



JIASHU LIAO      ZIRU LIU  
OF RUTGERS

ASSIGNMENT 1

CS 440

---

# Intro to Artificial Intelligence

---

*Submitted to:*

Pro. Kostas Bekris

Rahul Shome

Chaitanya Mitash

## Contents

1	Task 1	2
2	Task 2	3
3	Task 3	5
4	Task 4	8
5	Task 5	11
6	Task 6	14
7	Task 7	17
8	Task 8	19

# 1 Task 1

This task asked us to randomly generate a  $n \times n$  puzzle. In the assignment  $r$  is the row axis of the current cell,  $r_{max}$  is the max step from the current cell to the bound,  $c$  is the column axis of the current cell,  $c_{max}$  is the max step from current cell to the bound. In this puzzle, we need to follow these rules:

1. Row and column for the puzzle should be same
2. The size of the puzzle should be in range in 5, 7, 9, 11 or larger odd numbers.
3. All cells in the puzzle should have an random step number basing on the following rules(Expect goal cell and center cell):

Set Legal Step number as  $S$ .

$$S_{max} = \text{Max}\{r_{max} - r, r - r_{min}, c_{max} - c, c - c_{min}\} \quad (1)$$

$$S_{min} = 1 \quad (2)$$

4. For the goal cell, the step should always be 0, and for the center cell. The  $S$  of center cell should follow this equation:

$$S_{max}^{center} = \text{Max}\{n/2 - 1\} \quad (3)$$

$$S_{min}^{center} = 1 \quad (4)$$

## 2 Task 2

This task asked to evaluate the puzzle. The value is how many steps that go from the left right start cell to the right bottom goal cell

The assignment asked to do the evaluation in BFS. In order to do so, I used two queue to do BFS. The first queue is called Parent Queue which store all cells that need to search their child cells. The other one is called Child Queue which stored all cells that are not visited yet. The algorithm will keep searching until there are no reachable cell.

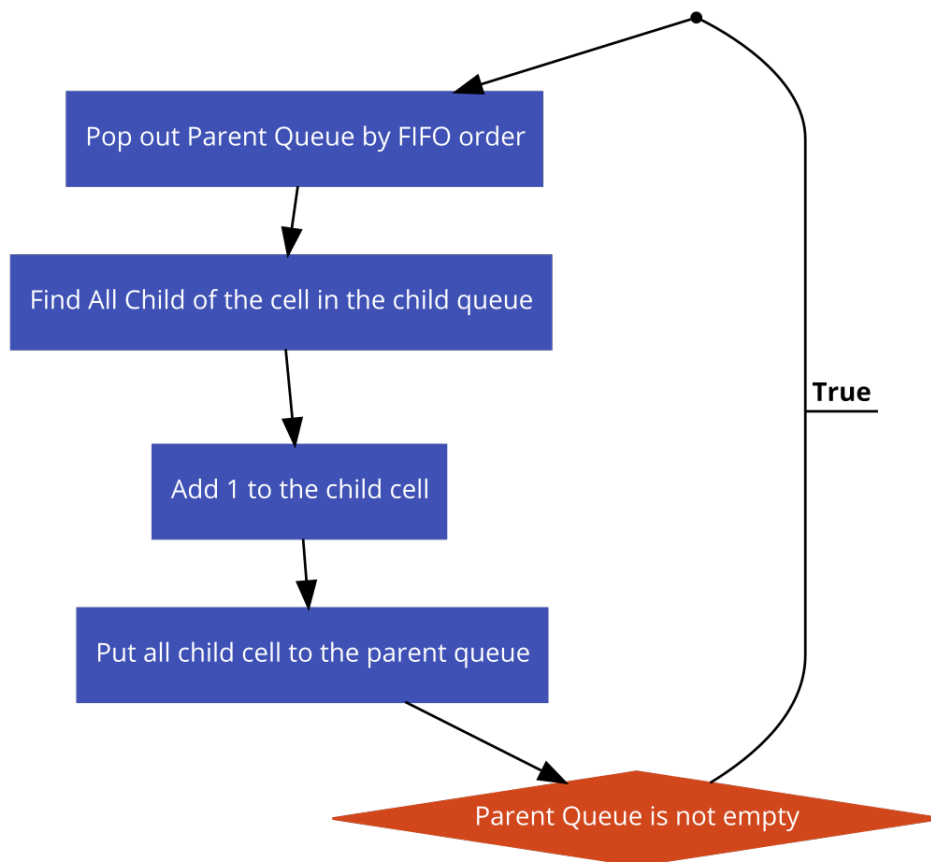
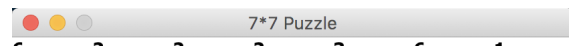


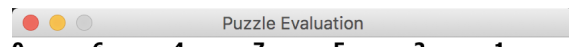
Figure 2.1 Structure of Puzzle Evaluation



A window titled "7\*7 Puzzle" with three colored circles (red, yellow, grey) in the top-left corner. Below the title bar is a 7x7 grid of numbers.

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

Figure 2.2



A window titled "Puzzle Evaluation" with three colored circles (red, yellow, grey) in the top-left corner. Below the title bar is a 7x7 grid of numbers and 'X' characters.

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

Figure 3.1

For figure 2.2 puzzle 7\*7, we can easily get a new puzzle after evaluation as figure 2.3

### 3 Task 3

In task 3, the algorithm should randomly changed a step number in a non goal cell by following equation (1)(2)(3)(4) showed in task 1. In this puzzle, we need to follow these rules:

The following plots are based on Avg result of 50 times of operation in size 5, 7, 9, 11. Each operation is based on 2000 iteration time by Pure Hill Climbing Algorithm.

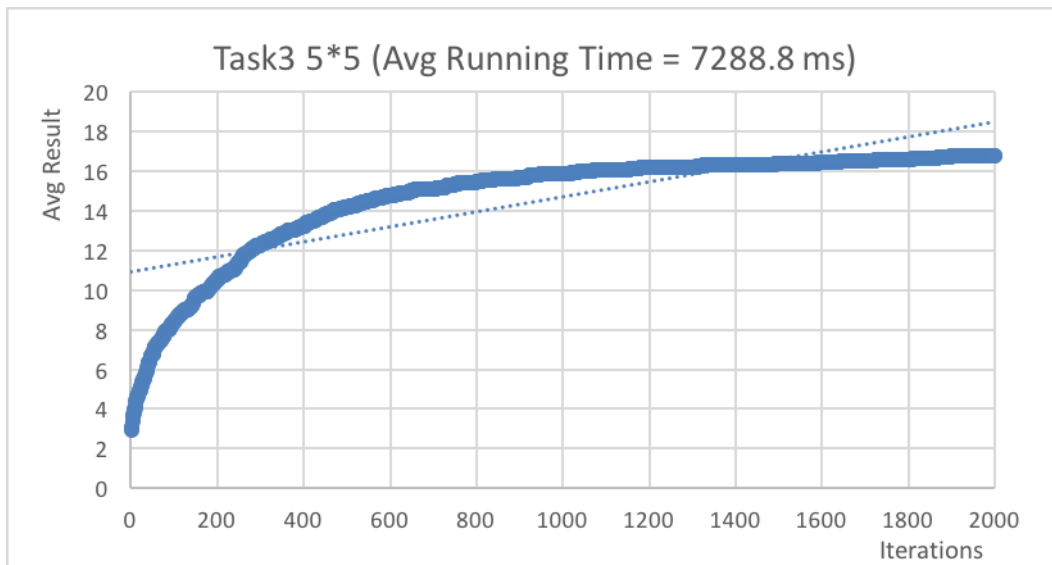


Figure 3.1

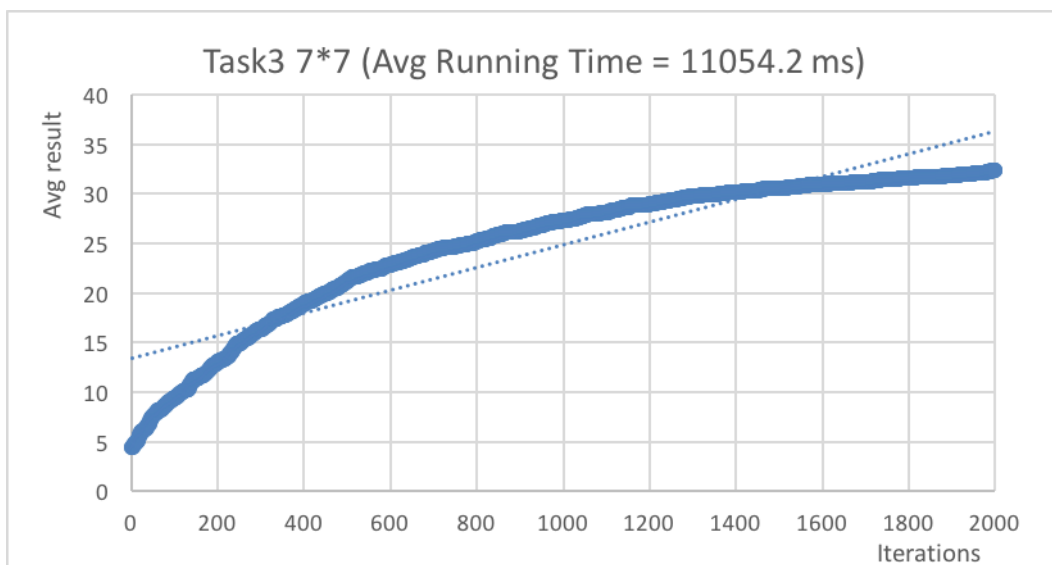


Figure 3.2

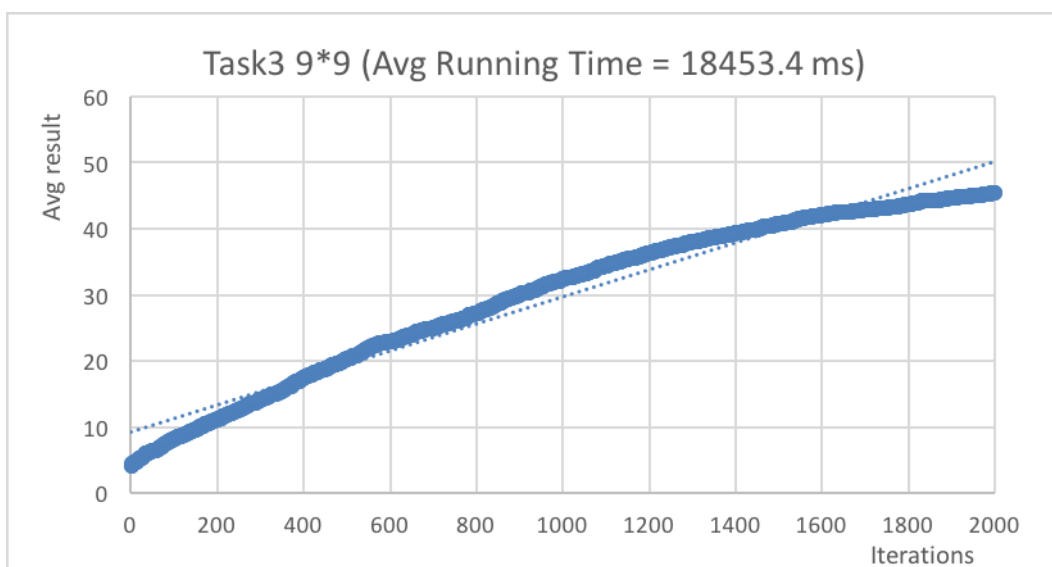


Figure 3.3

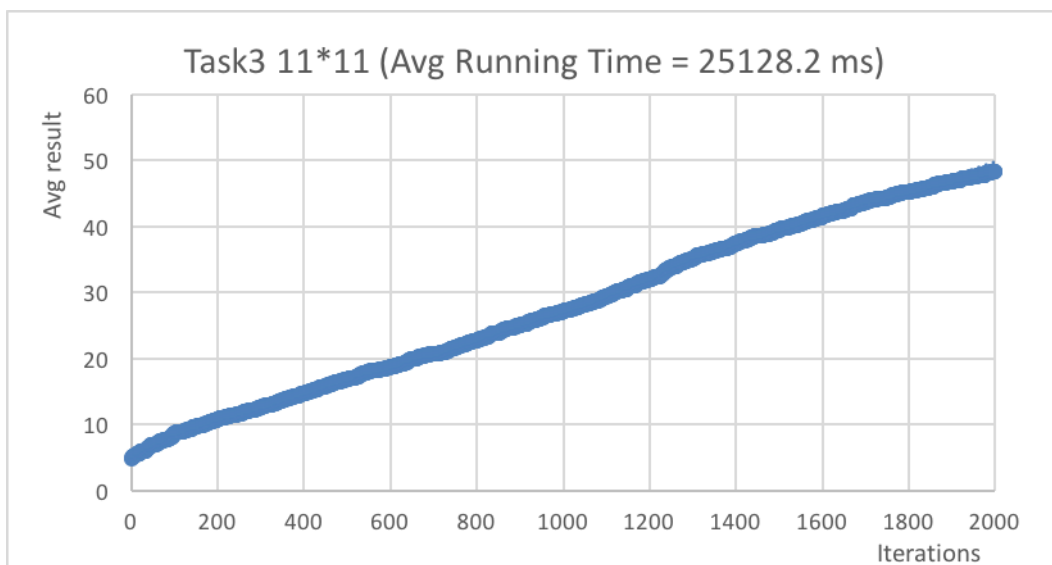


Figure 3.4

From the plots above, the basic hill climbing algorithm reach the plateau near 900 iteration in size 5. It reaches plateau near 1800 in size 7 and 8. And for the size 11, due to the limit of iteration it did not reach the plateau.



## 4 Task 4

In task 4, the algorithm is based on the basic hill climbing, but it will restart when it reach the target iteration time and keep track the best result:

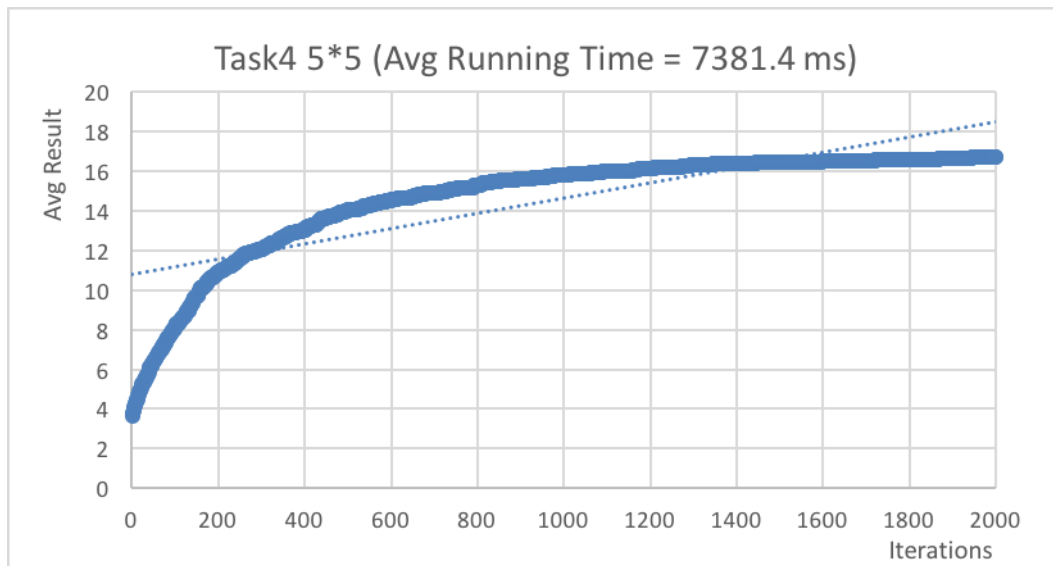


Figure 4.1

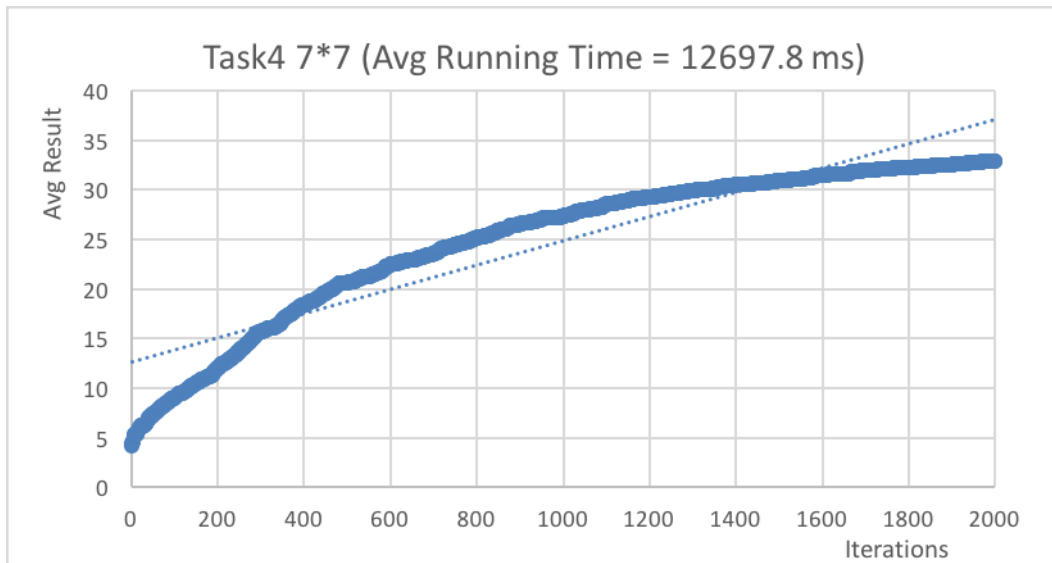


Figure 4.2

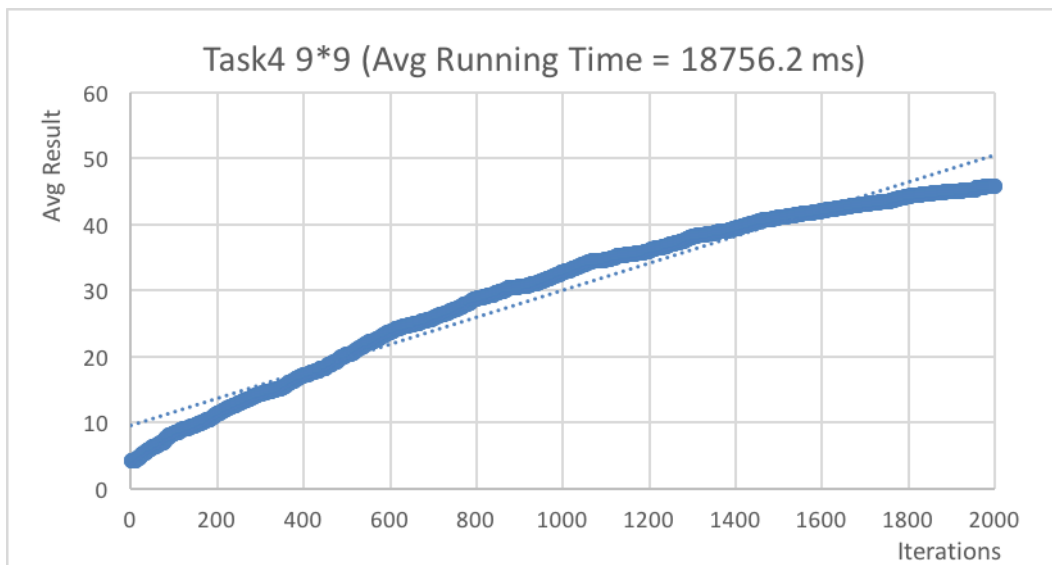


Figure 4.3

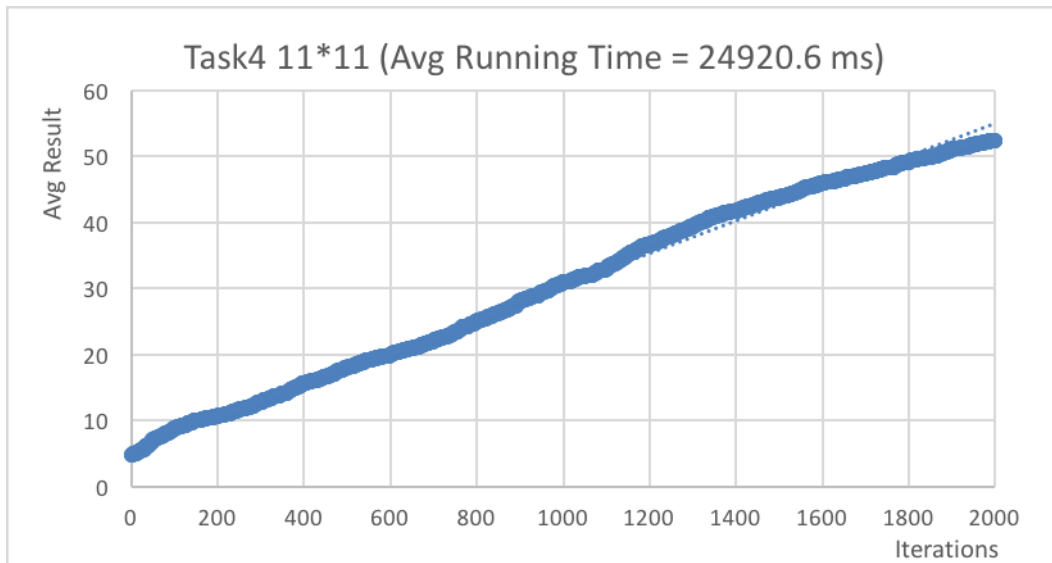


Figure 4.4

Comparing to the result of Task3, Task 4 did not show a great advantage at most time, but task 4 showed a better gradient near 2000. Which means that the task 4 has a better ability to reach better results in more iteration cases. During collecting data, we found out that the best restart time is after reaching the plateau. For example, for size 5 puzzle, we should restart around 500 iteration, because after 500 iteration the algorithm will go to a plateau. Restarting will give more chances for the algorithm to leave the plateau.

## 5 Task 5

In the task 5, the algorithm is based on the basic hill climbing algorithm. However, it requires a fix possibility to accept the worse result. Which means sometimes the algorithm will accept the random change that makes the result worse. This is so called down hill climbing.

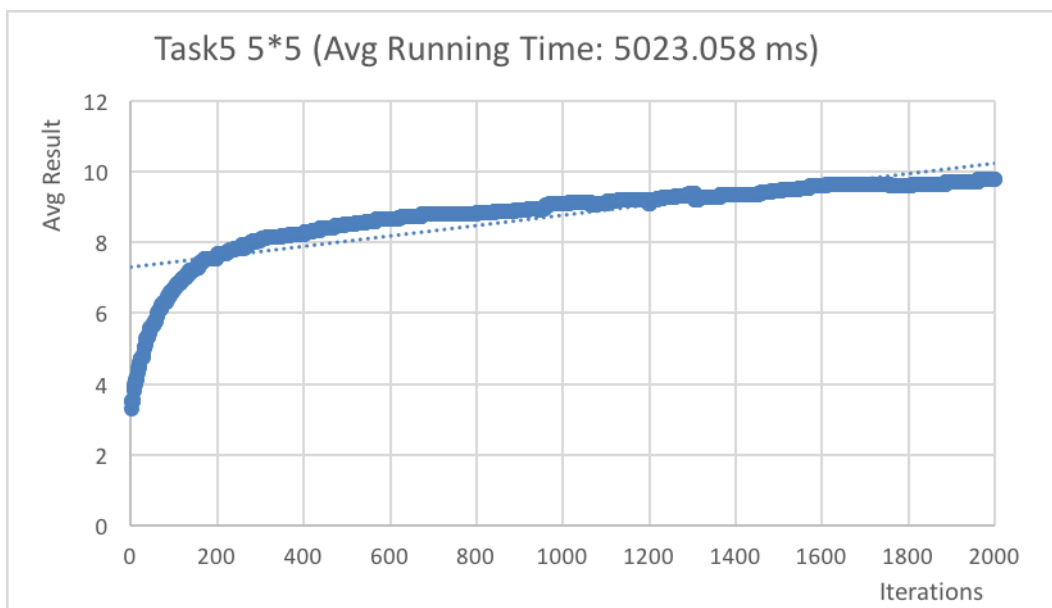


Figure 5.1

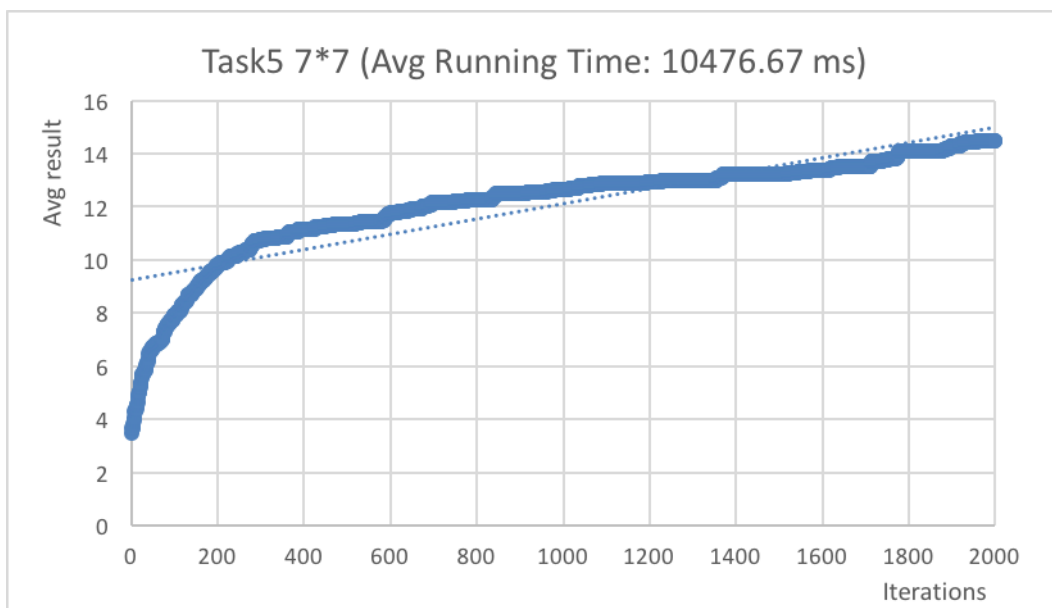


Figure 5.2

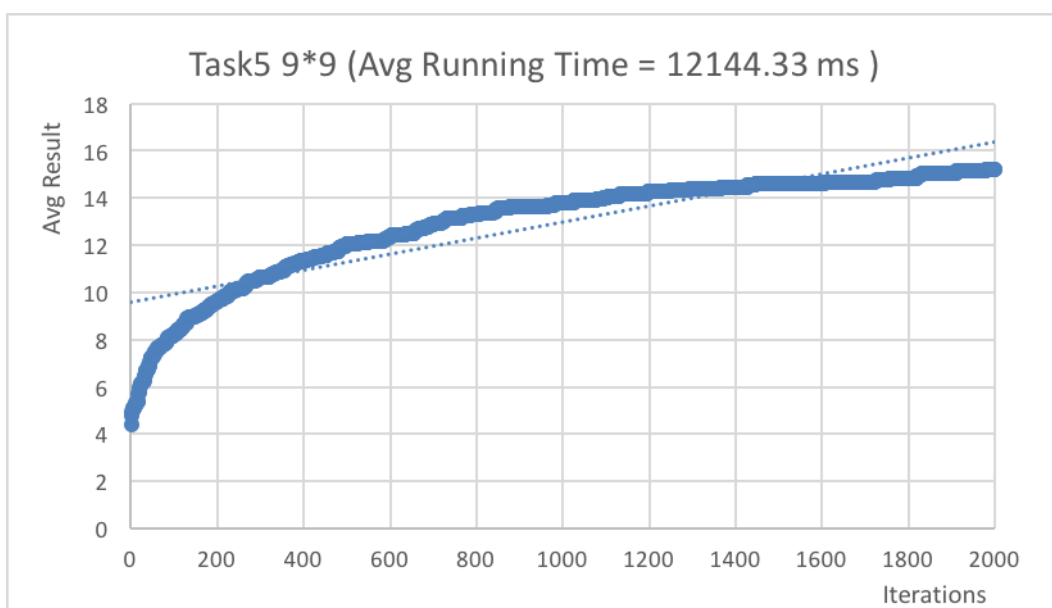


Figure 5.3

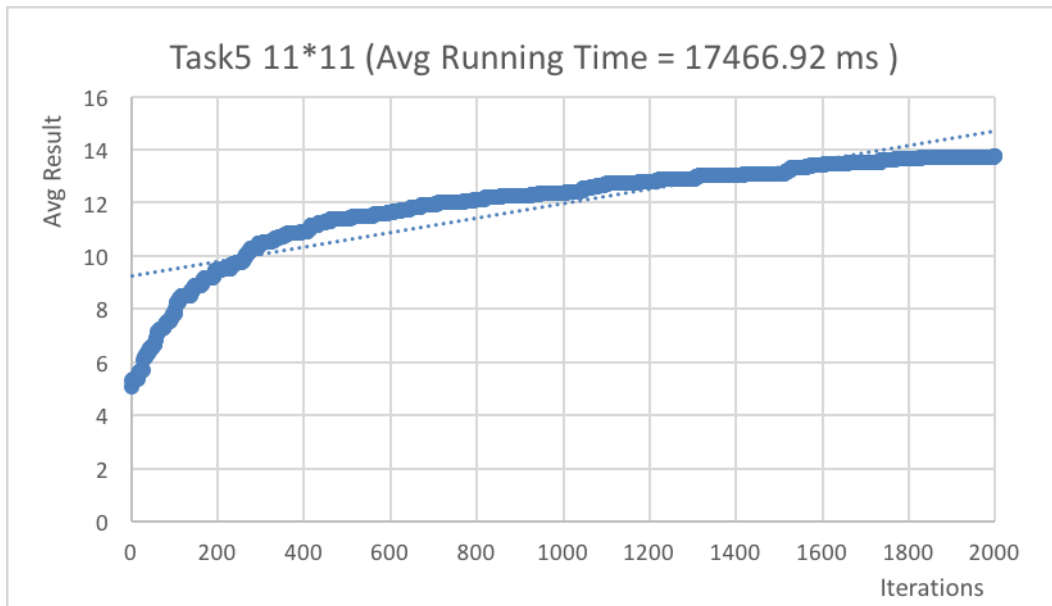


Figure 5.4

For the task5 we set the fix possibility as 0.2. The reason is that if we set a high possibility to accept the worse result. While the iteration go to higher, the higher possibility for the algorithm to meet worse result will increase. While it go to a high iteration time, it will waste a lot of time.

## 6 Task 6

In the task 6, the algorithm is based on the a s same as task5. However, it's possibility to accept the worse result become a variable. As we set a temperature  $T$  and  $V$  as the result of the puzzle,  $d$  as the decay rate. It follows the equations:

$V'_j$  as new result of puzzle  $V_j$  as last result of puzzle  $V_j$  as last result of puzzle  $T_0$  as last or initial temperature

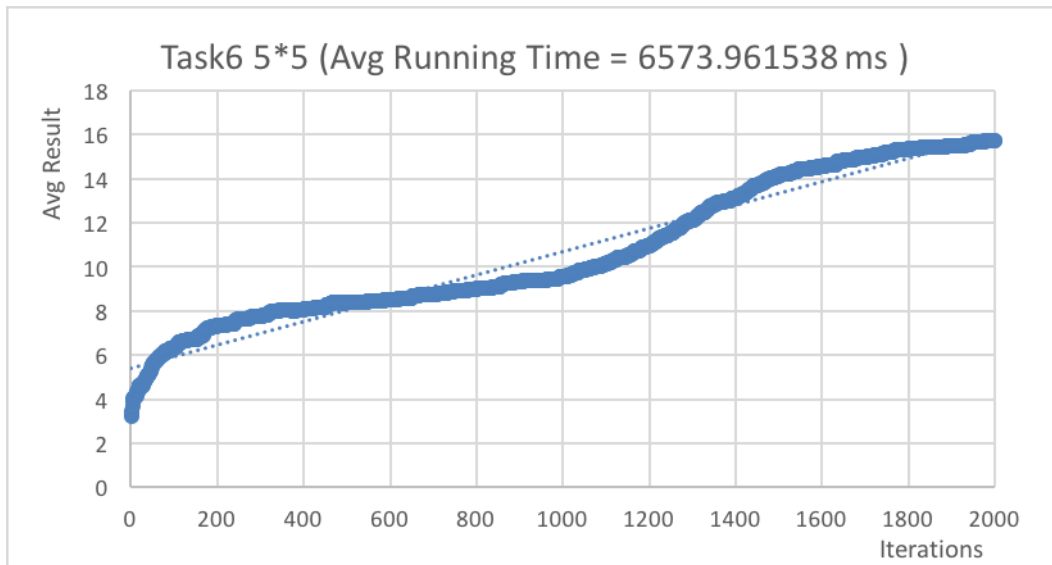


Figure 6.1

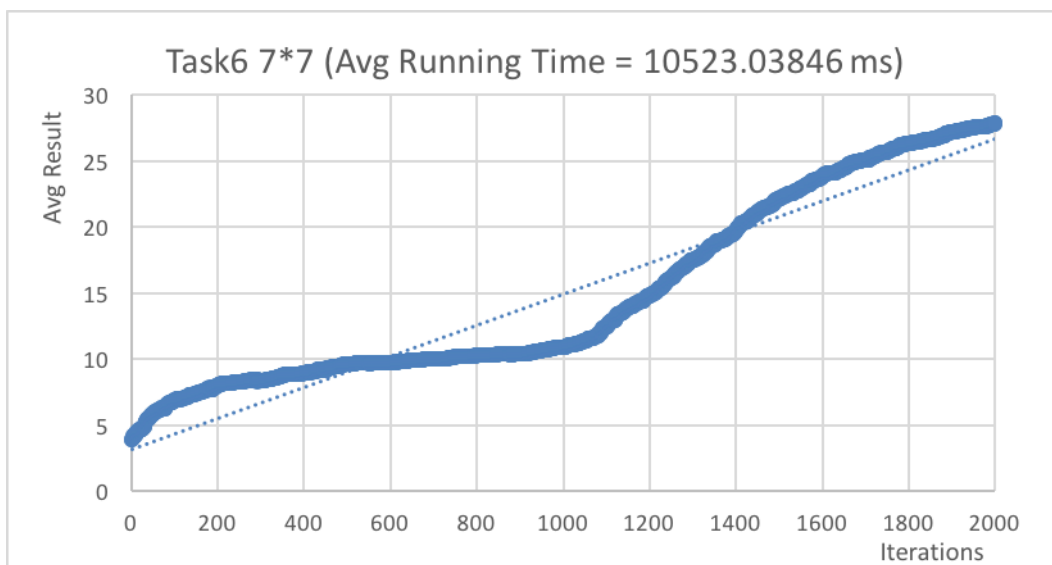


Figure 6.2

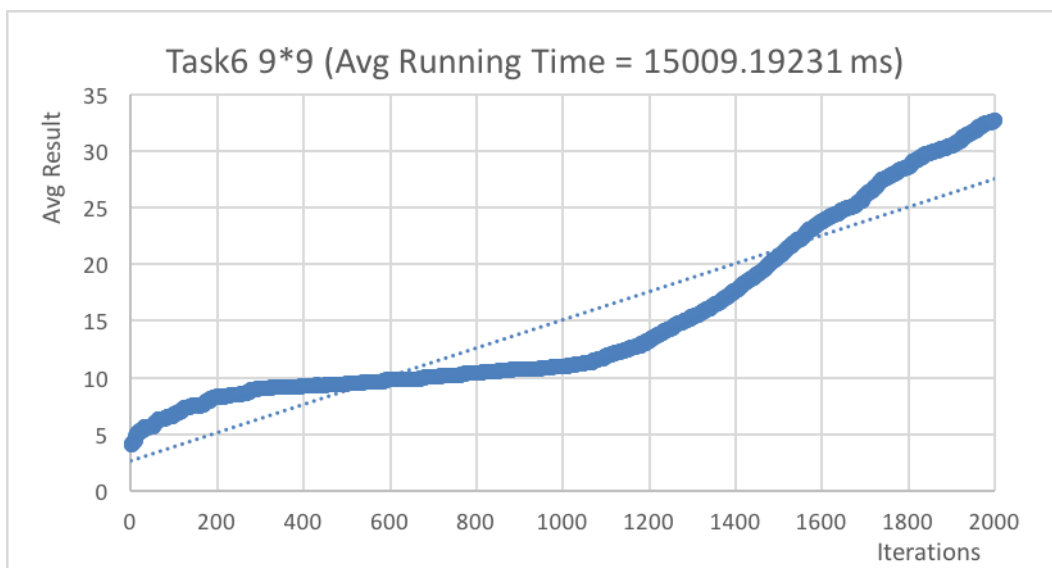


Figure 6.3



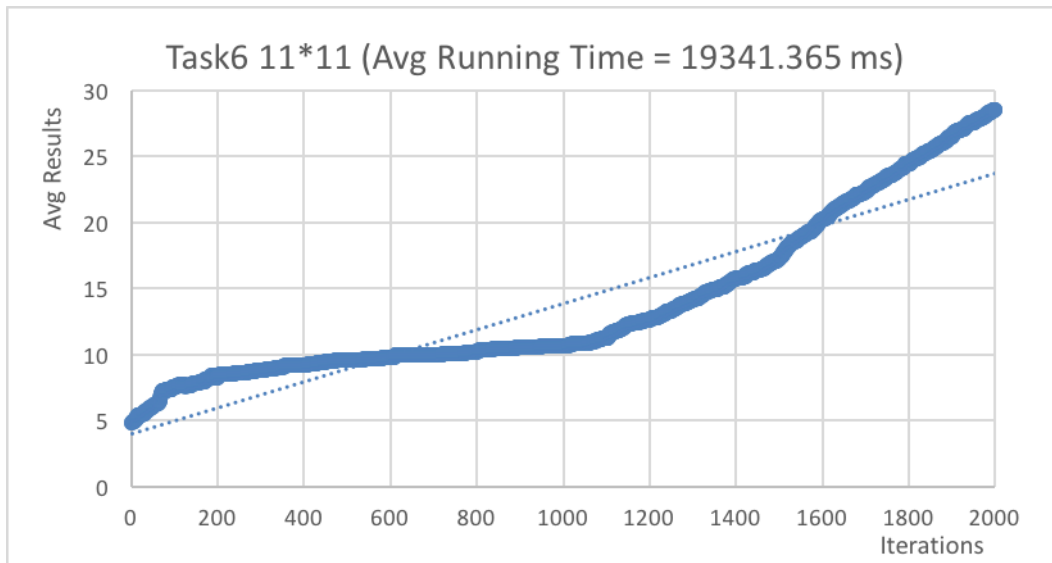


Figure 6.4

In task 6, since we set iteration number as 2000. We need to set an initial temperature as high enough and decay rate that fit with the temperature with 2000 iteration. Considering that we need get the advantage of down hill climbing while facing the plateau problem. We decide to set initial temperature as 20000 and decay rate as 0.99. Thus we can obviously observe the rise up in the mid bottom.

## 7 Task 7

In task 7, we used genetic algorithm which is based on natural selection. First we need to get 4 random puzzle, then we evaluate them. After evaluation, we will drop the lowest one. Second we will count the weight for the resting 3 puzzles. We will randomly pick 1 of the 3 basing on weight of them to duplicate itself. Third, we will use the duplicated puzzle to crossover with the other 2 puzzles. Four, we will get 4 new puzzles after crossover. Then we will mutate 3 of them (randomly). Then we will repeat the first step. During the algorithm, we will keep tracking the best result.

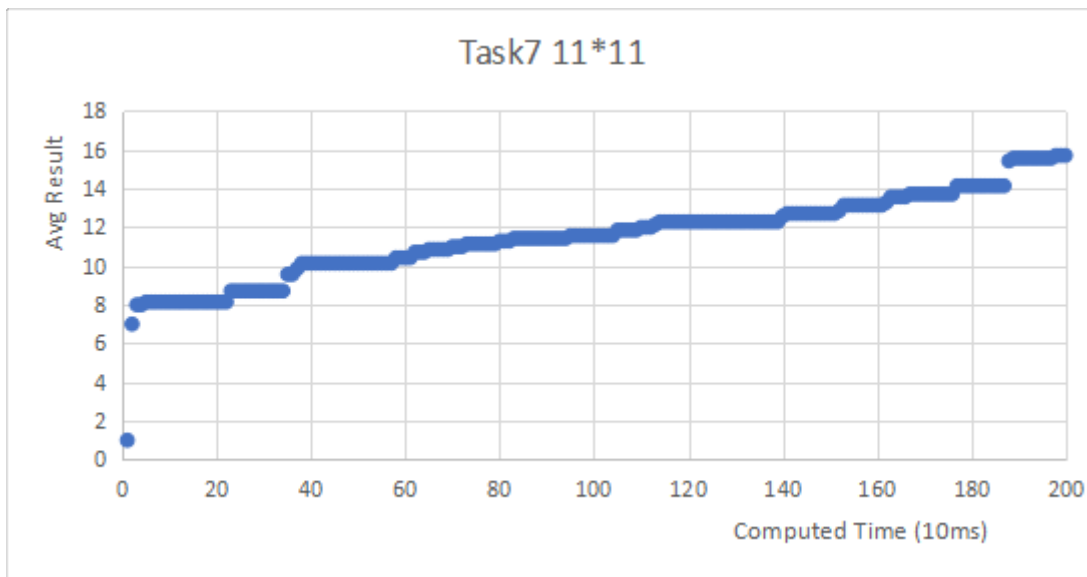


Figure 7.1

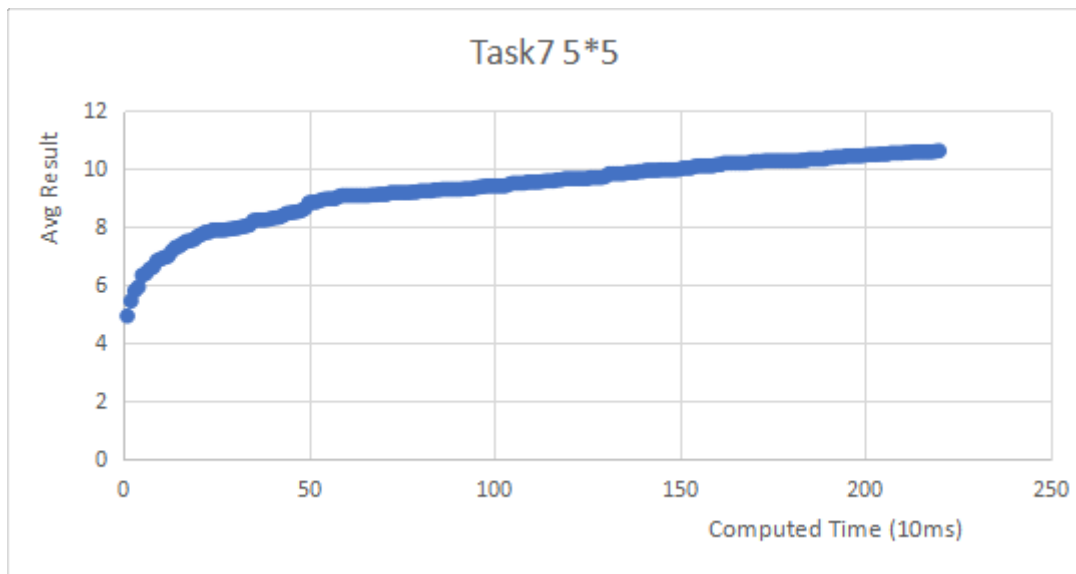


Figure 7.2

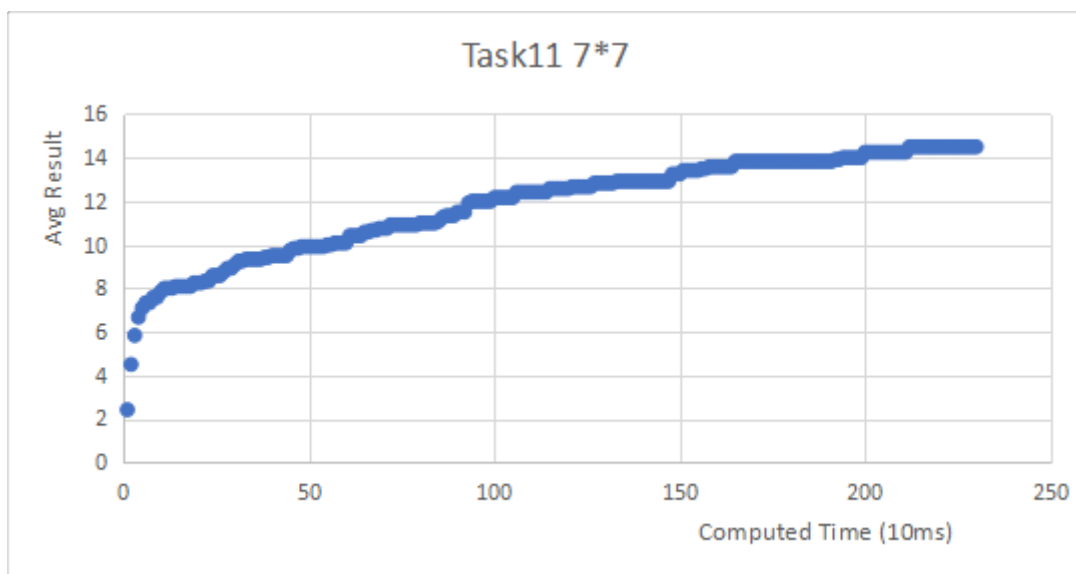


Figure 7.3

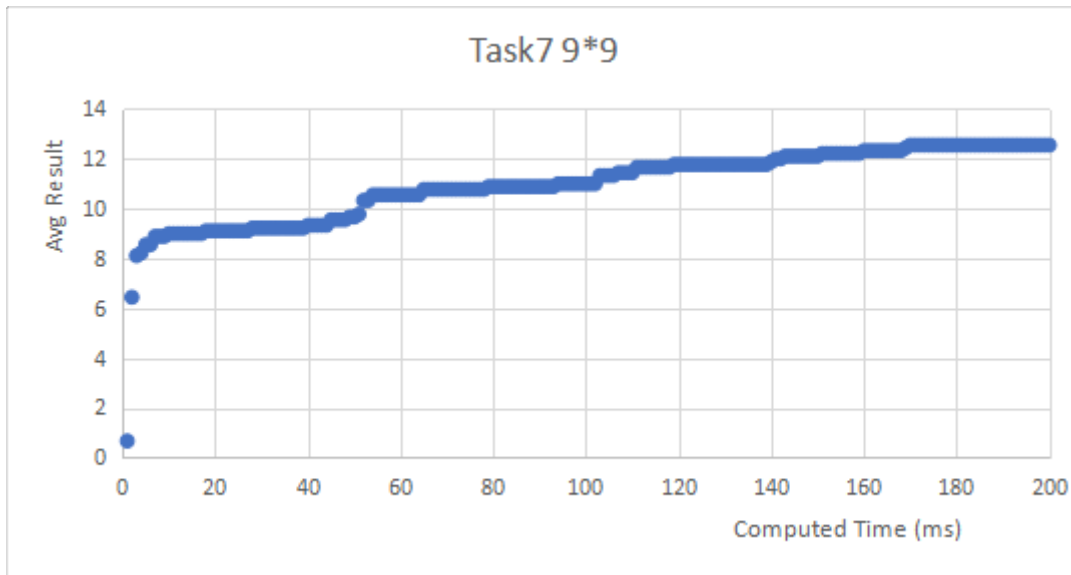


Figure 7.4

By comparing all figure in task 3, 4, 5, 6, 7, we can obviously get the assumption that the task 7 and task 6 and task 4 all have a better performance than the rest while the size is large or the iteration is high enough.

## 8 Task 8

In task 8, we set size as 17 and iteration number as 150000. By using random restart algorithm, we get a result as 251 when the maximum possible result is 289.

Random Restarts																	—	□	×
0	14	98	85	140	234	27	120	19	5	1	160	13	220	97	18	131			
59	178	58	82	52	94	115	116	22	202	5	114	196	222	95	169	113			
46	245	42	73	240	239	25	34	244	203	153	32	242	241	47	243	246			
60	90	8	86	120	92	103	119	20	189	7	104	87	61	91	166	250			
207	176	209	212	138	135	205	210	213	204	3	136	137	206	211	208	248			
123	30	4	75	227	228	29	32	183	193	109	30	31	225	226	5	130			
44	37	41	83	50	237	42	40	216	188	152	39	38	39	49	40	43			
55	16	146	145	53	230	148	56	18	201	149	161	39	144	147	17	54			
63	179	64	106	67	236	104	109	65	199	108	159	107	62	68	105	66			
79	180	181	155	183	233	186	185	182	187	154	158	157	156	78	184	247			
1	35	33	84	141	93	30	117	214	200	151	40	32	34	2	167	131			
11	13	9	72	163	238	102	110	218	192	8	162	12	219	10	164	111			
45	89	3	87	139	232	128	126	217	4	2	171	127	221	88	170	129			
122	175	173	76	142	235	26	121	21	191	78	172	79	143	77	174	249			
56	177	57	71	X	134	28	118	19	198	150	70	197	133	69	18	132			
124	36	100	81	51	231	101	125	215	190	4	31	80	223	96	168	112			
47	15	99	74	49	229	24	33	23	194	6	160	195	224	48	165	251			