Moderation Paper

Jacob Lester

March 2018

1 Foreword

Prior to enrolling in Bard College I had no experience in computer science. I chose computer science because of my interest in quantum computation. I am joint majoring into mathematics because of the application it has to most every study/profession. I am also studying finance because of interest and it's a good field to apply my studies towards. I chose not to major in physics because my interests lie in the more abstract and quantum that would require classes I had very little interest in.

In 2016, while looking in my bathroom mirror, I came up with an idea that shook me with excitement. What if an algorithm was developed to simulate evolution on a quantum computer? By defining the environment, the algorithm could in concept replace the need for human creativity. This algorithm would plug and chug until developing a Dinosaur of a solution, evolutionary perfected. Application could include discovery of new materials, development of machinery blueprints, and even GMOs. I wrote this idea down on my phone and titled the concept 'Quantum Darwinism.'

Last semester I took Intro to Artificial Intelligence and discovered something incredible- Genetic Algorithms. My idea had already been pursued! Well- it was being pursued but had less following than other MLA's because of it's heavy processing requirements. After the semester ended, I devoted 15 hours a week to developing a simple GA that solved a maze. I built the GA from scratch and without numpy. Over Spring Break I have altered it runs a Traveling Salesperson-esque problem instead of just a Maze. I believe that with the imminent and inevitable development of functional quantum computers, Genetic Algorithms will reign supreme as the mainstream MLA approach. As such, I am dedicating my moderation project to this simple GA I have created.

2 Abstract

In this project I will be outlining my MazeRunner GA and contrasting two versions. In version-48, organisms are constituted by 48 bits. In version-24, organisms are constituted by 24 bits. For each organism of both versions, adjacent pairs of bits represent a direction the organism will move in. The fitness of the organism is increased by visiting more unique nodes and decreased by path costs. The organism will run the maze until it reaches the end of it's sequence or visits all nodes - whichever comes first. The population size was kept constant at 90. The number of epochs was constant at 6. The path-cost multiplier was variable at 0, 2, and 10. Each case was run three times for a total of 18 times, 9 for each version (24 and 48).

3 Results

It is to be noted that the column labeled 'COSTS' is in fact fitness. U.N.V stands for Unique Nodes Visited. The max is 11.

3.1 48bit Version Data Collection

	COSTS U.N.V.					
48bit CM = 0	**Run 1**		**Run 2**		**Run 3**	
	1100	11	1000	10	1100	11
	1100	11	1000	10	1100	11
	1100	11	1100	11	1100	11
	1100	11	1100	11	1100	11
	1100	11	1100	11	1100	11
	1100	11	1100	11	1100	11
48bit CM = 2	**Run 1**		**Run 2**		**Run 3**	
	886	10	948	11	896	10
	892	10	948	11	896	10
	954	11	948	11	896	10
	954	11	948	11	960	11
	954	11	948	11	960	11
	954	11	948	11	960	11
48bit CM = 10	**Run 1**		**Run 2**		**Run 3**	
	440	9	420	10	380	11
	450	10	440	10	460	10
	450	10	440	10	470	11
	530	11	440	10	490	9
	530	11	440	10	490	9
	530	11	440	10	490	9

3.2 24bit Version Data Collection

	COSTS U.N.V.					
24bit CM = 0	**Run 1**		**Run 2**		**Run 3**	
	800	8	900	9	900	9
	900	9	900	9	900	9
	900	9	900	9	900	9
	900	9	900	9	900	9
	900	9	900	9	900	9
	900	9	900	9	900	9
24bit CM = 2	**Run 1**		**Run 2**		**Run 3**	
	826	9	744	8	756	8
	826	9	844	9	756	8
	936	10	844	9	756	8
	936	10	844	9	756	8
	936	10	844	9	756	8
	936	10	844	9	756	8
24bit CM = 10	**Run 1**		**Run 2**		**Run 3**	
	520	8	650	10	450	7
	520	8	650	10	630	9
	620	9	650	10	630	9
	620	9	650	10	630	9
	620	9	650	10	660	10
	620	9	650	10	660	10

4 Analysis

The program runs rather quickly (under a second), so processing time was not an issue. In Results 3.1 when path costs were not taken into consideration the generation(self) method generated an ideal sequence in 2 of the three runs. Even still, when cost multiplier was set at 2 Run 2 generated a pretty good fitness from the get go. However, that initial generation failed in comparison to the evolved generations of the other runs. When cost multiplier was set to 10 is when the GA demonstrated the most dynamic evolution. While Run 2 was boring, Run 1 went from 9 unique nodes visited to 11 - good improvement. However, Run 3 generated a sequence that visited all nodes with but slowly decayed visited nodes to balance path costs. Further analysis is required to understand the relationship between cost-U.N.V priority.

With results 3.2, the 24bit organisms had half the life of their 48bit counterparts. With CM = 0, they always generated weaker organisms and averaged at 900 fitness as opposed to 1100 fitness of results 3.1 CM=0. When CM = 2,

the 24bit generated organisms were also weaker. They were also less dynamic in evolution. When CM = 10, runs 1 and 2 met the patter of less diverse evolution, but in fact generated stronger organisms than the 48bit program. Further investigation is required to understand why this is. Run 3 was a discrepancy that was not replicated in any other runs in that it experienced the most dynamic growth of fitness and U.N.V.

5 Conclusion

I would like to do more with this algorithm and add in mutation rate. I did not include mutation rate but suspect that it would allow me to run longer epochs. I chose 6 epochs because after 6 iterations the species became homogeneous. Lastly, I would also like to create a more complex maze, larger population size, and 256bit organisms.

6 API Documentation

6.1 Class Attributes

6.1.1 self.maze

Stores nodes, paths, and path costs as listed tuples nested into dictionary with each node as a key. Remains constant.

6.1.2 self.popSize

Stores normal population size number. Remains constant at 90.

6.1.3 self.epochsRemain

Acts as counter for how many self.newGeneration() iterations to perform. Starts at constant 6 and decends to 0.

6.1.4 self.costsMultiplier

Changes weight of path-costs when calculating fitness. Starts at constant of either 0, 2, or 10 and remains constant.

6.1.5 self.population

Nested list that contains 90 sequences and corresponding fitness. Variable.

6.1.6 self.orgLocation

Stores an organism's current location as its fitness is calculated. Extremely variable.

6.1.7 self.visited

Stores unique visited nodes for an organisms as its fitness is calculated. Extremely variable.

6.1.8 self.costs

Stores running path-costs of organism as its fitness is calculated. Extremely variable.

6.2 Methods

6.2.1 makeMaze(self)

Creates the maze. Initiates only once. Outlined in Fig1.

6.2.2 generatePop(self)

Randomly generates 90 organism sequences of either 24bit or 48bit strings depending on version. Runs only once. Outlined in Fig2.

6.2.3 select(self)

Sorts population by ascending fitness. Eliminates 60 organisms with lowest fitness from population. Runs self.epochsRemain times. Outlined in Fig3.1 and Fig3.2.

6.2.4 newGeneration(self)

Generates a new population of 90 using 30 organisms as parents. Runs self.epochsRemain times. Outlined in Fig4.1, 4.2, and 4.3.

6.2.5 fitScore(self, seq, ID

Takes sequence and self.population index as input and calculates then appends fitness to the organism in self.population. Runs self.popSize $\times$ self.epochsRemain times. Outlined in Fig5.

6.2.6 move(self, start, direction)

Helper function for self.fitScore(seq,ID). Updates self.visited, self.costs, and self.orgLocation. Runs many many times. Outlined in Fig5.

6.2.7 goalCheck(self)

Helper function for self.fitScore(seq,ID). Stops method from running if organism has visited every node. Runs many many times.

6.2.8 crossover(parent1,parent2)

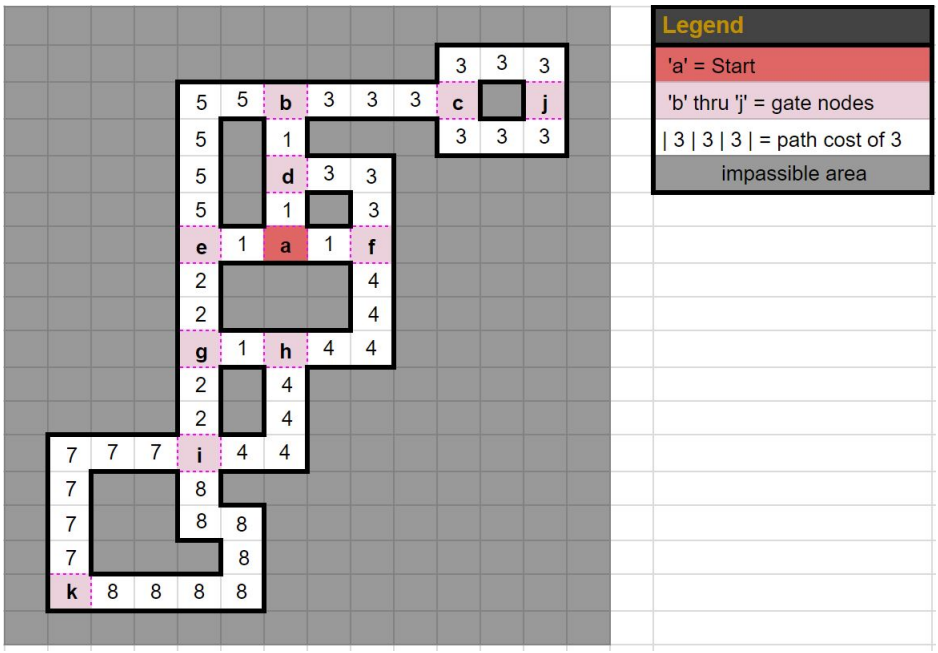
Create pairs of children by 'breeding' pairs of parent organisms. Runs 30 × self.epochs times. Outlined in Fig6.

6.2.9 display(self)

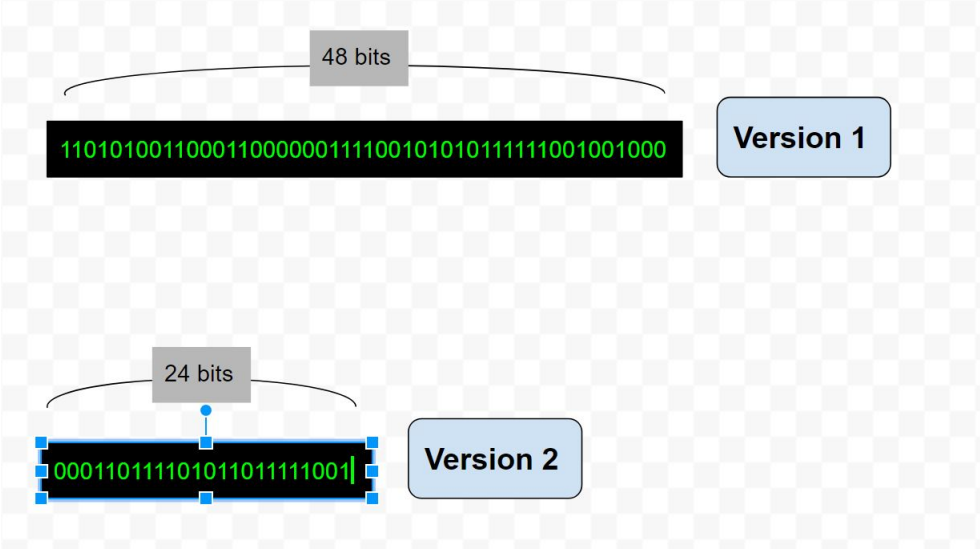
Prints most successful organism's sequence and fitness for each generation.

7 Figures

7.1 Fig1



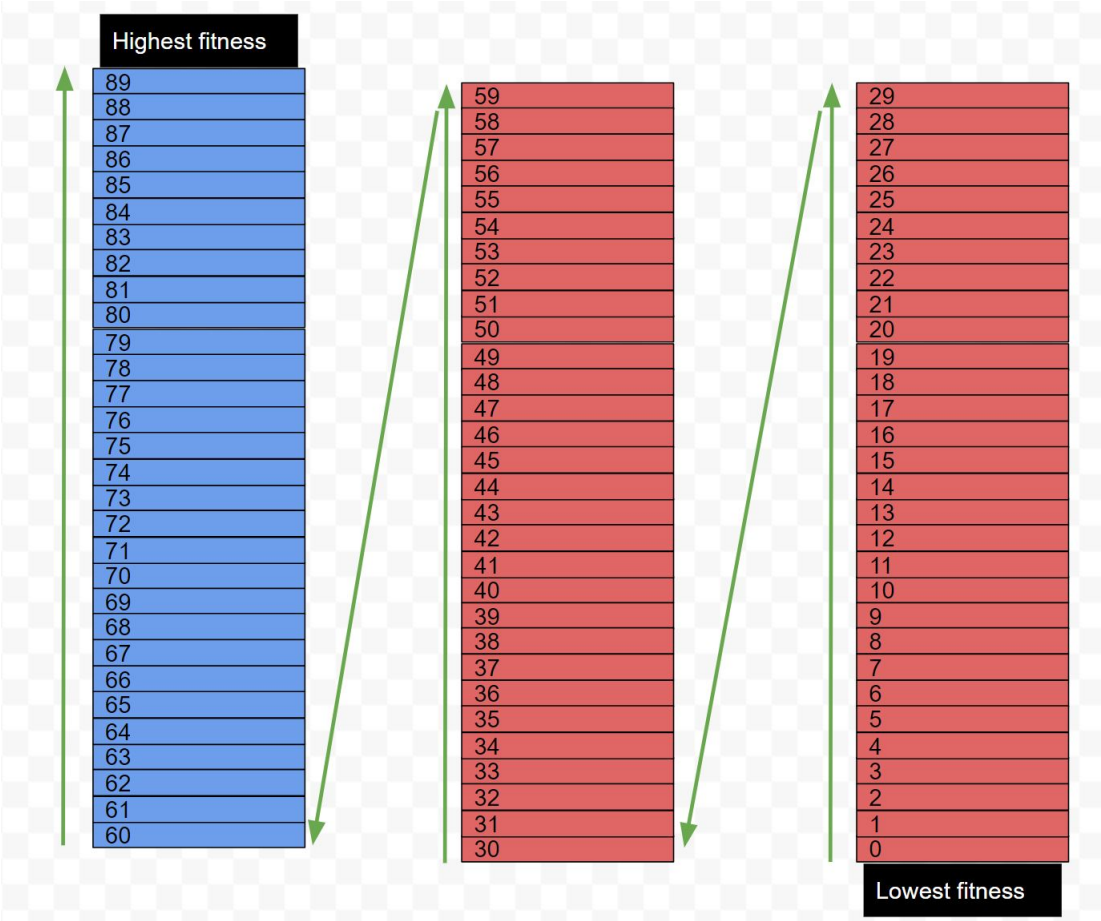
7.2 Fig2.1


48 bits
110101001100011000000111100101010111111001001000
Version 1
24 bits
000110111101011011111001
Version 2

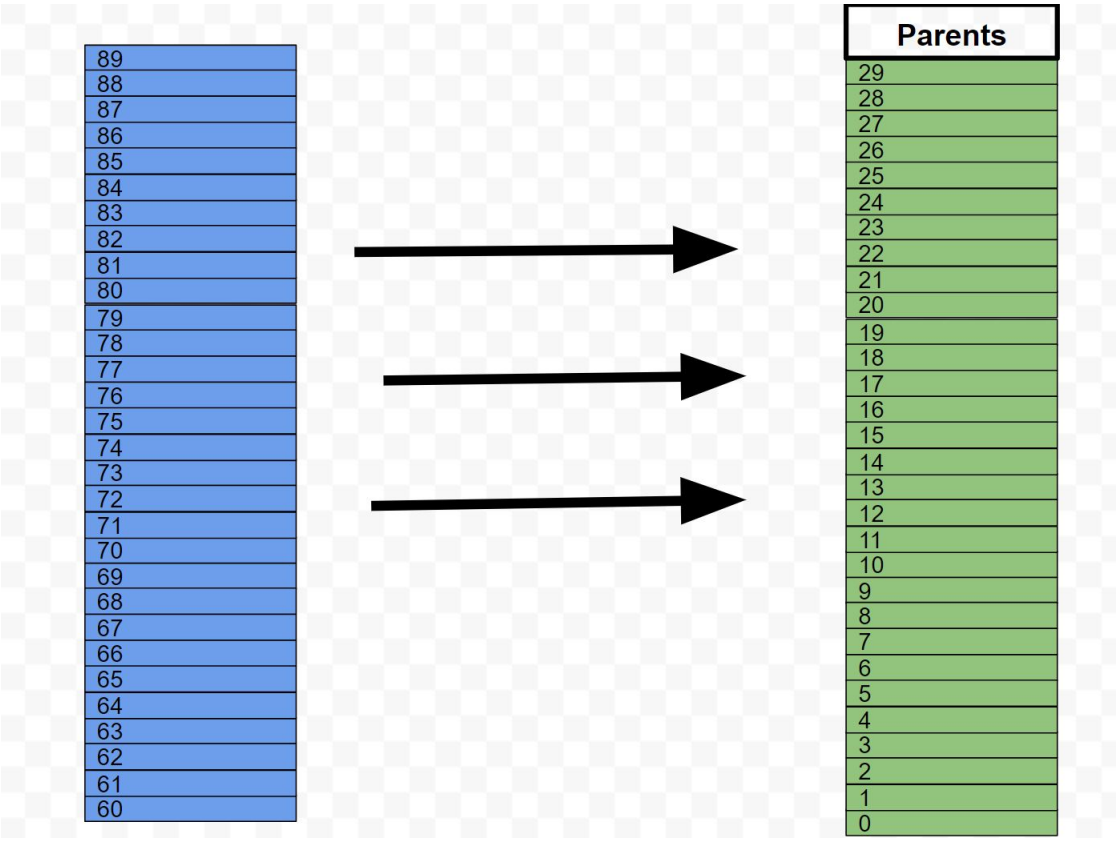
7.3 Fig2.2


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

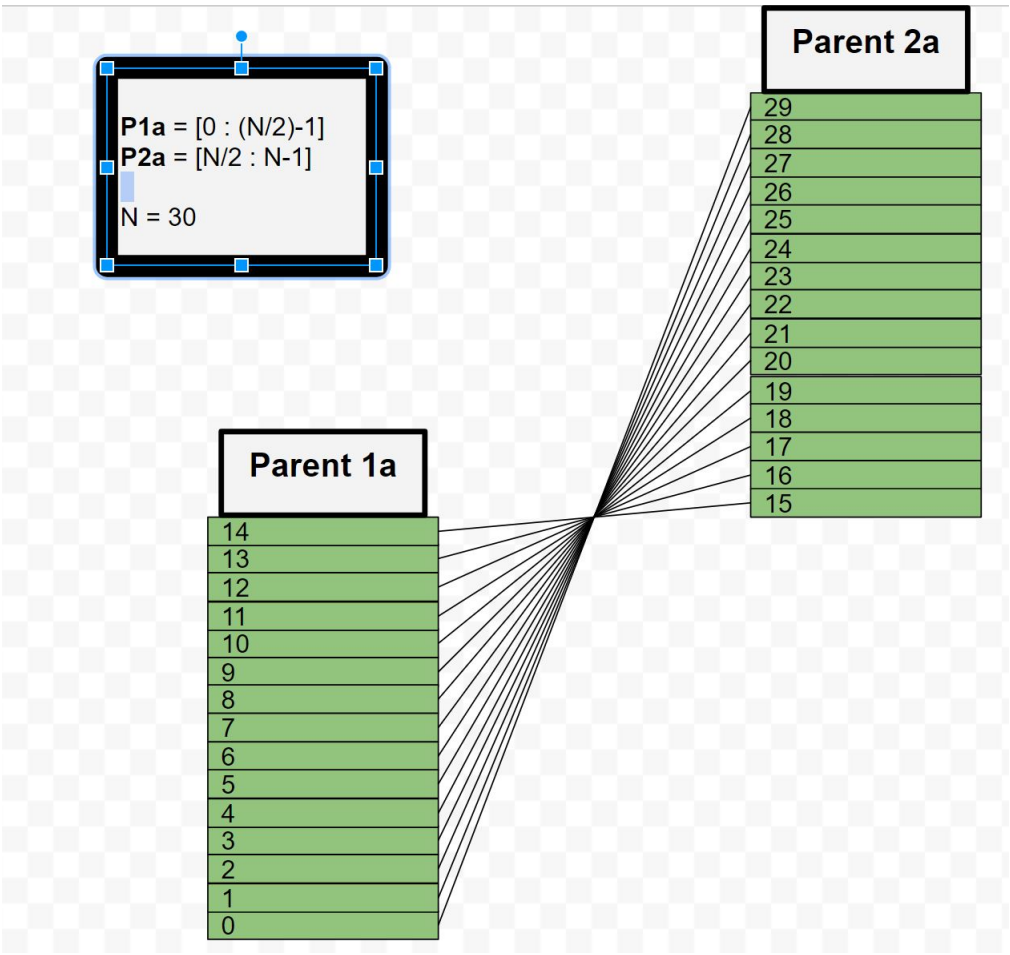
7.4 Fig3.1


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

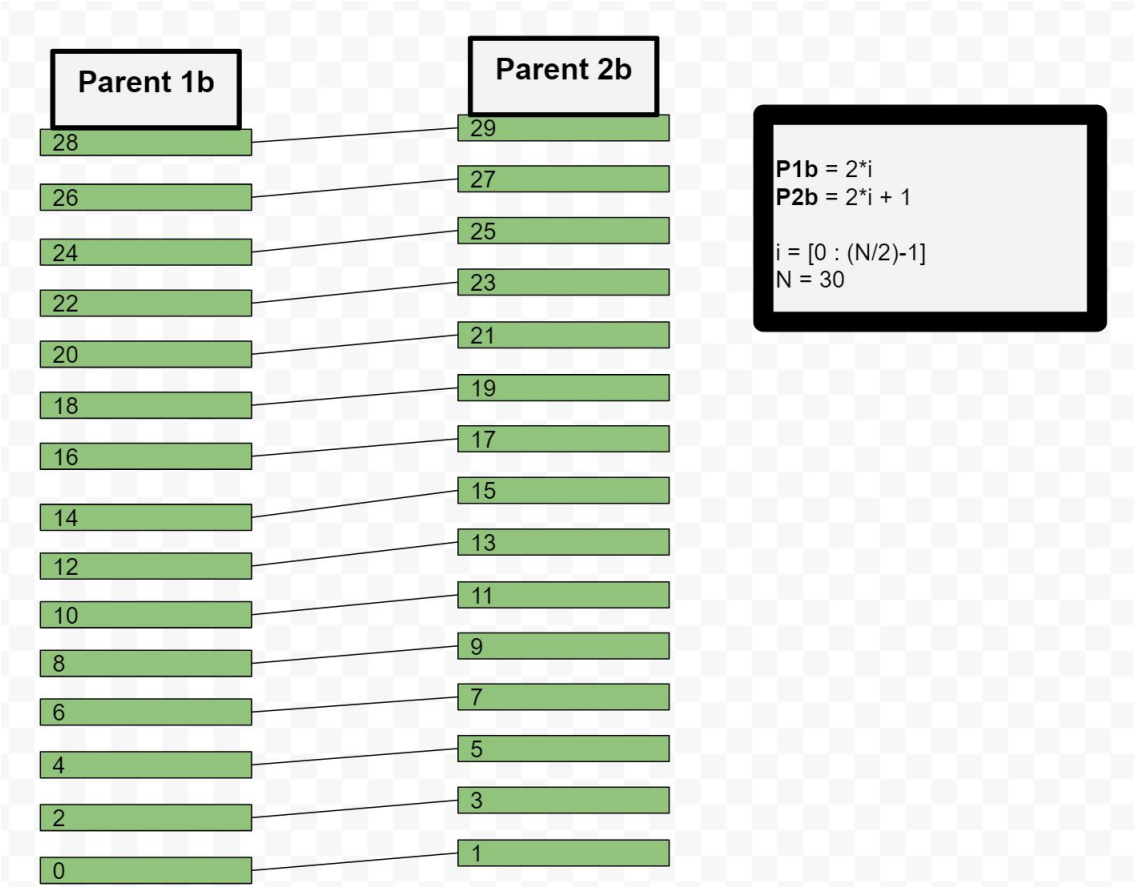
7.5 Fig3.2


89
88
87
86
85
84
83
82
81
80
79
78
77
76
75
74
73
72
71
70
69
68
67
66
65
64
63
62
61
60
Parents
29
28
27
26
25
24
23
22
21
20
19
18
17
16
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0

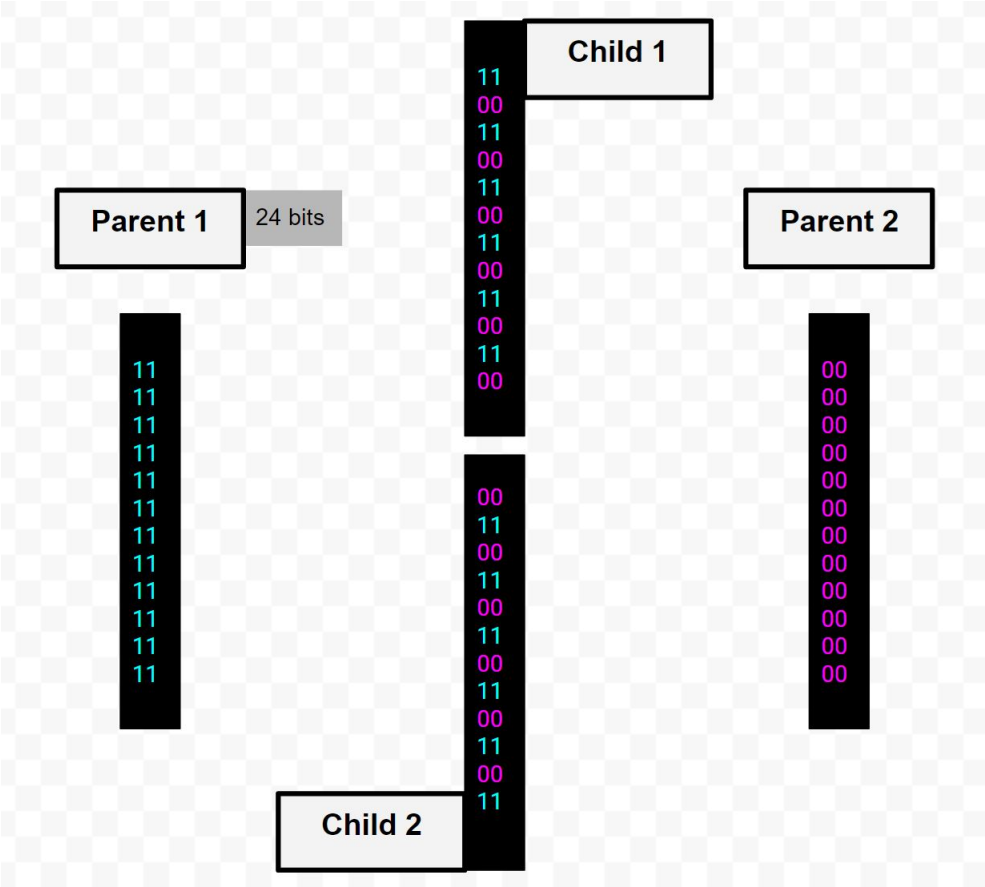
7.6 Fig4.1


P1a = [0 : (N/2)-1]
P2a = [N/2 : N-1]
N = 30
Parent 2a
29
28
27
26
25
24
23
22
21
20
19
18
17
16
15
Parent 1a
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0

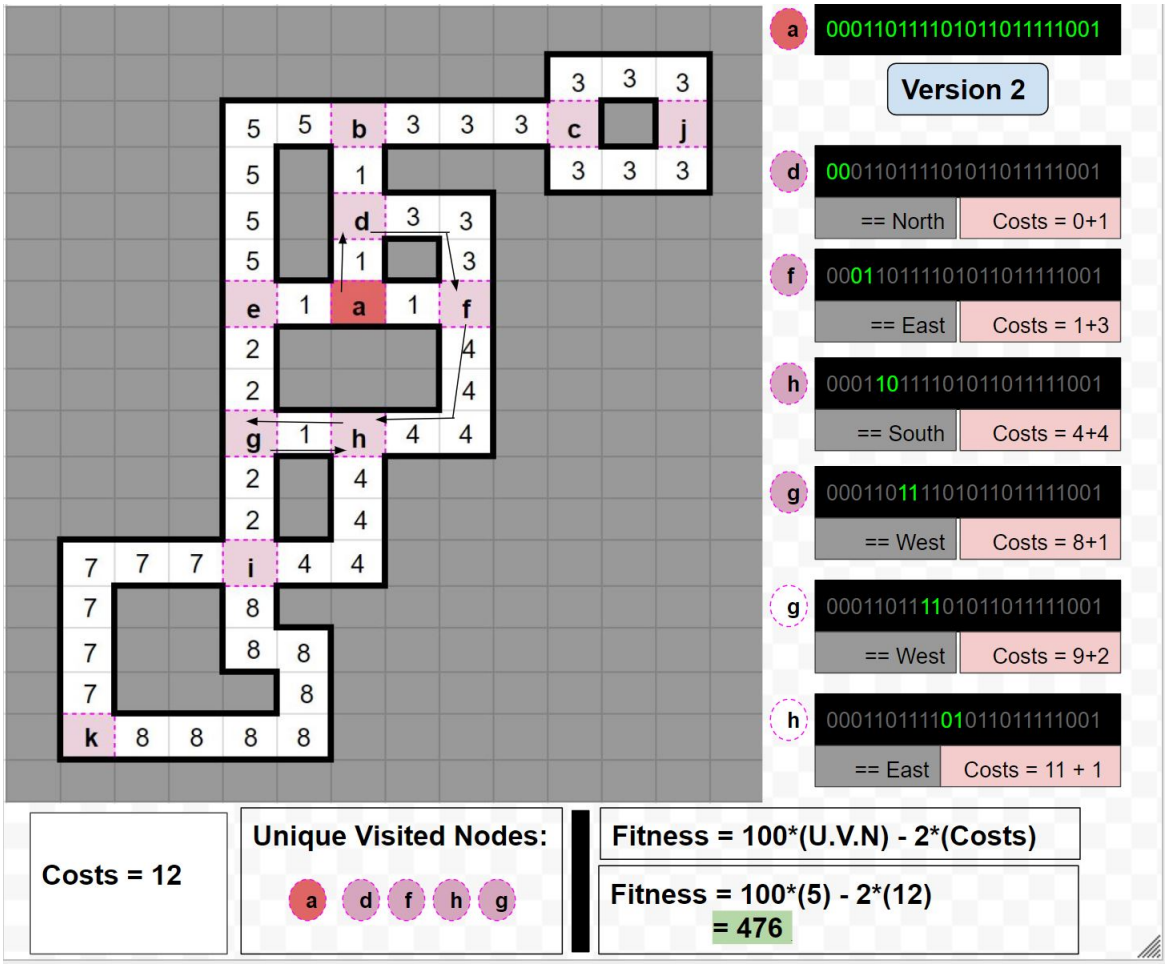
7.7 Fig4.2


Parent 1b
28
26
24
22
20
18
16
14
12
10
8
6
4
2
0
Parent 2b
29
27
25
23
21
19
17
15
13
11
9
7
5
3
1
P1b = 2*i
P2b = 2*i + 1
i = [0 : (N/2)-1]
N = 30

7.8 Fig4.3


Child 1
11
00
11
00
11
00
11
00
11
00
11
00
Parent 1
24 bits
Parent 2
11
11
11
11
11
11
11
11
11
11
11
11
00
00
00
00
00
00
00
00
00
00
00
00
00
11
00
11
00
11
00
11
00
11
00
11
00
11
Child 2

7.9 Fig5


a 000110111101011011111001
Version 2
d 000110111101011011111001
== North
Costs = 0+1
f 000110111101011011111001
== East
Costs = 1+3
h 000110111101011011111001
== South
Costs = 4+4
g 000110111101011011111001
== West
Costs = 8+1
g 000110111101011011111001
== West
Costs = 9+2
h 000110111101011011111001
== East
Costs = 11 + 1
Costs = 12
Unique Visited Nodes:
a d f h g
Fitness = 100*(U.V.N) - 2*(Costs)
Fitness = 100*(5) - 2*(12)
= 476