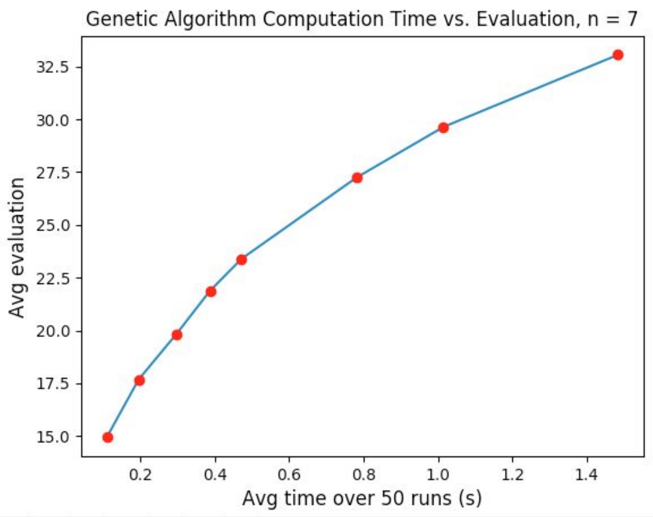
4. Iteration / Termination
   Iteratively, the population returns to the selection phase. After a finite number of iterations the evolution is complete and the fittest baby will be chosen.

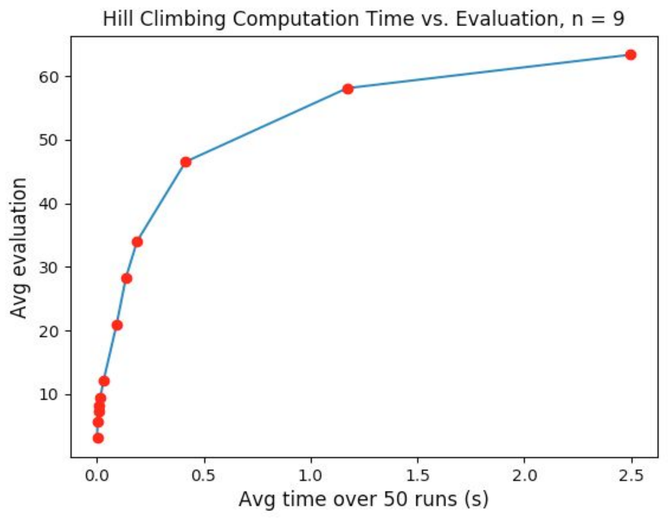
Task 4: Hill Climbing & Task 7: Genetic Algorithm

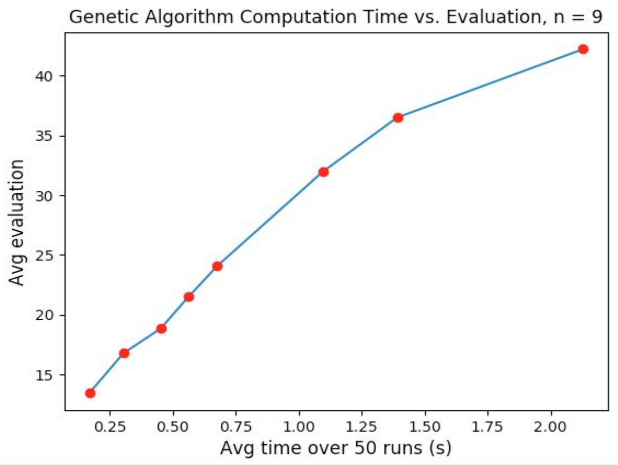


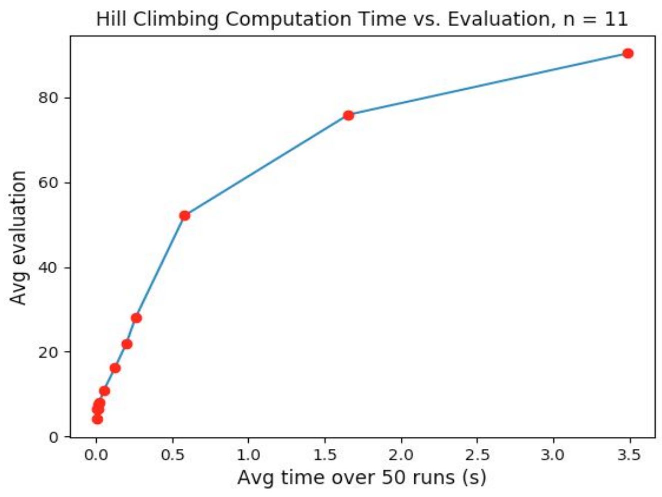


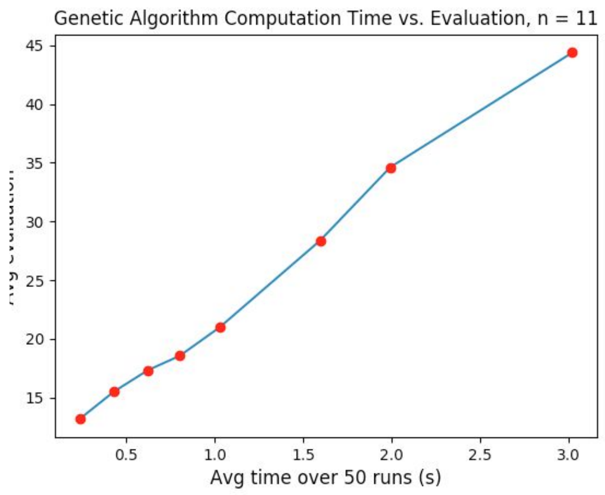










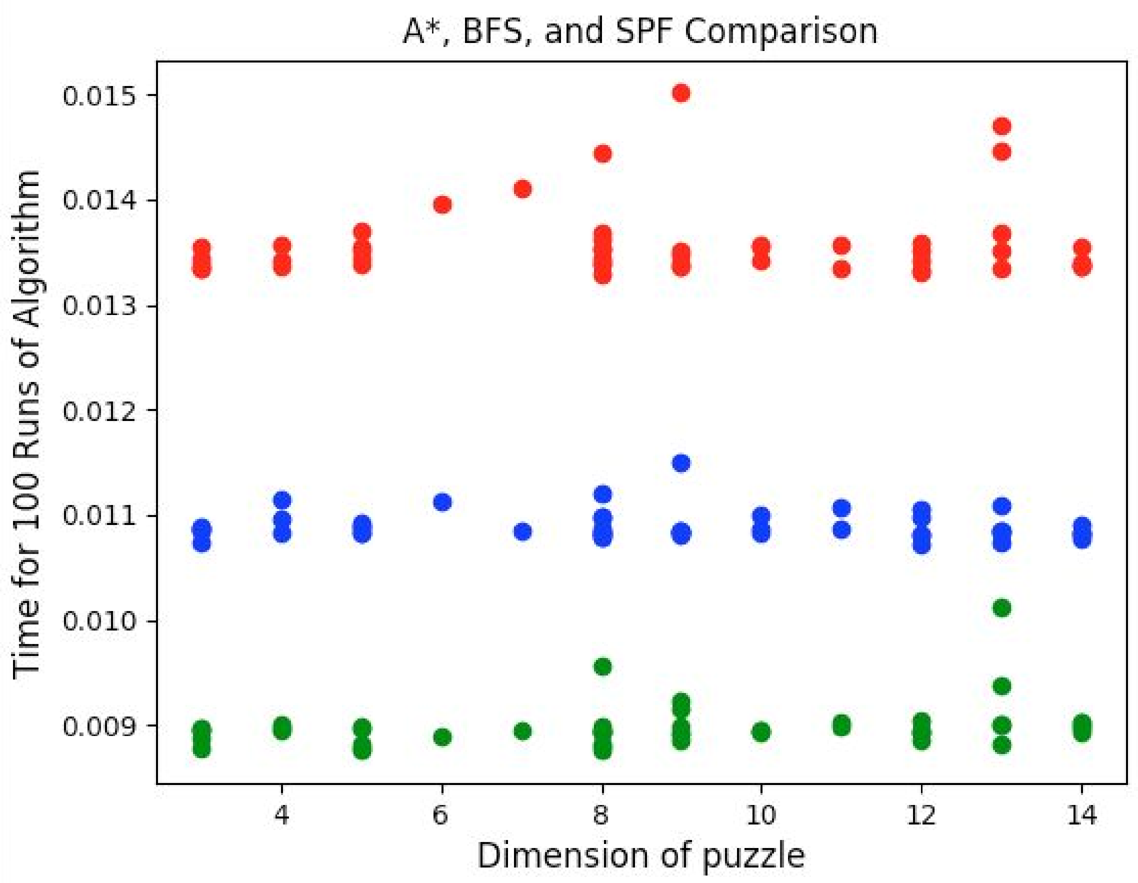


Analysis:

- We chose to implement computational time in seconds to reflect an increase in iterations for each algorithm. This is because the hill climbing has a clear definition for “iteration”, a single random mutation of a move node. However, the genetic algorithm carries out three different operations: mutation, crossover, and selection. In selection the list of population members is sorted and filtered by their corresponding evaluations. In crossover there is an n x n traversal of each parent puzzle in the generation of a baby puzzle. Mutation is similar to hill climbing. Therefore, the fairest and most accurate comparison of evaluation improvement over number of iterations is on actual computation time it takes to execute each algorithm as calculated by the python timeit library.
- However, in choosing this approach the graphs are somewhat distorted. Since we are timing a minimum of one iteration of each algorithm, the initial average point evaluation is higher for the genetic algorithm. This is because the genetic algorithm starts at a higher evaluation after its first iteration since for each run it executes selection, crossover, and mutation whereas we can choose arbitrarily small execution time for the hill climbing puzzle.
- In terms of the benefits of each algorithm the genetic algorithm performed better on puzzles with a smaller n value while the hill climbing algorithm succeeded on those with higher dimension. This is due to the fact that in higher dimensions, there is a larger benefit to introducing randomness since there are more nodes that do not fall in the current longest path. Therefore there is a higher likelihood that randomly changing an unimportant node will improve the performance of the algorithm.

- The genetic algorithm performs worse at the higher dimensions since our crossover algorithm evidently was not optimal. It seems as though the hill climbing graph has a stricter horizontal asymptote and the genetic algorithm continues to increase with the computation time. In fact, some of the most complex puzzles we were able to generate came from giving the genetic algorithm significantly more time than the hill climbing. Once the hill climbing reaches a somewhat optimal path it levels off since changing random nodes will either disrupt the optimal path or change a node that falls off the path. Whereas for the genetic algorithm, since diverse populations are mated and more randomness is consistently being introduced, this effect is less prevalent.

Task 8: Evaluation



O A*   O Shortest Path First   O Breadth First Search

Analysis:

- If one uses an admissible heuristic function with A*, then it will always find the most optimal path without visiting every node or revisiting nodes. A* is guaranteed to find optimal paths and never overestimate the cost of reaching the goal due to the heuristic function. Not using a heuristic with BFS means that the algorithm will be uninformed, therefore there cannot be an estimate in distance between each node and the target.
- In our case the computation required to calculate the heuristic repeatedly for every enqueue of a node exceeds the benefit that the heuristic provides algorithmically. Our heuristic is very simple. Any node that can potentially reach the goal node with a legal move number has a heuristic value of 1. These nodes consist of only the union of the nth column and nth row of the puzzle (the bottom row and rightmost column). Then any node not falling in these categories, but whose neighbor has a heuristic of 1 receives a heuristic of 2. Any node that does not fall in the category 1 or 2 receives a heuristic of 3.
- The nodes of heuristic 3 specifically are nodes that do not fall on the same row or column as the goal node, and do not neighbor any nodes that fall on them either. The neighbors to the nodes that fall on the same column or row as the goal node have heuristic 2. Those on the same row/column have 1.
- The purpose of the heuristic is to prioritize the selection of nodes that are closer to the goal node. Those that can potentially be neighbors to the goal node are selected with first priority and those that neighbor the first priority nodes second, and third for the remaining. Thus it is more likely to choose nodes that complete the path of the puzzle. However, the computation of these priorities takes longer than the benefit selecting nodes in priority order provides and therefore A* is the slowest search.
- Breadth first search in this case is essentially the same as shortest path first. In general they differ in selection of the next node to check. When the distances differ shortest path first always selects the smallest path from the initial node whereas breadth first search does an in order traversal of consecutive neighbors at each level. SPF prioritizes based on distance, however here all the distances between neighbors are 1. Thus, the only difference is that SPF uses a priority queue over a regular queue. This more complicated data structure requires more computation and hence we have our evaluation of the three algorithms in general.
- The times remain relatively constant over the dimension of the puzzle since the difference between the size of the puzzle in CPU processing required is very small relative to the capabilities of the CPU.