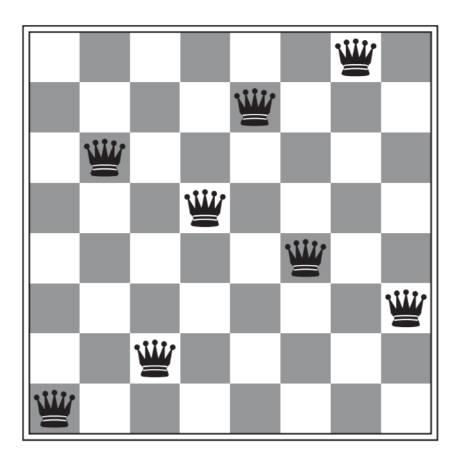
**Eight Queens – Genetic Algorithm**

*“One general law, leading to the advancement of all organic beings, namely, multiply, vary, let the strongest live and the weakest die.”****Charles Darwin, The Origin of Species (1859)***



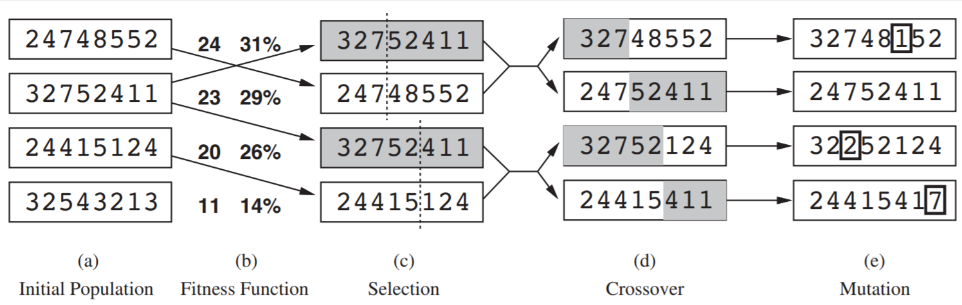
**Fig. 1**  from Russell and Norvig (2016) shows a completed eight queen problem, with no attacking queens

Darwin’s theory of natural selection **(Darwin, 1859)** was revolutionary in defining and understanding how the forces of nature guide and shape the evolution of species through generation to generation. The genetic algorithm, first proposed by **Holland (1975)** takes inspiration from this natural process, applying a similar pattern as seen among sexual organisms, producing offspring from parents of a population to seek an optimum solution that maximises fitness. Here I will apply the genetic algorithm to the Eight Queens Problem, first with a basic implementation and then with a more optimised solution.

The Eight Queen problem is a puzzle which involves the placing eight queens onto a board without any queens being attacked by another. A completed puzzle is shown in **fig. 1**.

**Basic** **Implementation**

The implemented basic genetic algorithm follows the pattern described in (**Russell and Norvig, 2016**).Here I will briefly describe each element, the full code can be seen in the accompanying jupyter notebook.



***Fig. 2*** *from Russell and Norvig (2016)**shows the genetic algorithm pattern applied to the eight queens problem*

The **genetic algorithm**  is made up of the following steps:

1. **Initial Population:** an initial population of genotypes or board states is created as an array of integers, the index of the integer indicates the column that the queen is on and the value indicates the row. Note I have included a zero value for the first row as opposed to a one shown in **fig 2**.
2. **Fitness Function:** a fitness function calculates the fitness score of each genotype
3. **Selection:** pairs of parents are selected using a roulette wheel selection mechanism; any genotype can be selected but the chance of being selected is proportional to their fitness score and each pair
4. **Crossover:** random crossover points are selected for the pairs of parents and they are crossed at this point
5. **Mutation:** each value in the genotype is subject to a chance of mutation; where it’s value can be changed to any other (except itself), the chance it will change is defined by the **mutation frequency**

The combinations of all these steps is a **generation**, it is repeated until a perfect state is achieved.

**Test**

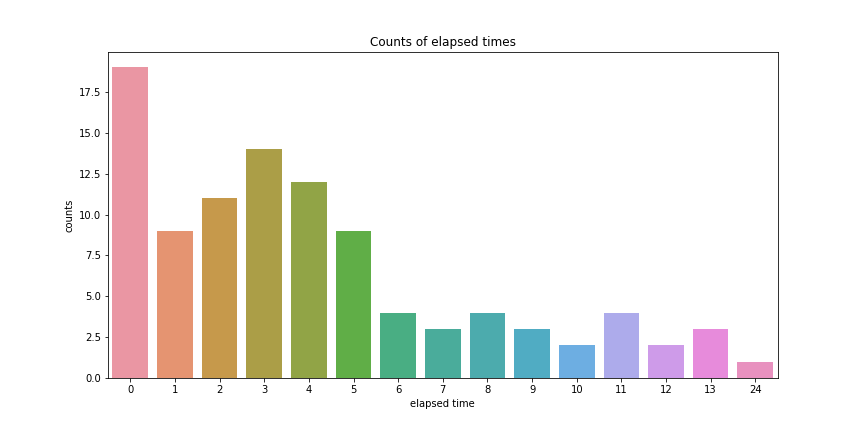
To fairly compare different implementations of the algorithm I have created multiple test data sets of a 100 randomly populations with varying population size (number of genotypes in the population). On the left of **fig. 3a** you can see the parameters chosen for the test, the size of the population being the number of genotypes in each, and the mutation frequency the chance that any value will be mutated. On the right of **fig. 3a** you can see the results of the basic genetic algorithm, performance is measured by the elapsed time and number of generations taken to find a complete solution. For **fig 3b**  the elapsed times are rounded down to whole integers (i.e. a time of 0.35s becomes 0) and **fig 3c** number of generations are rounded down to nearest 1000. They are and plotted as compared to the count.

**Mean no. of generations:** 4997.44 **Standard deviation no. of generations:** 4190.33  
**Range no. of generation:** 23667 **Mean elapsed time:** 4.685s **Standard deviation elapsed time:** 4.084s **Range elapsed time:** 24.395s

**Number of queens:**  8

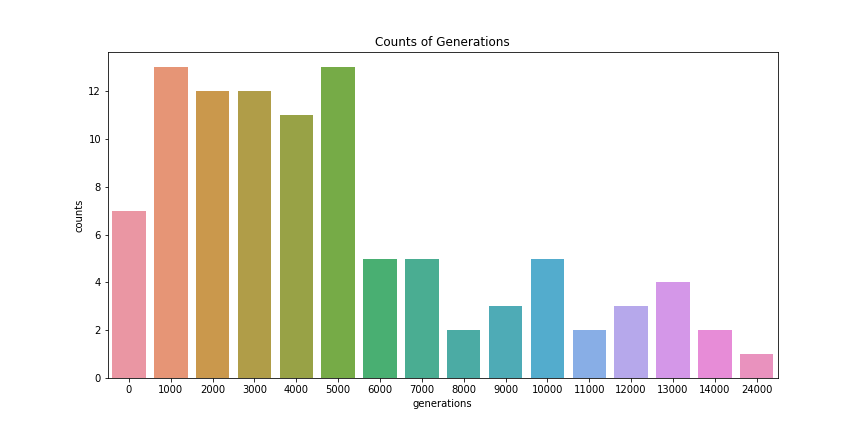
**Size of population:** 25

**Mutation frequency:** 1/8



***Fig. 3a*** *parameters (left) and results (right) of test of basic Genetic Algorithm implementation*

***Fig. 3b*** *count of elapsed time to completion, elapsed time rounded down to nearest whole integer*



***Fig. 3c*** *count of no. of generations to completion, rounded down to nearest 1000*

**Results**

For all the populations in the test data set the algorithm is able to find the completed solution, however the mean elapsed time is quite slow at 4.655 seconds and there is a high variance, with a range of 24.395s and a standard deviation of 4.084s. I was able to speed up the solution by increasing the mutation frequency, up to a point.

With limited improvements available through tuning of the parameters, I will next look at improvements to the algorithm itself.

**Optimised Implementation:**

The main issues with the basic implementation of the algorithm are a low mean speed and a high variance in the completion time, to tackle this I have implemented several extensions and adaptation.

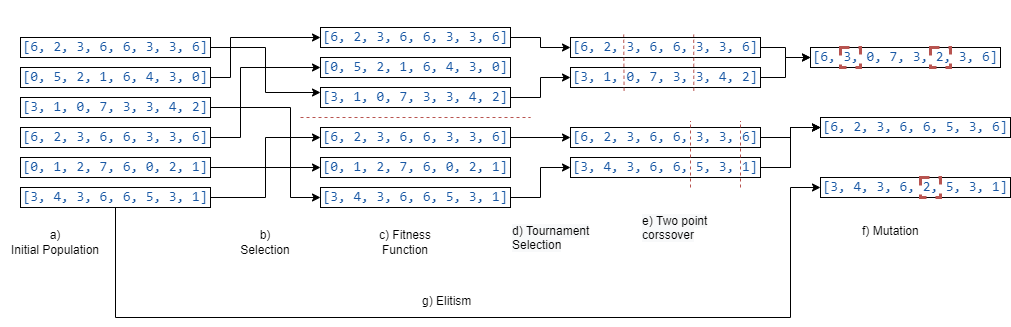
The low speed is due to the low likelihood of the algorithm to find a solution through crossover and mutation, this can be inferred by the effect of increasing the mutation rate in the basic algorithm, which up to a point increased the speed of the algorithm. Based on work by **Zhong et al. (2005)** tournament selection has been shown to be more performant than fitness proportional roulette wheel selection, I have modified my code to use an adaptation of the tournament selection method.

I have combined this new selection method with an updated crossover mechanism based on two points as compared to one in the basic algorithm. Double or two point mutation increases the number of potential combinations that the algorithm can produce given the mating of two parents and can greatly improve the speed of the algorithm **(Andrea et al., 2001).**

I have also implemented elitism which bypasses the selection and crossover process for a number of the original population, those with the highest fitness score, in each generation.

The high variance in the amount of time it takes for the algorithm to find the optimal solutions is due to premature convergence, overtime the algorithm will make the population more homogenous and if the algorithm is unable to easily find the solution, by crossover or mutation, from the now homogeneous population it may take many generations for random combinations to create a finished state. The use of tournament selection and two point crossover will allow for greater variance in the offspring and an increased chance of producing a solution. I have also increased the size of the population to 1000 this higher population means there is greater variance and it will maintain a more heterogenous state for more generations.

An overview of the optimised process can be seen in **fig 4**.



***Fig. 4*** *shows the optimised genetic algorithm pattern applied to the eight queens problem*

Below is an overview of the updated parts of the algorithm;

**b) Selection** the population is split into multiple tournament populations the number of genotypes in each is defined by the **tournament population size**

**d) Tournament Selection** the fitness function is utilised to calculate the fitness score and the two winners, those with the highest score, of each tournament population are selected as parents for the next generation. Note each tournament produces one offspring so the number of tournaments will be equal to the size of the population

**e) Two point cross over** two random points are selected and the parents are spliced, with the external values of one parent and the internal values of the other chosen

**g) Elitism** a number of values defined by the **elitism ratio** are passed to the final stage, these also undergo the mutation operation with a chance of mutating at any phenotype

**Test**

To test the optimised solution I used the randomly generated 100 member population data set with 1000 genotypes in each population.

**Mean no. of generations:** 153.31 **Standard deviation no. of generations:** 191.34  
**Range no. of generation:** 1084 **Mean elapsed time:** 0.366s **Standard deviation elapsed time:** 0.445s **Range elapsed time:** 2.527s

**Number of queens: 8**

**Size of population: 1000**

**Mutation frequency: 1/8**

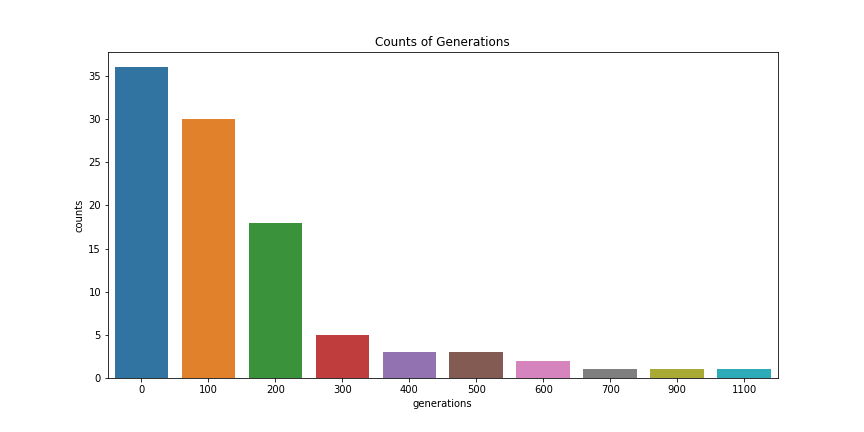
**Tournament Population size: 5**

**Elitism Ratio: 0.25**

***Fig. 5a*** *parameters (left) and results (right) of test of optimised Genetic Algorithm implementation*



***Fig. 3b*** *count of elapsed time to completion, elapsed time rounded down to nearest whole integer*



***Fig. 3b*** *count of no. of generations to completion, rounded down to nearest 1000*

**Results**

As can be seen from the results there is a marked improvement in the optimised version of the genetic algorithm as opposed to the basic implementation. The algorithm is able to find the solution with a mean time of 0.366s and a mean number of generations of 153.31. This shows an improvement in the speed of the process. What’s more the robustness of the algorithm is greatly improved with less variation in the amount of time and number of generation it takes to find the solution from different populations, this is shown by a reduced range of both time and generation at 2.527s and 1084 and a reduced standard deviation at 0.445s and 0.445s respectively.

**References:**

Andrea, J., Siarryb, P. and Dognona, T., 2001. An improvement of the standard genetic algorithm fighting premature convergence in continuous optimization. Advances in Engineering Software [Online], 32(1), pp.49–60. Available from: https://doi.org/10.1016/S0965-9978(00)00070-3 [Accessed 18 April 2021].

Darwin, C., 1859. The origin of species. London: Vintage.

Holland, J.H., 1975. Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control and artificial intelligence. Ann Arbor, Mich.

Russell, S.J. and Norvig, P., 2016. Artificial intelligence : a modern approach. Pearson Education, Limited.

Zhong, J., Hu, X., Gu, M. and Zhang, J., 2005. Comparison of Performance between Different Selection Strategies on Simple Genetic Algorithms. Proceedings of the International Conference on Computational Intelligence for Modelling Control and Automation / International Conference on Intelligent Agents, Web Technologies and Internet Commerce, CIMCA-IWATI, July 2005. pp.1115–1121.