IMPLEMENTATION OF GENETIC ALGORITHM WITH SIMULATED ANNEALING FOR ARCHITECTURAL SEARCH

STUDENT ID: 20007748

Item Implementation

1. Implement a population-based search algorithm
   Initially I started off with a very basic genetic algorithm, with a simple replacement strategy, replacing every individual in the existing population with the new offspring and repeating the process until the number of generations are met. However, in attempts to increase the accuracy, I adopted a change in strategy for generating offspring. In the initial generation (generation 1), a population of size *2n* is randomly generated, where n is the total population size. Next, the best *n* chromosomes are selected based off their initial validation accuracy and they will constitute the new generation. This leads to an initial higher accuracy for a better initial search.

   For the remaining generations, I implemented an elitism population replacement strategy inspired by Hwang and He (2005). The best x% (in my case, 30%) of chromosomes evaluated by their max accuracy in the previous generation are selected for reproduction, and along with their offspring, subsequently make it into the next generation as candidate solutions. This ensures the next generation of candidate solutions have a good starting point to ensure at least an equal or better solution quality. (**Figure 1.0**) Crossover in my algorithm occurs by randomly selecting 2 parents from the mating pool, and by generating a random crossover point. An offspring is produced from slicing the first parent from index 0 to the crossover point, and slicing the second parent from the crossover point to the end of the chromosome, then merging both together to form a new chromosome .(**Figure 2.0**)

2. Incorporate local learning (memetic algorithm)
   - In local learning, I replaced the original hill climbing algorithm with simulated annealing (SA). The simulated annealing algorithm that I have implemented is referenced from the one provided in the slides Lecture 4. Using that as a basis to improve on, I integrated it with the genetic algorithm proposed above. The algorithm consists of SA followed by differential search, where non-worsening moves are always accepted, and worsening moves are accepted based off the Metropolis criterion. (**Figure 3.0**)

3. Include some variant of differential search
   - Regarding a variant of differential search, the original code only computes the difference between the neighbours generated from the chromosome in the annealing / hill climbing process (*resulting in new_res)* The change I implemented was to perform the differential search on *new_res*. This should provide results from different neighbourhoods generated from the solutions, therefore increasing the likelihood of finding a solution to break local optima or exceed the current best solution.

I have also included checks for stagnating accuracy, be it an accuracy above 70, or accuracies below 70. If the max accuracy is below 70 for a certain number of iterations, a **reheating** process (**Figure 4.0**) will take place in order to increase the temperature to have a higher probability of accepting worsening moves. The temperature in my algorithm is either increased or decreased geometrically with a factor of 0.8. Else, if the max accuracy stagnates above 70 for a certain number of iterations, and the reheating process has taken place, I assume that that chromosome has converged, and proceed onto the next chromosome.

Performance Evaluation

By controlling the number of evaluations for both pieces of coursework, the performance measure that we intend to compare will be in terms of accuracy of CW2 compared to CW1. I reran CW1 to be able to meet the controlled number of evaluations, taken into consideration as CW2 has considerably more evaluations compared to CW1. 5 independent trials were ran on both pieces of coursework, with 1 trial in CW2 consisting of 5 generations each. CW2 was not built off CW1, but instead from the original code that was provided prior to CW1.

The results across 5 trials for both pieces of coursework are as follows:
Mean accuracy across 5 trials - CW1: 75.6%, CW2: 74.58% (**Figure 5.0**)
Standard error across 5 trials - CW1: 0.245, CW2: 0.020
Standard deviation across 5 trials – CW1: 0.490, CW2: 0.041

The lower test accuracy achieved from CW2 could be attributed to the crossover operator implemented, where the crossover may not be optimal for the purpose of this assignment or the worsening moves accepted by simulated annealing have led to a lower overall accuracy. Nevertheless, the accuracies per generation remain very consistent with one another, with only a standard deviation of 0.04 across trials, and a standard error of 0.02. Noticeably, the first generation of my genetic algorithm always results in the highest accuracy across all generations that can be attributed to the selection process conducted on the initial population before the learning process.

By analysing the accuracies obtained over all 5 trials in **Figure 6.0**, we can see that the distribution is skewed to the left, where 75% of the accuracies are below 74.7%. The remaining top 25% of data are probably chromosomes which have successfully found better local optima or at best, the outlier in the top 25% might have resulted in a global optimum.

Next, by performing an unpaired two-samples Wilcoxon test on both samples, 5 trials of CW1, and CW2, I obtained a p-value of $0.04 < 0.05$, which does state that both samples are statistically significant from one another. This can be attributed to the different methods used to obtain the results above, where the results do not occur randomly but are tied into the methods used.

Lastly, I have been observing the models of the solutions produced in and noted that the best solutions found by my algorithm have models that usually consist of many identity layers. Perhaps the algorithm currently is still unable to find complex networks that potentially will provide a higher accuracy after sufficient training in favor of the identity layers which make up a simpler network.

Overall, despite the population based simulated annealing implemented in this coursework not achieving the level of accuracy as the point based search algorithm in CW1 with an equal number of evaluations, it resulted in very consistent accuracy values across generations, with a lower standard deviation and standard error values across trials compared to CW1. Perhaps the results could be further improved given enough resources and time for the algorithm to run. After all, genetic algorithms are a comparatively expensive process, however not necessarily optimal as convergence towards the optimal solution will be somewhat dependent on the diversity of the initial population and the methods of generating offspring. Without a sufficiently diverse initial population, the variation of solutions found may be limited to local optima, and the offspring produced might be limited in that sense as well.

REFERENCES

Hwang, S.F & He,R.S. (2005). Improving real-parameter genetic algorithm with simulated annealing for engineering problems. *Advances in Engineering Software, 37,* 409.

Van Hentenryck, P & Vergados, Y. (2007). Population-Based Simulated Annealing for Traveling Tournaments.. Proceedings of the National Conference on Artificial Intelligence. 1. 267-271.

APPENDIX

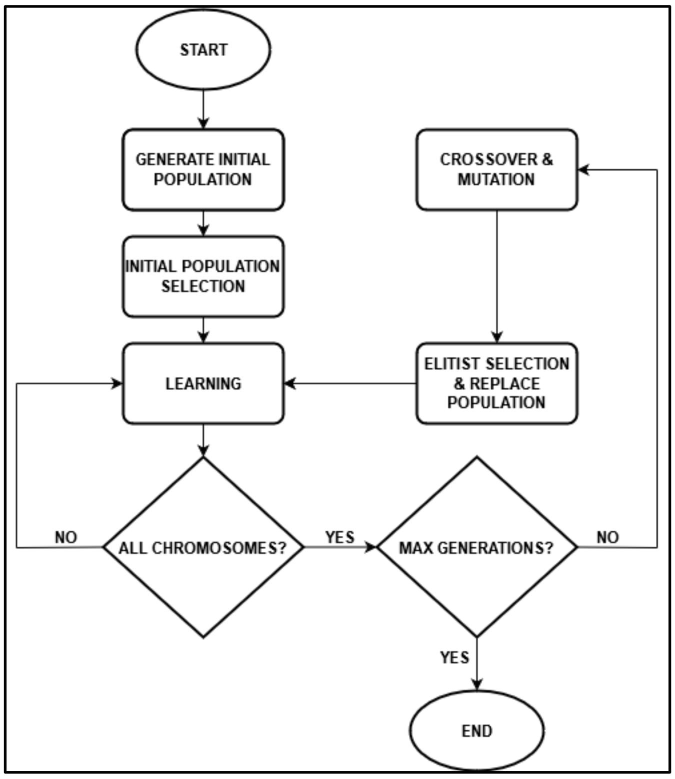

START
GENERATE INITIAL POPULATION
INITIAL POPULATION SELECTION
LEARNING
ALL CHROMOSOMES?
NO
YES
MAX GENERATIONS?
NO
YES
CROSSOVER & MUTATION
ELITIST SELECTION & REPLACE POPULATION
END

Figure 1.0: Flowchart of genetic algorithm process

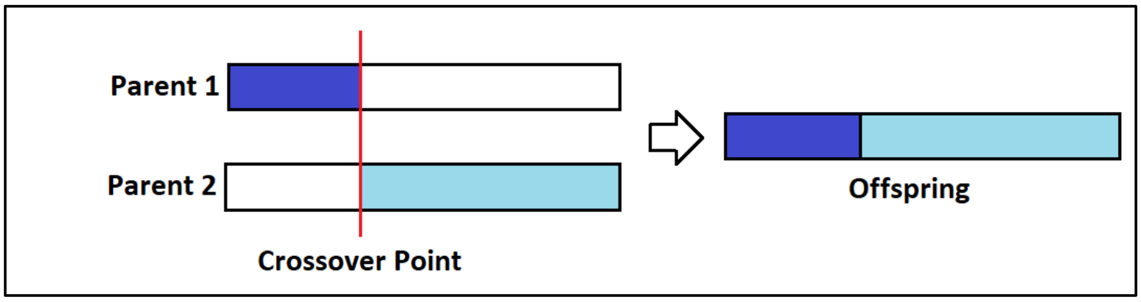
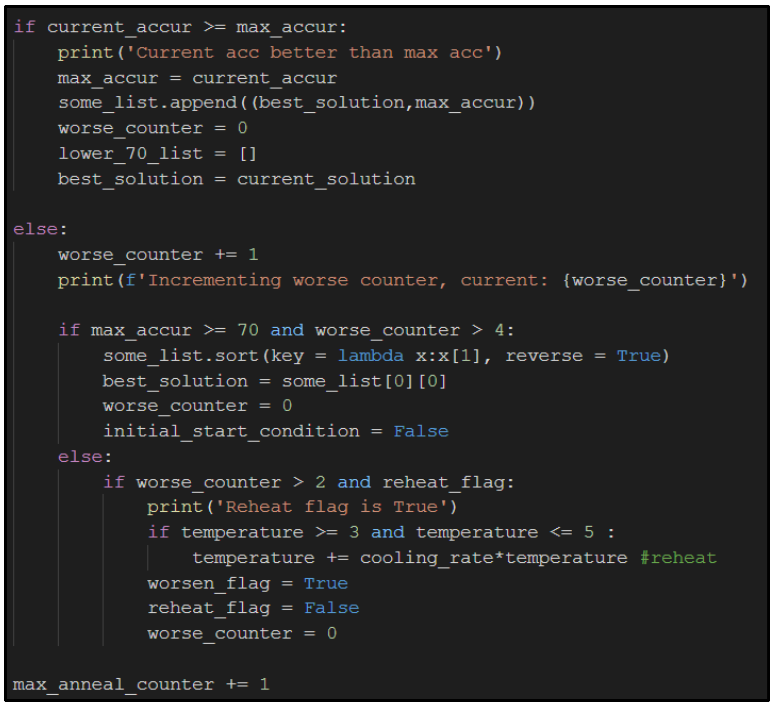


Figure 2.0: Crossover process

```
forall chromosome ⊆ population do
      Sbest <- chromosome
      Scurrent <- chromosome
      temperature <- initial_temperature
      while running is True and runs <= n do
            S' <- create_neighbours(Scurrent)
            Snew <- evaluate_neighbours(S')
            Δ(Snew, Scurrent) <- Snew - Scurrent¬
            if Snew > Scurrent
                  Scurrent+1 <- Snew
            else
                  random_num = random(0,1)
                  if random_num <  Δ(Snew, Scurrent)
                        Scurrent+¬1 <- Snew
            if Snew > Sbest
                  Scurrent = Snew
            temperature <- temperature * cooling_rate

            S'' <- fdiff_search(S')
            Snew' <- fdiff_search(Snew)
            if S'' > Snew'and S'' > Sbest
                  Sbest <- S''
            else
                  Sbest <- Snew'
            n <- n + 1
      endwhile
      if accuracy stagnates and accuracy > threshold
            running <- False
endfor
```

Figure 3.0: Pseudocode of simulated annealing implemented utilised



```python
if current_accur >= max_accur:
    print('Current acc better than max acc')
    max_accur = current_accur
    some_list.append((best_solution,max_accur))
    worse_counter = 0
    lower_70_list = []
    best_solution = current_solution

else:
    worse_counter += 1
    print(f'Incrementing worse counter, current: {worse_counter}')

    if max_accur >= 70 and worse_counter > 4:
        some_list.sort(key = lambda x:x[1], reverse = True)
        best_solution = some_list[0][0]
        worse_counter = 0
        initial_start_condition = False
    else:
        if worse_counter > 2 and reheat_flag:
            print('Reheat flag is True')
            if temperature >= 3 and temperature <= 5 :
                temperature += cooling_rate*temperature #reheat
            worsen_flag = True
            reheat_flag = False
            worse_counter = 0

max_anneal_counter += 1
```

Figure 4.0: Stagnating / non-improving accuracy check

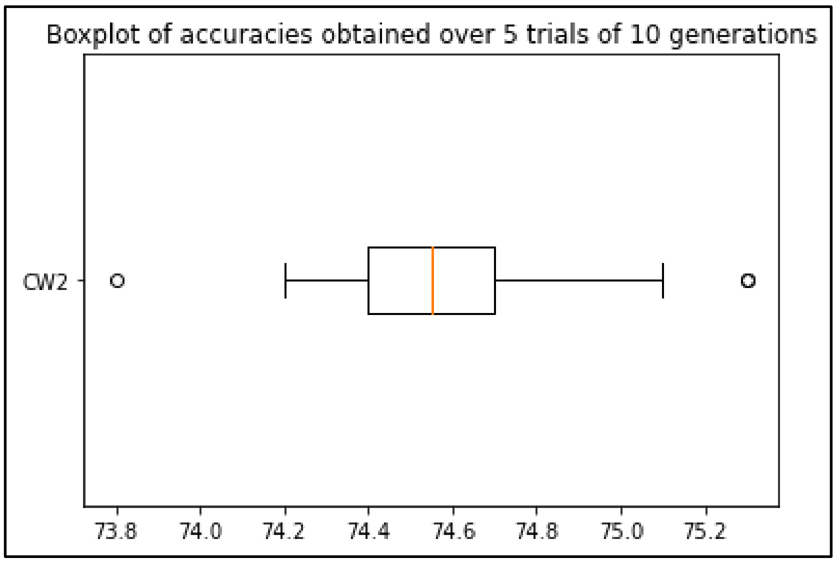
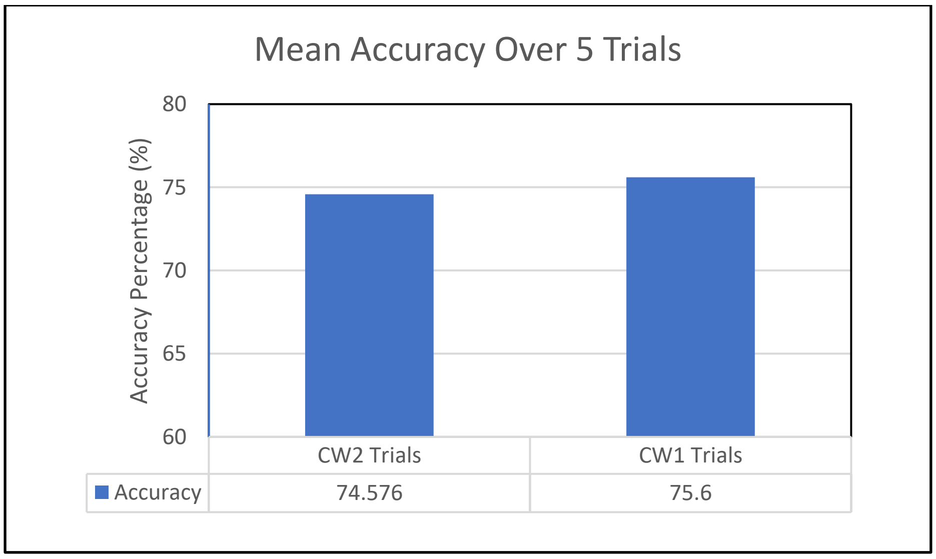


Figure 5.0: The range of values obtained over all trials in CW2(50 values)



	CW2 Trials	CW1 Trials
🟦 Accuracy	74.576	75.6

Figure 6.0: Bar graph of Mean Accuracy over 5 trials on both pieces of coursework