**מעבדה 2 חלק ב**

**סאמר חרעובה – 209050202**

**עבד אלרחמן אבו חוסין – 208517631**

Section 1:

* Firstly we created a problem set for Baldwins effect that is based on our classic class DNA:

1. Selects 1,0,? As described in the Baldwin experiment 0.25,0.25,0.5 respectively

Text

Description automatically generated

* Then we added 2 fitness functions :

1. fixed\_distance : calculated number of fixed correct/incorrect placements of bits
2. baldwinss: fitness function given in the lecture

Text

Description automatically generated

* created a memetic algorithm called PureMA:

Text

Description automatically generated

* baldwin(self,individual): replaces every ‘?’ with either 1 or 0
* correctness: function that calculates percentages of fixed correct/incorrect bits
* local\_search: the described local search given in the lecture

Text

Description automatically generated

Text

Description automatically generated

In local search , every individual is searched up to 1000 times , as

Section 1.a:

In terminal demo:



* In code we used the correctness function mentioned and explained above

A live demo of the code :

Text

Description automatically generated

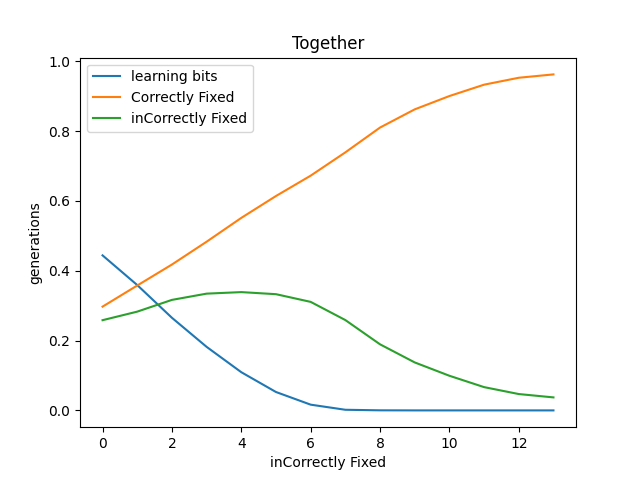
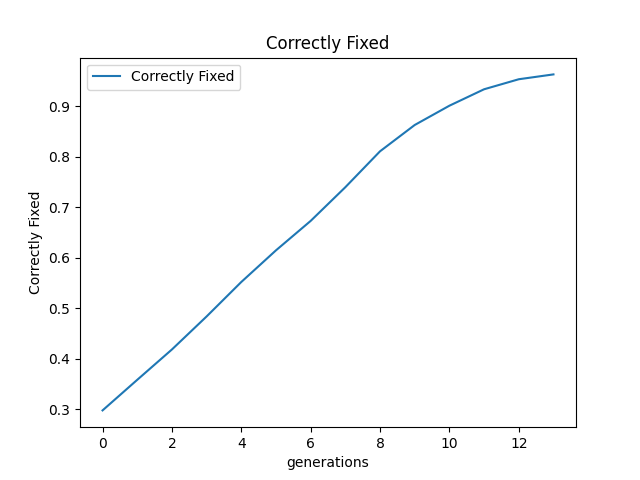
In the above simulation we used :

1. Swap mutation
2. RWS selection
3. 11101011110001010101 string made of 20 bits

As we can see , the results are amazing , in 6 generations the algorithm finds the best individuals , as we have made the algorithm work until fixed bits converge to 1 and about 0 .

,so the Baldwin effect actually does work wonders

Section 1.b:

Chart, line chart

Description automatically generatedChart, line chart

Description automatically generatedThe graphs of the above simulation :

Results of the graphs:

1. As we can see the correctly fixed bits increase with time up until they get to 100%
2. And the number of incorrectly fixed bits increases in a couple of generation , mainly due to the mutation type that we selected , and then rapidly decreases to zero
3. Learning bits decrease significantly , we can see that by the 6’th generation they almost came to be 0 and then we can also see that the algorithm has already found the solution by generation 6

Section 2 :

Added a generic implementation of MA with options for a learning algorithm and a fitness for learning :

Text

Description automatically generated

Section 2.a:

Added intensity and frequency ,so that we can use them in the upcoming learning algorithms:

Text

Description automatically generated

Section 2.b:

1. -> learning\_fitness
2. -> algo\_huristic

Text

Description automatically generated

Section 2.c:

1. Hill Climbing algorithm:
2. We used a function that creates all neighbors of each individual
3. Then sorted them according to the learning fitness and chose the best individual
4. We iterate over them according to the number of tries left for each individual (intensity)

Text

Description automatically generated

1. Random walk:
2. We get the probability of up or down movement from the user .
3. We then create a pattern of up and down indexes (walk)
4. Then we iterate over the selected individual’s neighbors until we meet our stoppage point

Text

Description automatically generated

Section 2.d:

We used the a heuristic that we used in section 1 as it best suits this problem

Section 2.e :

Implementing k-gene-exchanges was easy as we used the base engine of section 1 which allowed us to use previous functions with a small change , that being using k instead of another parameter .

Text

Description automatically generated

Section 2.f :

* **Completeness:** this algorithm always got to the solution just like most of our algorithms ,given the right parameters it is complete
* **Optimality :** it always get’s the solution and not a close approximation of the solution
* **Convergence speed :** as we have seen in section 1 the algorithm already gets the solution in less steps than the best solution from Lab 1 , as it takes about 4 to 6 generations to find the solution which is much better than 10 to 15 generations.
* **Speed of runtime :** as we have seen through all of this experiment ,we added a lot more calculations than in Lab1 algorithms ,which means of course that given the same parameters would yield the lab1 algorithms to be faster , yet if we lower the population size we can get a close fight between the two .