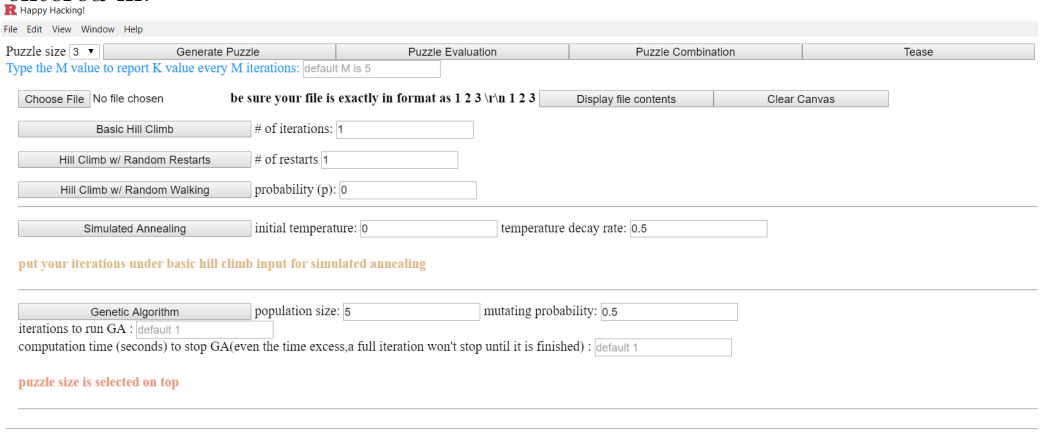
Assignment 1 Local Search Report
By Brandon Young and Ruicheng Wu

Task 1. Puzzle Representation

Here are a couple of images of the GUI we used for this assignment. This first image shows all of the options available as well as where the parameters are entered in:



1	2	2
1	1	1
2	1	0

The image below was obtained after generating a puzzle and running hill climb (number of iterations = 50). The (x:y) format shows the k values at various iterations where x is the iteration number and y is the k value found at that iteration. The GUI also shows the time it took to run basic hill climb, the best K, the puzzle found (matrix on the left) and the evaluated version (matrix on the right).

Happy Hacking!

File Edit View Window Help

put your iterations under basic hill climb input for simulated annealing

Genetic Algorithm population size: 5 mutating probability: 0.5

iterations to run GA : default 1

computation time (seconds) to stop GA(even the time excess,a full iteration won't stop until it is finished) : default 1

puzzle size is selected on top

, 5:4 , 10:4 , 15:4 , 20:4 , 25:5 , 30:5 , 35:5 , 40:5 , 45:5 , 49:5 , 50:5

final K is 5

best K : time is 0.042 seconds to reach k :5 and by the way the max is the same as final result k

The tree data structure is :["(0,0)","|","(2,0)","(0,2)","|","(2,3)","(1,2)","(0,3)","(0,1)","|","(3,3)","(2,4)","(1,3)","(2,2)","(1,1)","(0,4)","(4,1)","|","(4,3)","(3,4)","(3,2)","(1,4)","(1,0)","(4,2)","(3,1)","(4,0)","|","(G,G)","(3,0)","(2,1)"]

2	1	3	2	2
4	2	3	1	1
1	1	2	2	1
1	3	1	1	2
2	2	1	1	0

0	4	1	5	4
2	3	5	4	3
1	2	3	4	4
2	3	2	3	4
3	4	3	4	5

Task 2. Puzzle Evaluation

The puzzle is on the left, while the BFS output is on the right. The following shows 2 puzzles for each possible size, one that is solvable and one that is unsolvable

1. 5x5 (Solvable):

1	1	4	1	1
1	1	2	1	1
4	1	1	2	1
1	3	1	1	1
2	1	1	1	0

0	1	2	5	5
1	2	3	5	4
2	3	4	4	3
X	4	4	5	4
5	4	3	4	5

2. 5x5 (Unsolvable):

3	4	2	1	1
3	2	1	1	1
2	3	2	1	1
4	1	2	1	2
3	2	1	2	0

0	X	2	1	2
6	4	3	2	3
4	6	3	3	4
1	5	3	4	2
5	5	4	5	X

3. 7x7 (Solvable):

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

0	1	7	5	2	X	X
1	5	6	4	3	4	2
6	6	7	5	4	5	4
5	2	6	4	5	3	3
6	X	7	5	4	5	4
5	7	X	4	3	4	5
7	6	7	5	4	5	5

4. 7x7 (Unsolvable):

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

0	3	2	7	1	6	2
X	4	7	6	5	5	4
X	3	6	5	2	5	4
2	5	4	4	5	4	3
1	2	4	3	4	5	4
2	4	3	4	3	6	3
4	3	4	5	6	6	X

5. 9x9 (Solvable):

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

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

6. 9x9 (Unsolvable):

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

0	2	5	6	3	4	1	7	4
5	4	5	5	6	6	4	6	6
2	5	5	4	5	3	X	4	7
6	3	5	4	5	5	6	6	6
7	5	5	4	4	4	3	5	5
6	4	5	4	3	3	2	3	7
1	4	4	3	2	3	3	4	X
3	4	5	4	3	4	4	4	5
6	5	6	5	6	5	6	4	X

7. 11x11 (Solvable):

4	9	7	7	4	1	7	10	9	9	1
3	5	5	1	6	1	7	1	2	5	9
4	5	3	1	4	1	3	2	3	7	8
1	5	6	6	7	3	4	3	1	5	5
10	1	2	5	1	6	4	1	1	1	6
10	3	5	2	2	1	2	1	5	1	7
1	5	8	7	3	5	1	1	4	1	6
3	2	1	5	3	1	4	4	1	1	7
1	2	2	7	1	3	4	1	6	1	1
1	7	6	3	5	3	6	1	7	6	6
4	7	1	5	7	4	6	1	3	5	0

0	X	5	9	1	7	7	7	2	6	7
X	8	9	8	7	6	7	10	11	X	8
7	5	8	7	6	5	6	11	4	7	X
X	6	6	7	3	6	5	6	5	6	6
1	X	7	3	2	3	6	5	4	5	2
7	6	4	6	3	7	4	5	5	6	7
6	7	7	5	7	5	7	6	7	7	6
7	5	6	6	4	X	5	5	7	8	6
6	7	7	8	8	9	7	10	8	7	7
5	4	6	4	8	X	5	9	3	7	8
6	5	5	6	4	4	9	8	6	5	3

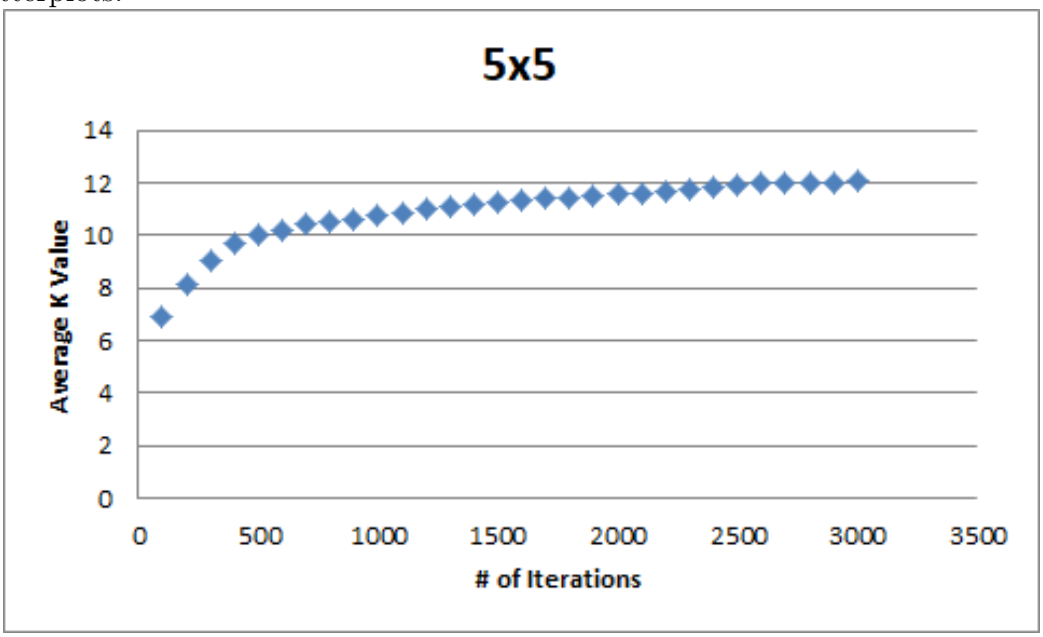
8. 11x11 (Unsolvable):

2	1	4	9	10	7	7	1	5	4	5
10	1	9	1	2	8	1	6	3	3	4
6	1	1	7	8	2	2	4	1	5	2
4	1	1	7	6	6	2	3	2	5	8
6	9	1	2	1	3	1	5	1	6	8
3	4	2	1	1	3	1	1	5	1	3
4	5	7	1	1	5	3	4	1	3	5
10	3	8	4	2	1	4	1	5	1	8
5	1	8	4	2	1	6	3	2	3	1
5	1	1	7	3	1	4	1	7	2	2
4	6	2	3	4	7	6	1	6	7	0

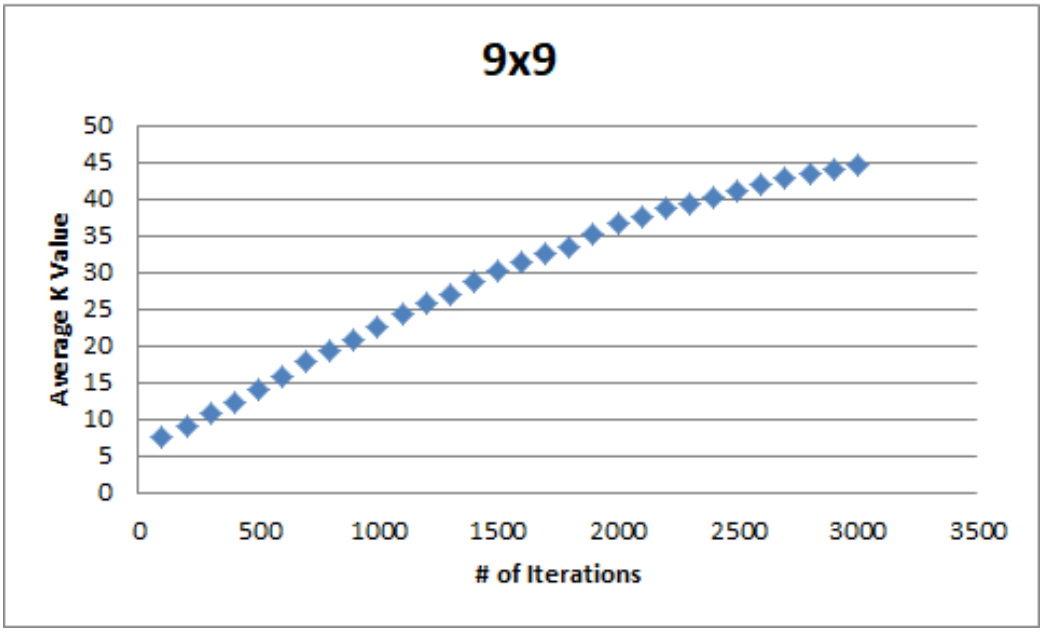
0	7	1	7	X	7	2	X	6	X	6
7	6	5	X	X	5	5	6	4	X	7
1	5	4	4	3	6	2	4	3	4	5
3	4	3	4	4	6	4	X	4	7	5
8	3	2	3	5	4	3	4	5	6	4
4	X	3	5	4	5	4	5	5	7	7
6	X	5	4	5	5	5	5	X	6	6
4	8	4	5	5	4	3	6	6	5	4
2	7	5	X	4	3	4	6	X	6	6
7	6	6	5	5	4	5	5	6	7	6
5	7	6	5	4	5	6	6	5	7	X

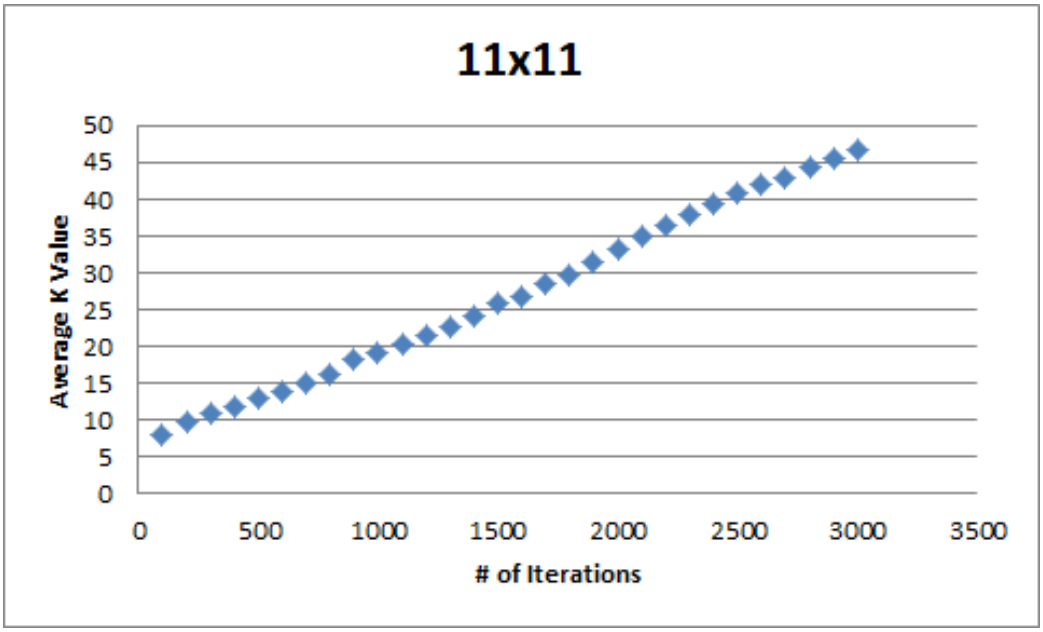
Task 3. Basic Hill Climb

To get the following plots we ran hill climb 50 times for 3000 iterations and at every 100th iteration we took the K value at that interval. Then we averaged the K values at each interval to get the data for the following scatterplots:



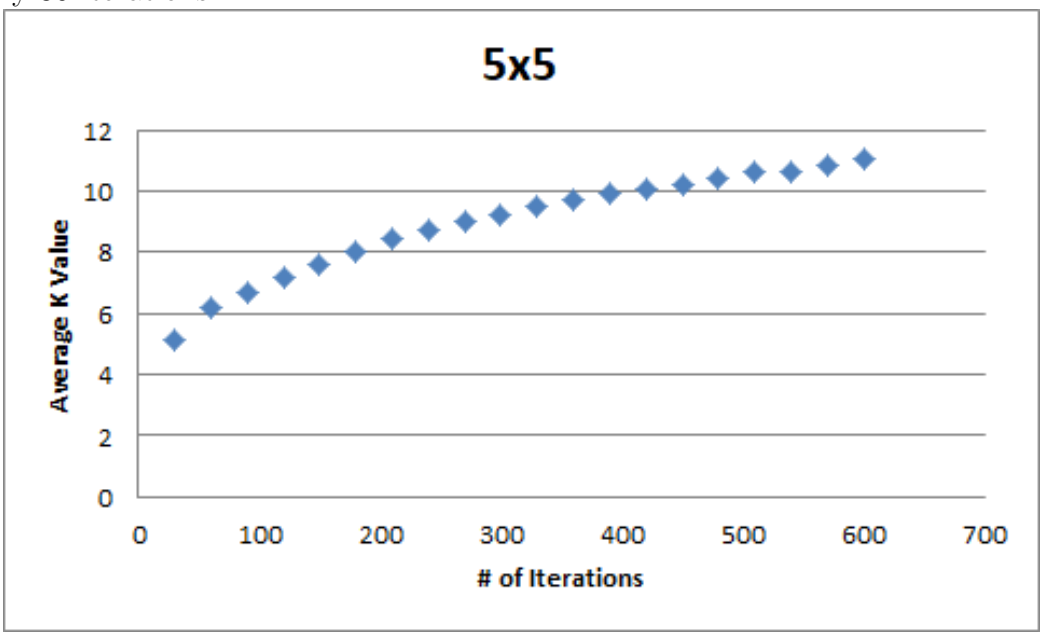

7x7
Average K Value
35
30
25
20
15
10
5
0
0
500
1000
1500
2000
2500
3000
3500
# of Iterations


9x9
Average K Value
50
45
40
35
30
25
20
15
10
5
0
0
500
1000
1500
2000
2500
3000
3500
# of Iterations

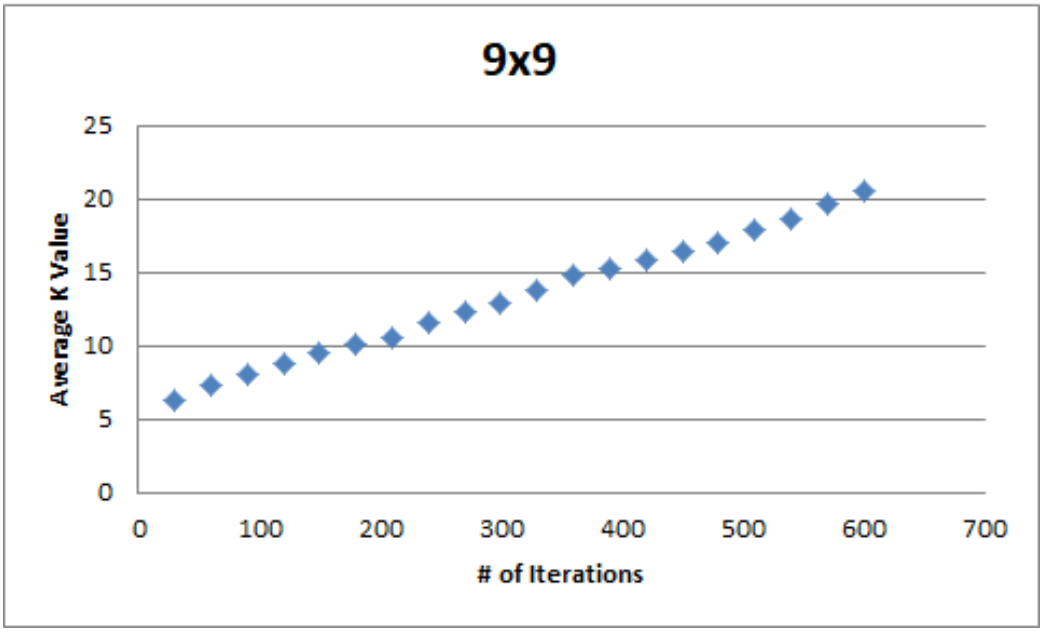

11x11
Average K Value
50
45
40
35
30
25
20
15
10
5
0
0
500
1000
1500
2000
2500
3000
3500
# of Iterations

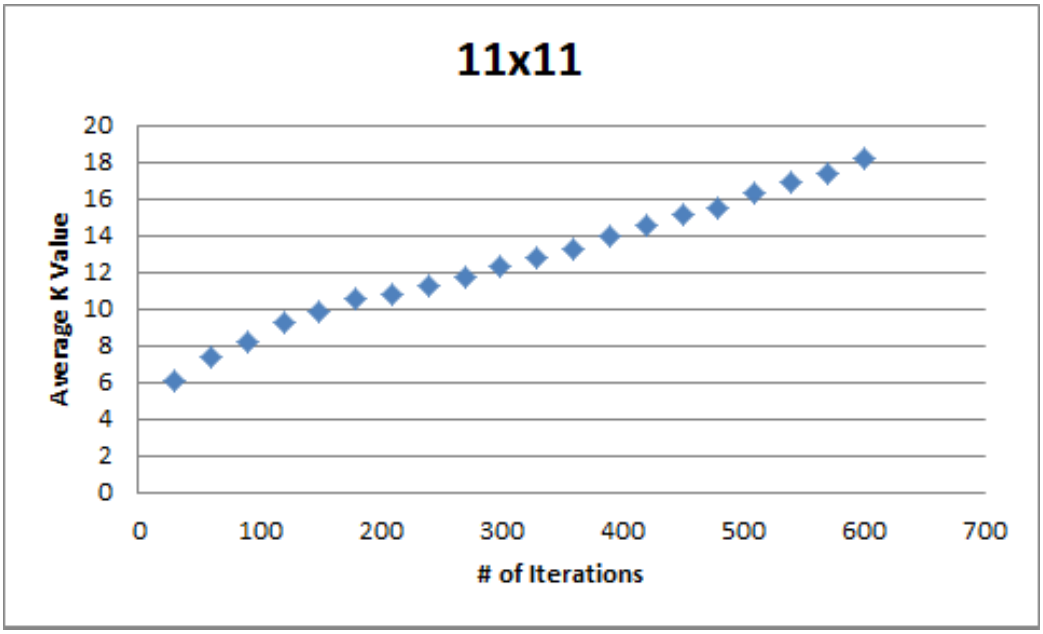
Task 4. Hill Climbing with Random Restarts

For hill climbing with random restarts, using 600 iterations and 5 restarts, the best individual hill climb was picked and its K values were recorded at every 30 iterations:




7x7
Average K Value
25
20
15
10
5
0
0
100
200
300
400
500
600
700
# of Iterations


9x9
Average K Value
25
20
15
10
5
0
0
100
200
300
400
500
600
700
# of Iterations

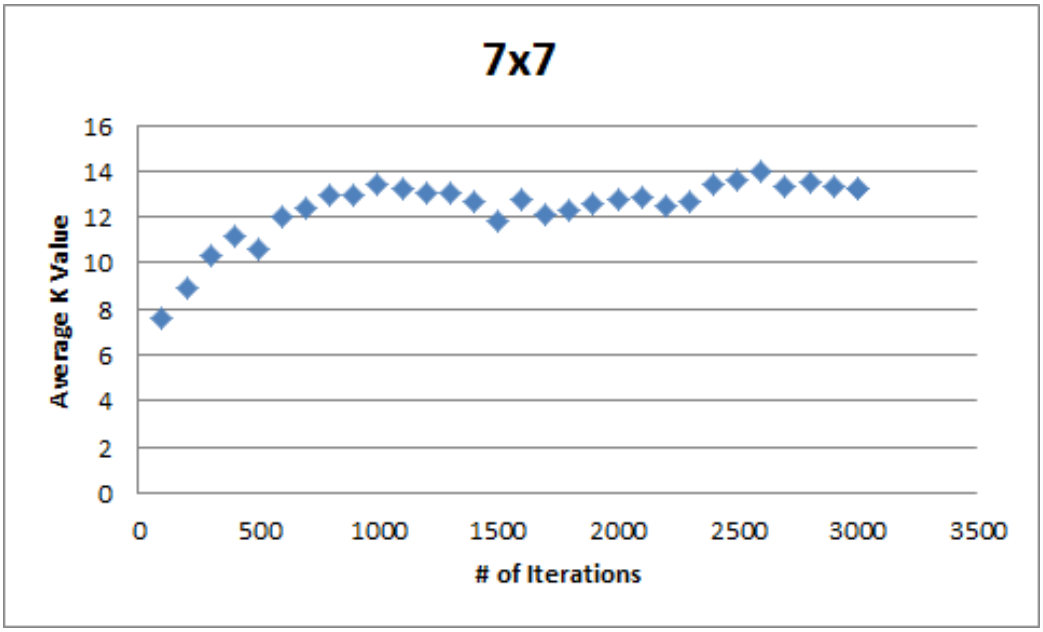


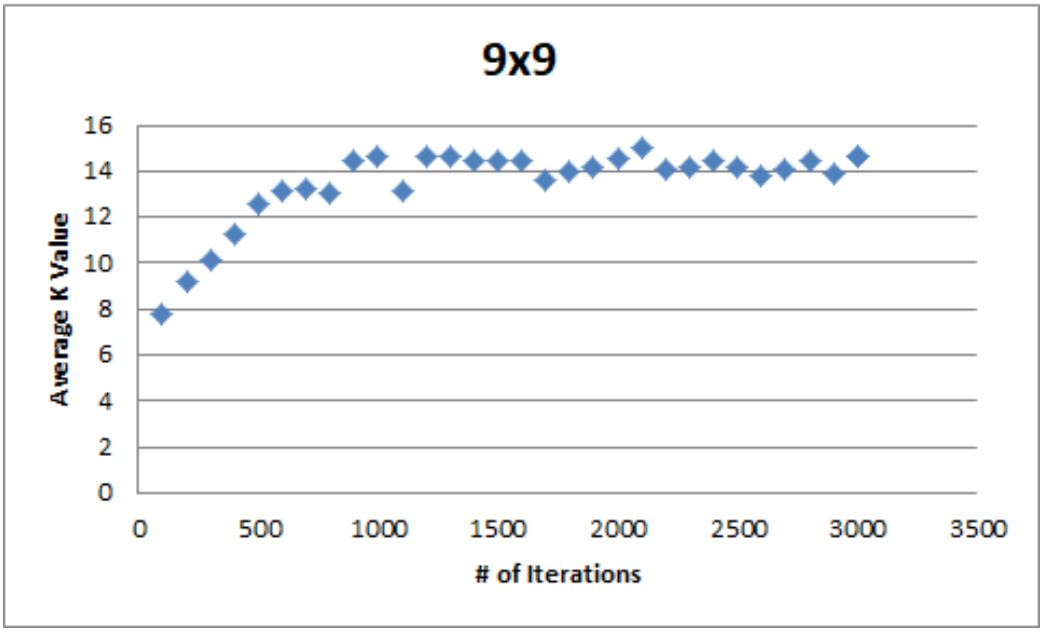
Compared to basic hill climbing, hill climbing with restarts appears to do worse. On the 5x5 plots, for example, restarts only reaches K = 11 at most, but basic hill climb reaches K = 12.

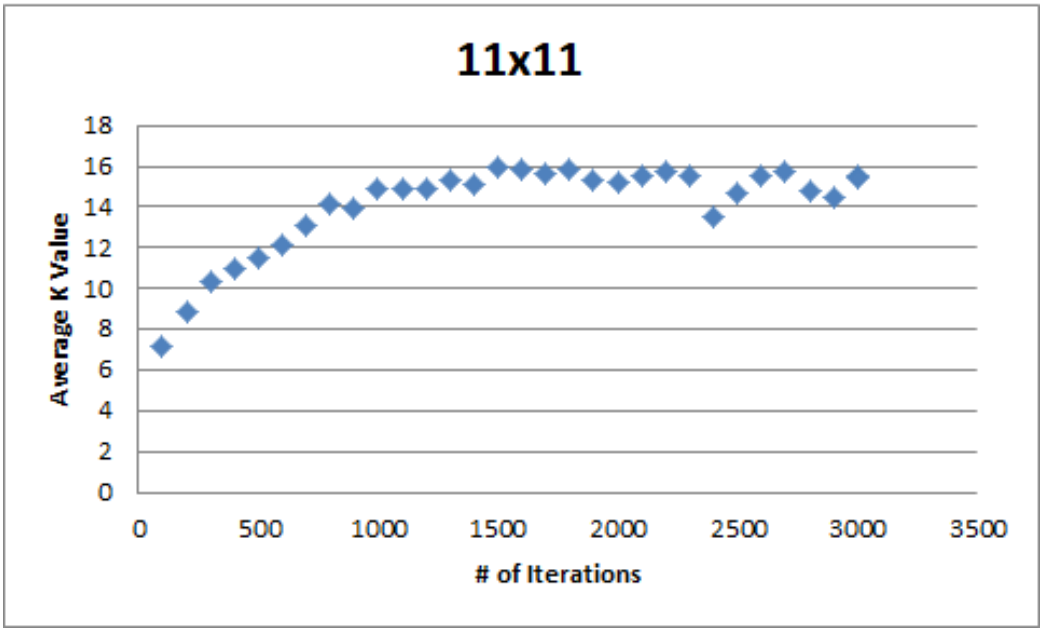
For the number of restarts, more than 2 restarts are preferred, to differentiate from basic hill climb. Yet the number of restarts should not be too high, or the hill climb process will be too short to be effective compared to basic hill climb. So 5 restarts was chosen for the plots above.

Task 5. Hill Climbing with Random Walking

For hill climbing with random walking, p = 0.01, where p is the probability of allowing downhill movement. A similar process with basic hill climbing was used to obtain the scatterplots below:


7x7
Average K Value
16
14
12
10
8
6
4
2
0
0
500
1000
1500
2000
2500
3000
3500
# of Iterations


9x9
Average K Value
16
14
12
10
8
6
4
2
0
0
500
1000
1500
2000
2500
3000
3500
# of Iterations

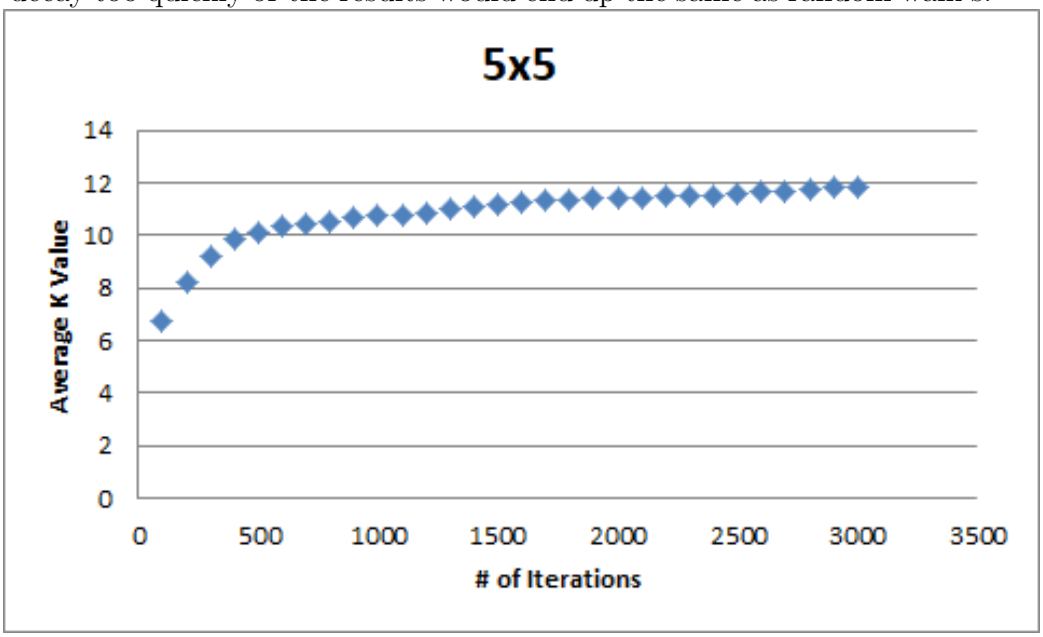


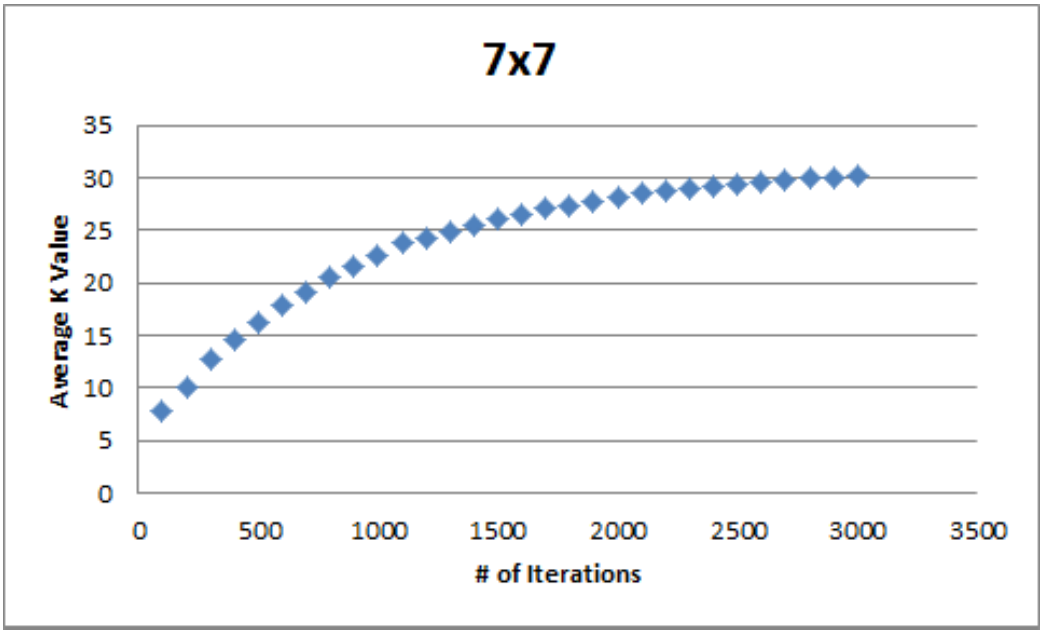
Random walk does appear to work, however, the dips caused by downhill movement can be too severe to recover from. This is why p was set to 0.01, but it looks like it could be set even lower to retain the effectiveness of basic hill climbing.

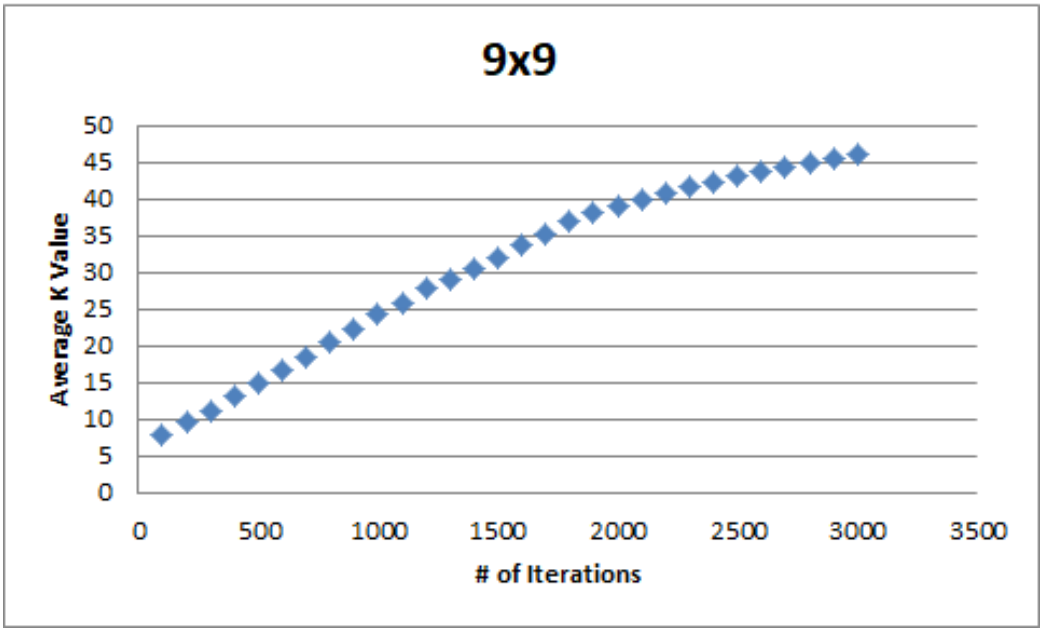
Compared to plain hill climbing and climbing with restarts, random walk performs much worse, reaching a high of about K = 8.5 for 5x5 puzzles while the other 2 methods surpassed 10. However, this may be caused by the averaging of data points, since at some points in the plots, there can be a jump in K values (such as from iterations 800 to 900 on the 9x9 plot).

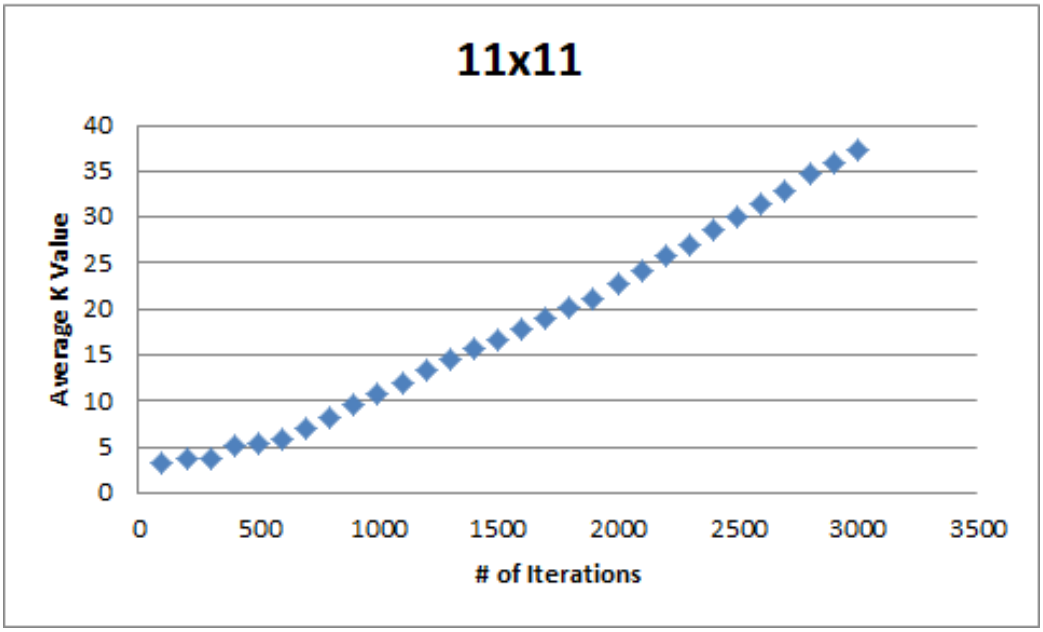
Task 6. Simulated Annealing

The parameters used for simulated annealing were: T (initial temperature)= 1000, decay rate = 0.99 with 3000 iterations and every 100th iteration being recorded. The decay rate was picked so that the temperature would not decay too quickly or the results would end up the same as random walk's.




7x7
Average K Value
35
30
25
20
15
10
5
0
0
500
1000
1500
2000
2500
3000
3500
# of Iterations


9x9
Average K Value
50
45
40
35
30
25
20
15
10
5
0
0
500
1000
1500
2000
2500
3000
3500
# of Iterations

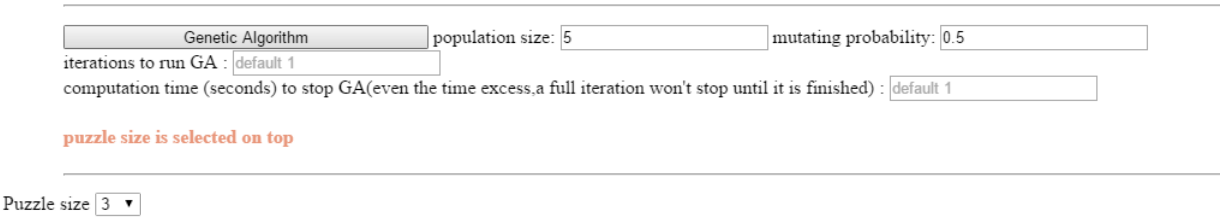


Compared to random walk, simulated annealing (SA) is much more successful, reaching a max K of 12 at the 3000th iteration compared to random walk's 8.5. Simulated annealing achieved maximum K values of about 12, 30, 46 and 37 for 5x5, 7x7, 9x9 and 11x11 puzzles respectively. In comparison, in the same total number of iterations, hill climbing with random restarts obtained K values of about 11, 22, 21 and 18. Simulated annealing performs objectively better.

However, compared to basic hill climbing, simulated annealing performs about the same, with the exception of 11x11 puzzles where simulated annealing has a lower maximum K value. This may be caused by the randomness due to the probability of downhill movement, but it shows that basic hill climbing is the most reliable out of the four methods.

Task 7. Genetic Algorithms

In order to run the genetic algorithm (GA), you need populations size and mutating probability, as seen below. Iterations is optional (by default set to 1) and computation time was used to collect statistics on runtime VS K values. By default, the population size is 5, the mutation probability is 0.5 and the number of iterations is 1. Note: there are parameters like selection probability that we generates inside of the function.



In our implementation of GA, the final optimized puzzle is the candidate with the highest fitness value, K, after all of the iterations are done. For each iteration, multiple puzzles (determined by population size) are randomly created and placed in a set. In addition, if any puzzles are invalid (have a negative K value), then another puzzle is created, in order to start with a better population. Then each individual puzzle is evaluated with the fitness function. We select a decent amount of two puzzle groups according to each individuals' fitness value (some puzzles may not be chosen at all while some may be chosen repeatedly). After selection, we randomly choose one row of puzzles in same group and do this for every group. Then process to cross-over,that is swap Part A of first puzzle with Part A of second puzzle in same group. So now each group should have $PartA - PuzzleOne - PartB - PuzzleTwo$ $PartA - PuzzleTwo - PartB - PuzzleOne$ like structure. The last step is mutation, with the corresponding mutating probability, each individual puzzle will have a chance to mutate a randomly selected row (or not mutate at all).

Steps:

1, Generation: randomly generate population size p puzzles, store in a 3D Array, where each 2D array represents a puzzle.

2. Fitness Evaluation and Selection: process each puzzle with fitness

function and store corresponding k into an array. Will calculate the total k sum and the proportion of each puzzle, for instance, a k group : $[4, 3, 2, 1]$, the puzzle with $k = 4$ will be 40 percent and puzzle with $k = 1$ will be 10 percent. These probability will be mapping along one line with range $[0, 1)$, in this case,$[0, 0.4), [0.4, 0.7), [0.7, 0.9), [0.9, 1)$. After storing and mapping k, we use $Math.random()$ to determine the selection. In $Javascript$, $Math.random()$ will generate a number randomly between 0(inclusive),1(exclusive).We check which interval it is located.Do this twice for each selection iteration(an inner loop).According to the combinatorial theory, we decide to run this selection loop until we have either $C(p, 2)$ group with no duplicate matrix or this loop has iterated more than population size p(In case the loop stuck when some puzzles with extreme high k being choosing repeatedly or just the order being swapped like $A - B$ and $B - A$).After this loop, we should have a descent amount of group with two puzzles each.

(Note : in order to keep a good start, we sort out any puzzle with negative value.)

3. Random Pick and Crossing-Over: for each group , use an advanced random number trick to pick a row to cut off and cross over, each group will only need one random number bases their total row numbers. We didn't choose to pick column because it is easier to get entire row than column when we coded(Or if you visualize the $2d$ array as column-row,that is another story).When cross-over, PartA of 1st puzzle will be joined with PartB of 2nd puzzle to make a new puzzle with exact same size as before.And the rest part joined together. Apply this operation to each group, a group of new crossover puzzle is presenting.

4. Mutating and Find Best K: Going through all operations before, in our code,randomly pick a row for each puzzle first, then we use random trick again to decide if their chosen rows are going to be mutated or not(We choose to mutate one row instead of one cell).After mutating has done, we present all these puzzles to being evaluated again and capture the puzzle with highest K and draw it and its visualizing matrix on the screen presenting with other data.

5. Grand Outer Loop and Harvesting: All operations above are just one run of GA. So we take iteration number to go through GA and harvest the

puzzle with highest value K and present it and its visualizing matrix on the screen.

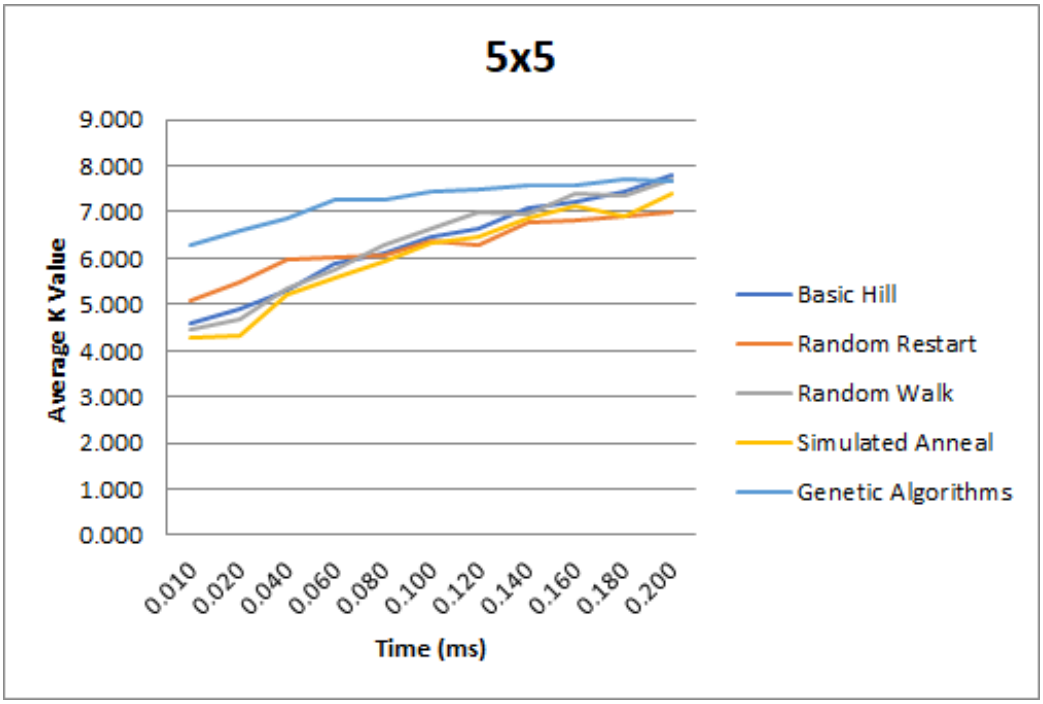
Computation Time Comparison

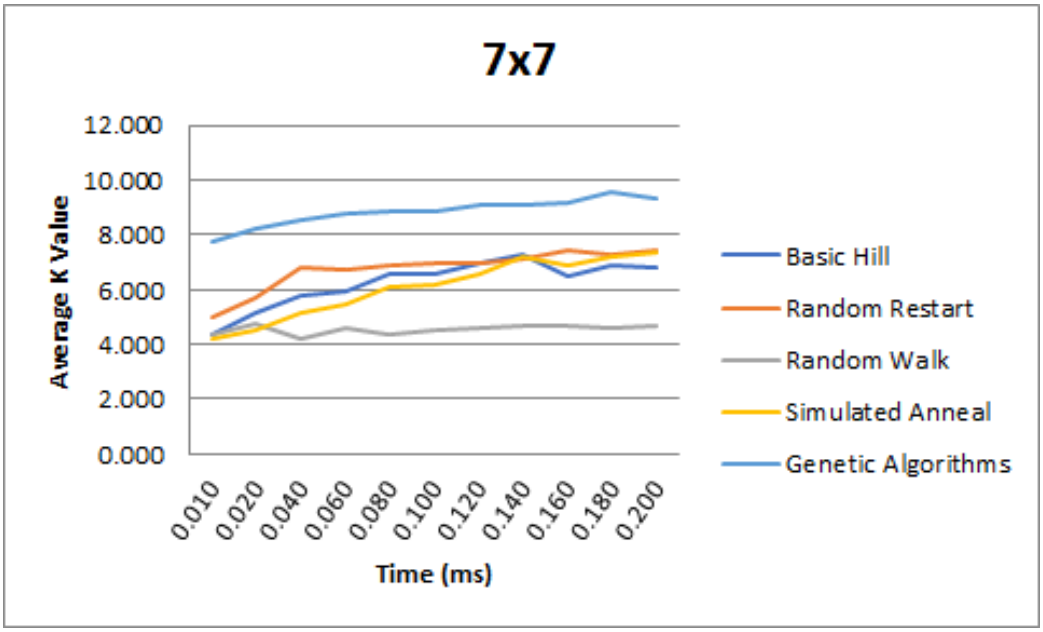
As far as the computation time goes, our genetic algorithm appears to perform the best overall compared to all other methods.

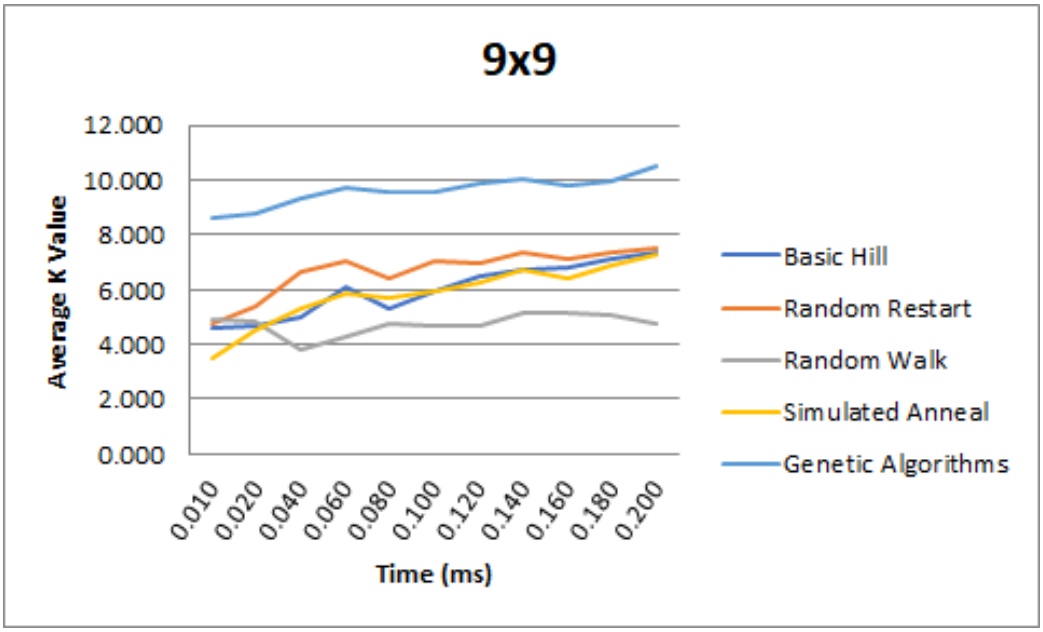
The parameters were used:

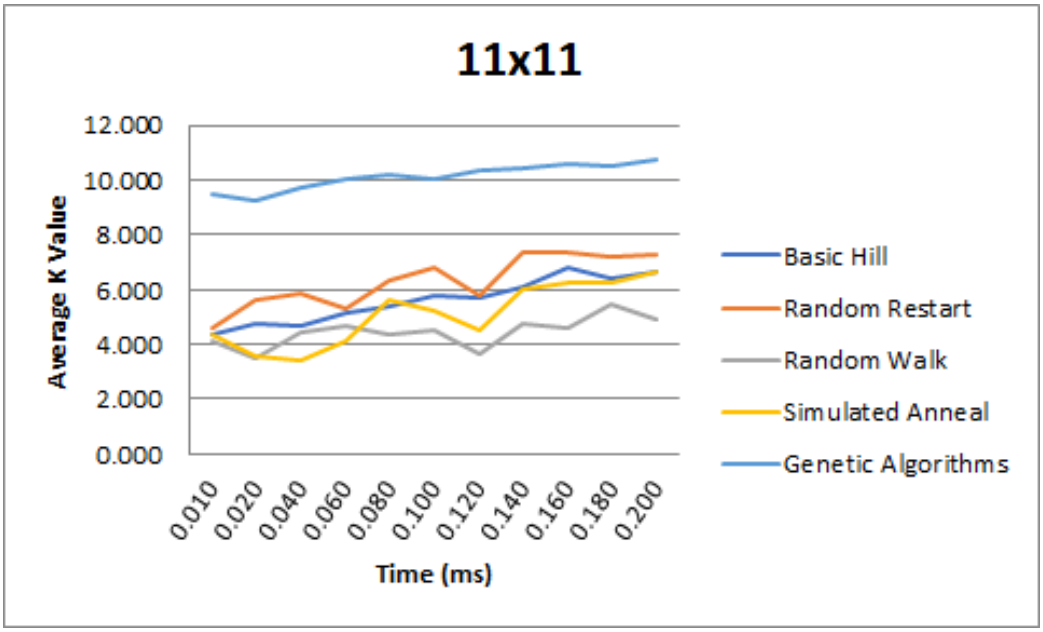
- Basic Hill Climb: 1000 iterations
- Random Restart: 40 iterations, 25 restarts
- Random Walk: 1000 iterations, p = 0.5
- Simulated Annealing: 1000 iterations, T = 1000, decay rate = 0.5
- Genetic Algorithm: population size = 50, iteration = 20, mutation probability = 0.5

The plots found are listed below with time (in ms) on the x axis and average K value on the y axis.


5x5
9.000
8.000
7.000
6.000
5.000
4.000
3.000
2.000
1.000
0.000
Average K Value
0.010
0.020
0.040
0.060
0.080
0.100
0.120
0.140
0.160
0.180
0.200
Time (ms)
Basic Hill
Random Restart
Random Walk
Simulated Anneal
Genetic Algorithms


7x7
12.000
10.000
8.000
6.000
4.000
2.000
0.000
Average K Value
0.010
0.020
0.040
0.060
0.080
0.100
0.120
0.140
0.160
0.180
0.200
Time (ms)
Basic Hill
Random Restart
Random Walk
Simulated Anneal
Genetic Algorithms


9x9
Average K Value
12.000
10.000
8.000
6.000
4.000
2.000
0.000
0.010
0.020
0.040
0.060
0.080
0.100
0.120
0.140
0.160
0.180
0.200
Time (ms)
Basic Hill
Random Restart
Random Walk
Simulated Anneal
Genetic Algorithms


11x11
Average K Value
12.000
10.000
8.000
6.000
4.000
2.000
0.000
0.010
0.020
0.040
0.060
0.080
0.100
0.120
0.140
0.160
0.180
0.200
Time (ms)
Basic Hill
Random Restart
Random Walk
Simulated Anneal
Genetic Algorithms

Task 8. Hardest Puzzle

To create the hardest puzzle, originally we used basic hill climbing for its reliability and got an 11x11 puzzle with K = 103 and got stuck for over 30000 runs. Then we tried simulated annealing, using t=10 and decay rate=0.5, which ran much faster and produced a puzzle whose K is 109:

7	1	9	7	7	5	5	8	10	7	6
9	4	9	1	2	4	6	9	6	2	6
8	9	5	1	8	6	5	6	2	3	3
10	3	5	3	5	6	7	5	4	7	1
6	1	2	5	2	2	3	6	5	4	5
6	6	1	5	6	5	5	3	6	2	3
2	1	3	7	5	3	2	5	5	3	5
10	4	5	1	2	5	4	5	6	7	8
9	1	5	7	6	5	2	5	3	7	6
7	1	7	5	2	2	6	2	6	7	1
5	1	3	6	7	7	3	4	7	5	0

0	61	19	99	67	65	60	1	74	18	66
56	62	34	102	54	63	55	45	33	44	53
72	6	4	103	11	40	23	5	73	22	7
105	80	4	104	13	37	25	3	31	14	106
83	88	89	28	90	86	84	87	29	85	107
58	60	97	98	10	64	59	9	96	16	8
82	81	48	51	91	49	93	47	50	92	52
1	79	3	100	68	39	24	4	32	17	2
42	78	3	101	10	41	94	2	95	43	9
71	77	20	29	69	38	70	39	30	21	108
57	76	35	27	12	36	26	46	75	15	109