Football Fixtures (EPL Fixtures) Using Genetic Algorithms

1. Introduction:

Genetic algorithms are a type of heuristic search algorithms that reflects the process of natural selection where the fittest individuals from the existing population are selected for reproduction to produce offspring of the next generation.

One of the applications of Genetic Algorithms is **scheduling** different events like University classes schedule, meetings schedule, fixtures schedule. We will study the use of GA for Football Fixtures, which is an NP complete problem and can be easily solved using Genetic Algorithm.

2. Problem Statement:

We implemented Genetic Algorithm for Football Fixtures problem. It is a scheduling problem which will create fixtures of the English Premier League (EPL) by taking Input as team names, match locations and the start date. The algorithm will create fixtures for the league. Each team will have their respective home ground. Standard league rules will apply.

Here,

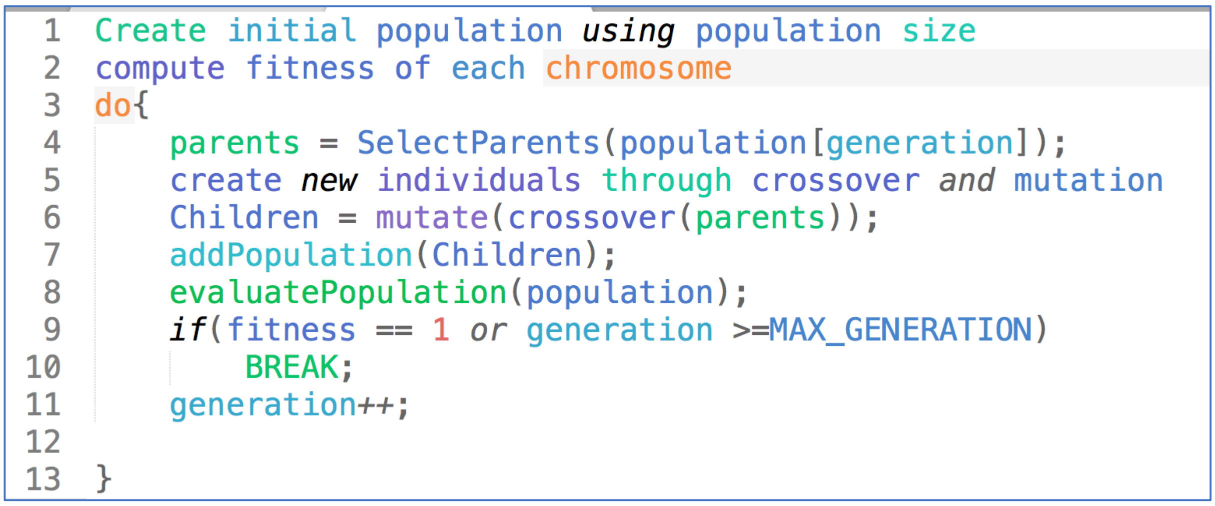
$$Totalnumberofmatches = \frac{(Numberofteams) * (Numberofteams - 1) * (Numberofrounds)}{2}$$

Following are the hard constrains used for creating the fixtures:

For example: Number of Rounds in the league: 2

1. A team cannot play with itself
2. Each team plays exactly 2(I.e. number of rounds) matches against each team in the league
3. Each team plays [total number of matches – number of rounds] matches in the league
4. Each team plays exactly one match at its own home ground and one match at opponent's home ground. (One – home, One - away)
5. Two matches cannot take place on the same day and same location
6. A team cannot play 2 matches on the same day

3. Basic Layout:



```
Create initial population using population size
compute fitness of each chromosome
do{
    parents = SelectParents(population[generation]);
    create new individuals through crossover and mutation
    Children = mutate(crossover(parents));
    addPopulation(Children);
    evaluatePopulation(population);
    if(fitness == 1 or generation >=MAX_GENERATION)
        BREAK;
    generation++;

}
```

Figure 3.1: Pseudo Code for Genetic Algorithm

Population, Chromosome and Gene:

In genetic algorithms, a chromosome (also sometimes called a genotype) is a set of parameters which define a proposed solution to the problem that the genetic algorithm is trying to solve. The set of all solutions is known as the *population*.

In the case of football fixtures, a chromosome is considered one possible solution to the league fixtures scheduler. **A chromosome** consists of **all the matches** played in the league.

A gene is one element position of a chromosome. In this case, a gene represents **a match** which is played between 2 team at a location on a date.

Encoding is useful for efficient manipulation of the representation of a chromosome. Object of type – Match was encoded in the chromosome for better accessing and manipulation. Further, a Team object was encapsulated in the Match gene to represent the playing teams. POJO class for Team was created to encapsulate the team name, home ground.

To sum it up, a population will contain multiple chromosomes where each chromosome is a possible solution to the fixture problem. The goal of a genetic algorithm is to solve an optimization problem.

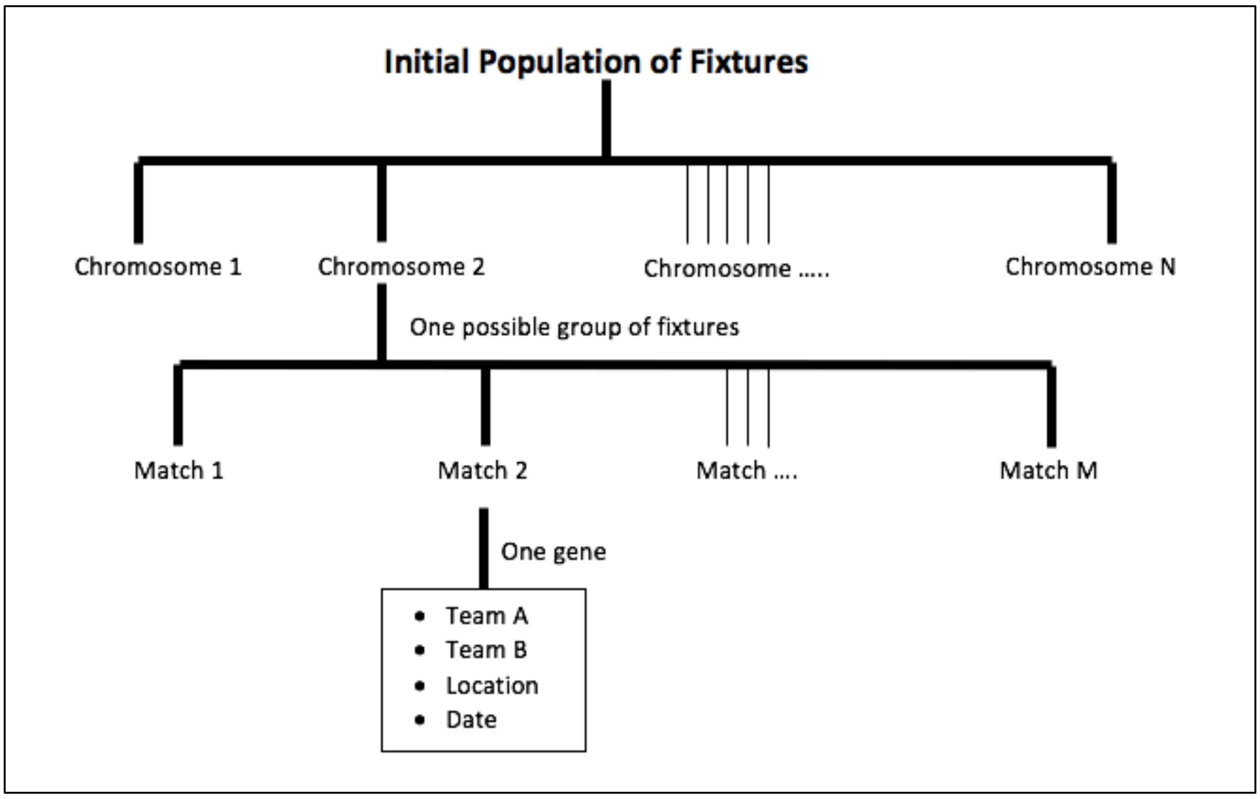


Figure 3.2: Basic layout of population, chromosome and gene in Football fixtures problem

4. Implementation Details:

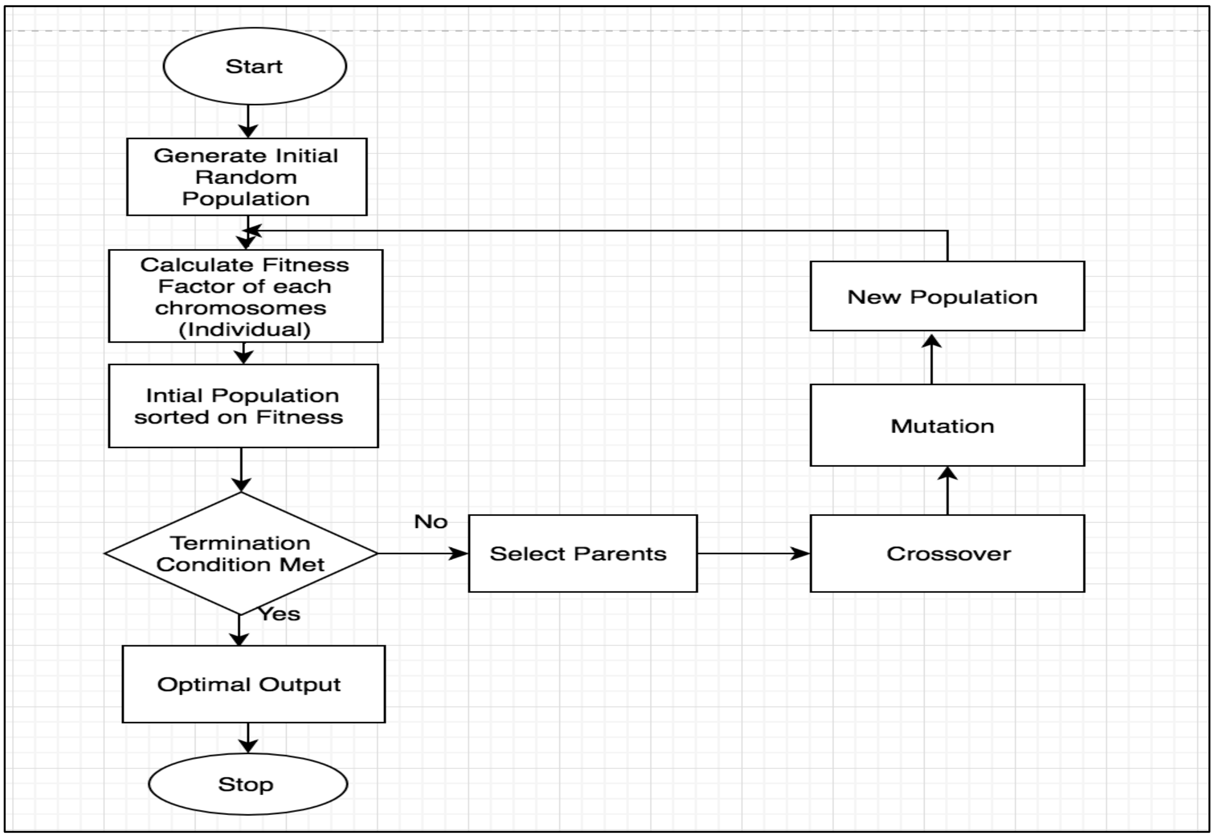

Start
Generate Initial Random Population
Calculate Fitness Factor of each chromosomes (Individual)
Intial Population sorted on Fitness
Termination Condition Met
No
Yes
Select Parents
Crossover
Mutation
New Population
Optimal Output
Stop

Figure 4.1: Flow chart of genetic algorithm implemented

We created separate packages for the helper classes and implementation classes for better code organization. The classes related to a Gene i.e. Match and Team are included in the helper package. The class related to the implementation of genetic algorithms are in the main java package.

A. Helper classes for Football fixtures implementation:

- Match: In this class, we have information about the two teams which are playing the match, the match date and the location. Getter and setter methods for each field are also included.

- Team: In this class, we have the team name and the home stadium associated with the team.

B. Main classes for Football fixtures implementation:

- FootballData: This class contains the data structures used for storing all the team names, location and dates. Getter and setter methods are also added for accessing the lists. We used array list data structure over arrays because of its ability to perform solo modifications on data and re-size ability.

- Configuration: This class is used to configure the data used for creating the football fixtures. This includes adding team names, locations and dates in its respective data structures.

- Chromosome: This class contains a complete schedule of matches in a league. The fitness of a chromosome is calculated based on how many hard constraints it follows.

  calculateFitness(): This method is used to determine the fitness factor on the basis of the number of hard constraints followed by the chromosome. We assign fitness score to every chromosome which is present in the population and see how close it to the desired solution. If the number of constraints followed is more, then there will be less number of conflicts in schedule. Hence the fitness of the chromosome will increase.

  The relationship between the fitness and number of conflicts is as follows:

  - Fitness of a chromosome = $\frac{1}{1 + \text{Conflicts}}$

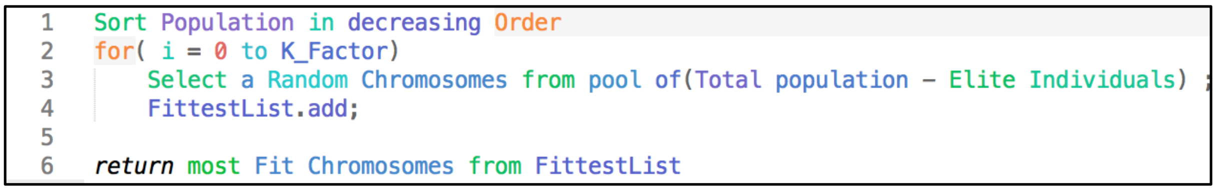
- Constants: This class is used to set the parameters used for running the genetic algorithm including the initial population size and other factors like cross over rate, mutation rate and k – selection rate. The following constants are used to obtain the fixtures:

  POPULATION_SIZE
  NUMBER_OF_ROUNDS
  ELITE_FACTOR
  K_FACTOR
  MUTATION_FACTOR
  CROSSOVER_RATE

MAX_GENERATION
MAX_COLONY_SIZE

- GeneticAlgorithm: This class is where the genetic operations are performed.

Parent Selection:
In nature the fittest chromosome, the one having highest fitness has high chances of mating and this causes their genes to contribute more in production of the next generation.
To evolve the generation, 2 chromosomes are selected for mutation. The selection of these parents can be done in different techniques. Roulette Wheel selection, K – tournament selection and Rank based selection are the techniques used to select the parents. In our problem, we have implemented the K – tournament selection technique.



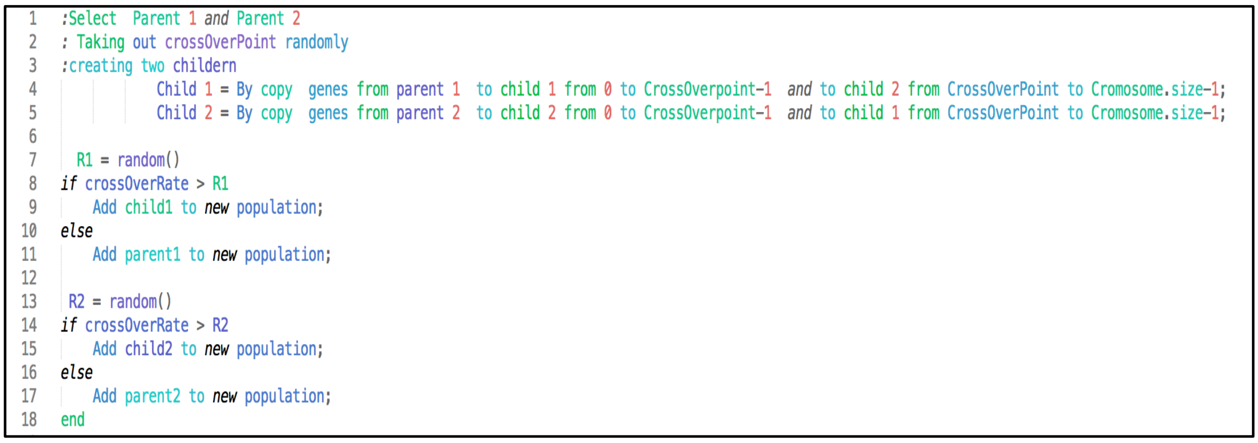
```
Sort Population in decreasing Order
for( i = 0 to K_Factor)
    Select a Random Chromosomes from pool of(Total population - Elite Individuals) ;
    FittestList.add;

return most Fit Chromosomes from FittestList
```

Figure 4.1.1: Pseudo code for K – tournament selection including elitism consideration

In **K - tournament selection**, we choose K chromosomes from the population randomly. Among the K chromosomes the best individual is selected on basis of fitness. Selection pressure can be adjusted by changing the tournament size. One advantage of this selection procedure is that it works with negative fitness value.

Elitism
Genetic Algorithm often loses its best chromosomes over generation because of mating and mutation. To avoid this, we use Elitism to preserve the best genes of the population.
This operator saves the small portion of best individual and call them elite and these elite individuals are copied to next generation without being unaltered. This way the fittest individuals are no longer lost between generations.
Number of individual to keep depend on the Elite Factor. This factor decides the percentage of population that should be kept and transferred unchanged to the next generation.

Cross – Over (Mating): Cross – over involved merging 2 parents and producing children which are a combination of both the parents. We are using single point crossover, in which crossover point is computed randomly to increase diversity of genes in future generations.



```
:Select  Parent 1 and Parent 2
: Taking out crossOverPoint randomly
:creating two childern
            Child 1 = By copy  genes from parent 1  to child 1 from 0 to CrossOverpoint-1  and to child 2 from CrossOverPoint to Cromosome.size-1;
            Child 2 = By copy  genes from parent 2  to child 2 from 0 to CrossOverpoint-1  and to child 1 from CrossOverPoint to Cromosome.size-1;

  R1 = random()
if crossOverRate > R1
    Add child1 to new population;
else
    Add parent1 to new population;

 R2 = random()
if crossOverRate > R2
    Add child2 to new population;
else
    Add parent2 to new population;
end
```

Figure 4.1.2: Pseudo code for Cross – Over

Mutation: To create a more diverse population, chromosomes undergoes mutation. In mutation, some of genes face random changes after recombination. These changes are completely random and can be good or bad for individual fitness.

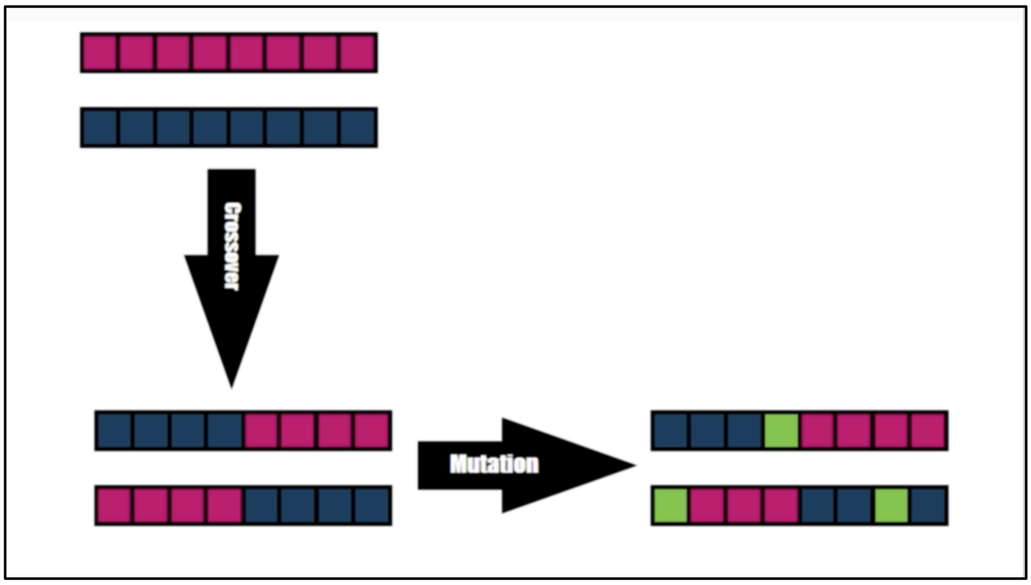
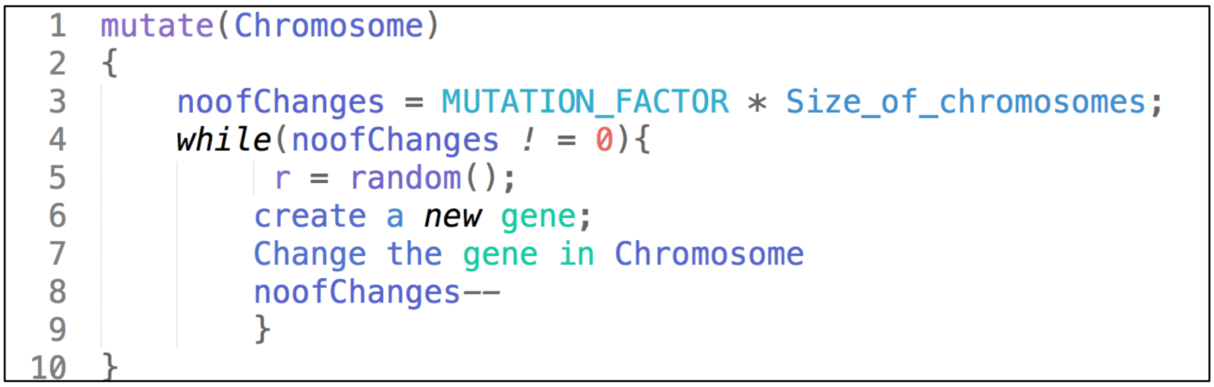


Figure 4.1.3: Cross Over and Mutation Workflow representation



```
mutate(Chromosome)
{
    noofChanges = MUTATION_FACTOR * Size_of_chromosomes;
    while(noofChanges ! = 0){
        r = random();
        create a new gene;
        Change the gene in Chromosome
        noofChanges--
        }
}
```

Figure 4.1.4: Pseudo Code for Mutation

- <u>FixturesMain:</u> This is our main class which interacts with other classes to make our genetic algorithm program work. It calls configuration class to initialize the data, configuration class assigns teams, match locations and dates to a match. It calls the population class to create initial population and then calls genetic algorithm class to run the algorithm on the population.

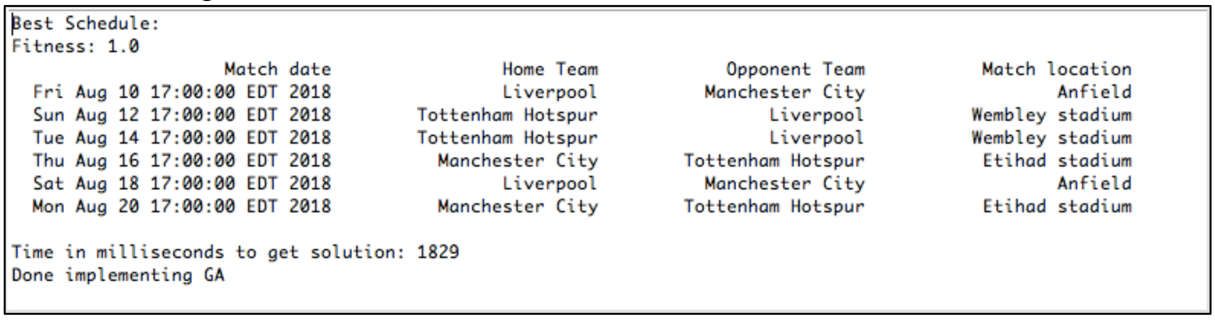
  Termination Condition:
  We have used two termination condition for terminating our program
  1) The program get terminated if fitness factor of any individual in the population becomes 1.
  2) When maximum generation equals to MAX_GENERATION constant value, the algorithm is terminated. If perfect schedule is not found, then the chromosome (match fixtures) with the maximum fitness is displayed.

5. Program Output:

The fixtures are displayed on the console output. If the perfect fixtures are not found, then the set of fixtures having the maximum fitness are displayed.

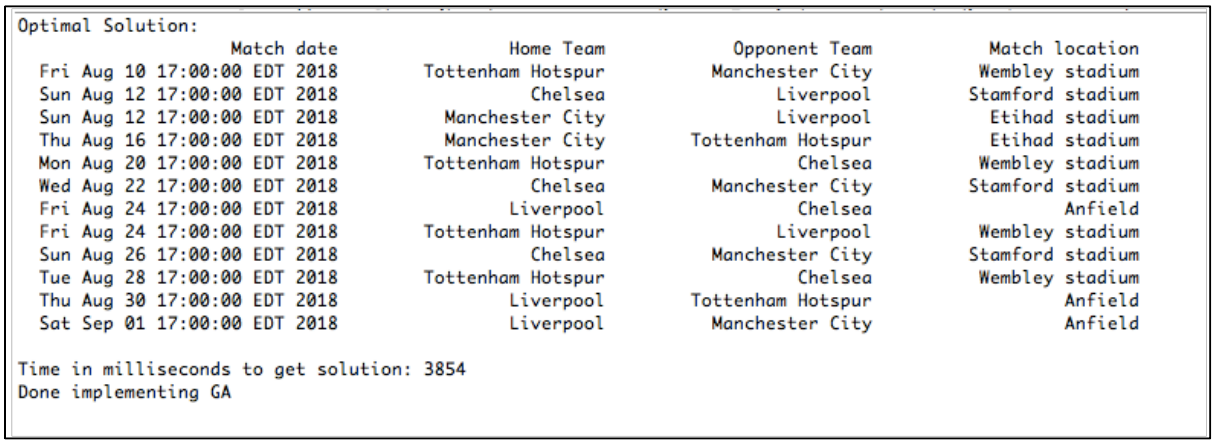
The following are outputs when the Football Fixtures was run for varying size of teams.

3 teams in a league:



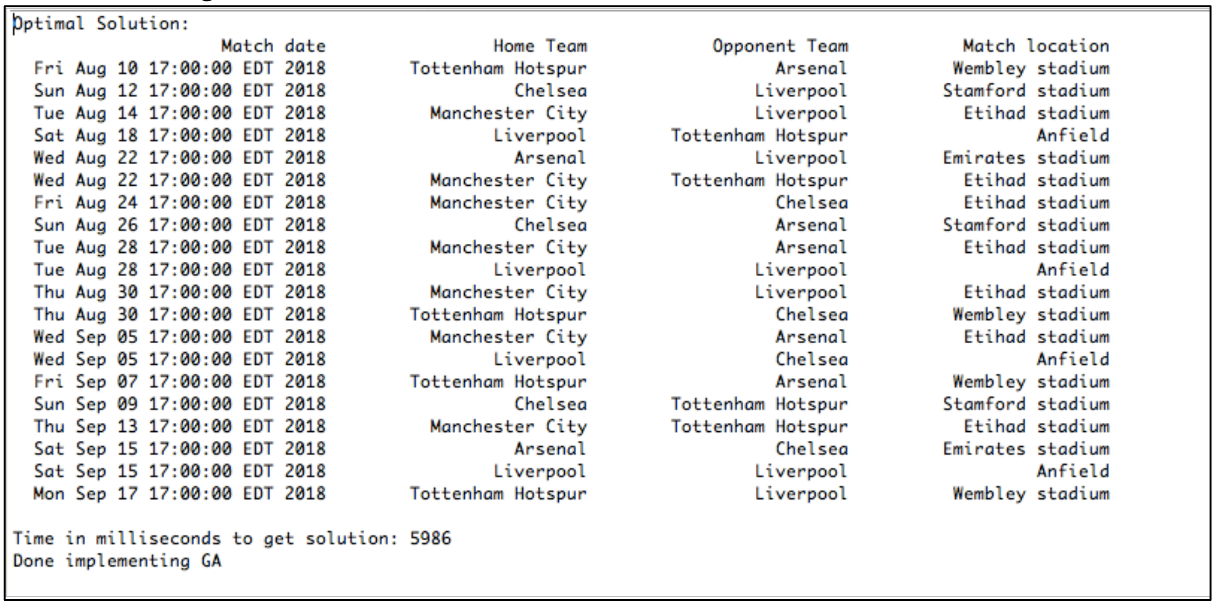
```
Best Schedule:
Fitness: 1.0
                        Match date                   Home Team               Opponent Team              Match location
  Fri Aug 10 17:00:00 EDT 2018                   Liverpool             Manchester City                   Anfield
  Sun Aug 12 17:00:00 EDT 2018           Tottenham Hotspur                   Liverpool           Wembley stadium
  Tue Aug 14 17:00:00 EDT 2018           Tottenham Hotspur                   Liverpool           Wembley stadium
  Thu Aug 16 17:00:00 EDT 2018             Manchester City           Tottenham Hotspur            Etihad stadium
  Sat Aug 18 17:00:00 EDT 2018                   Liverpool             Manchester City                   Anfield
  Mon Aug 20 17:00:00 EDT 2018             Manchester City           Tottenham Hotspur            Etihad stadium

Time in milliseconds to get solution: 1829
Done implementing GA
```

4 teams in a league:



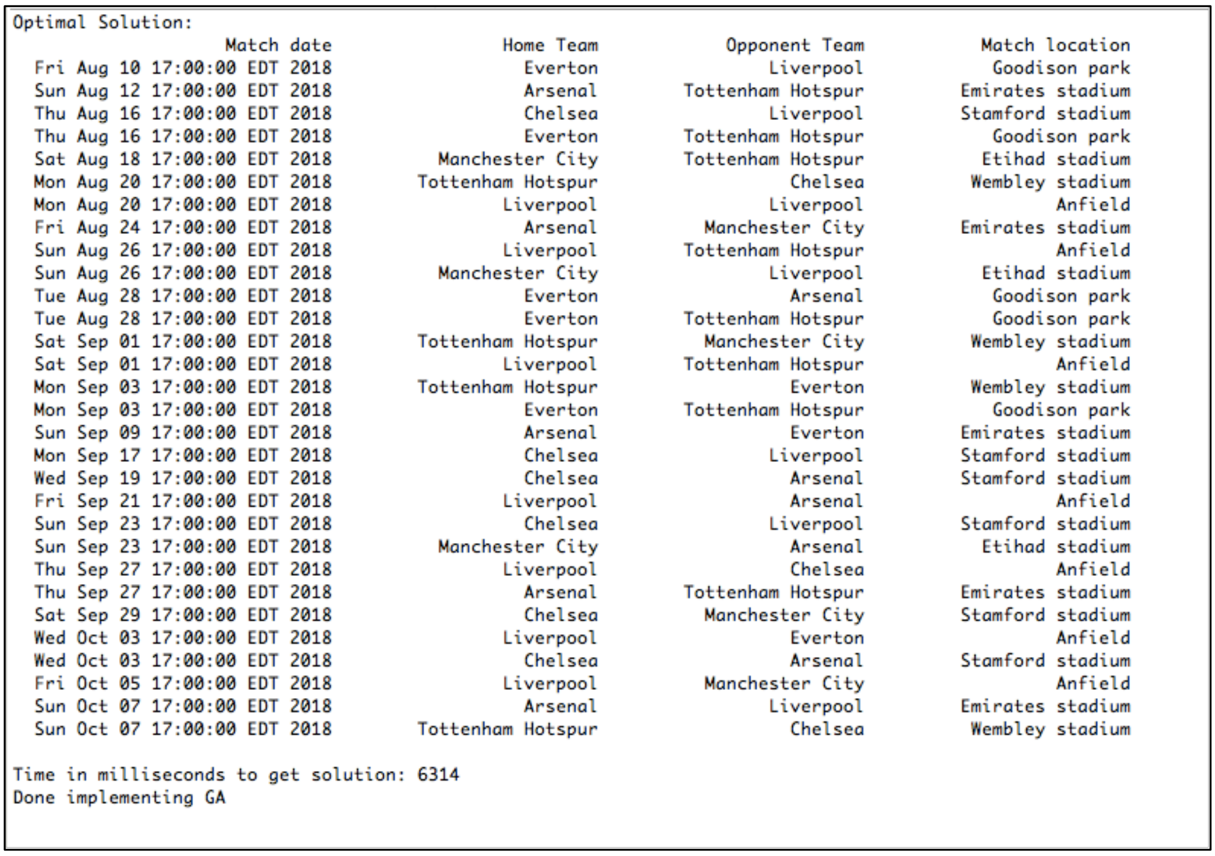
```
Optimal Solution:
                   Match date             Home Team          Opponent Team         Match location
  Fri Aug 10 17:00:00 EDT 2018    Tottenham Hotspur      Manchester City         Wembley stadium
  Sun Aug 12 17:00:00 EDT 2018              Chelsea            Liverpool        Stamford stadium
  Sun Aug 12 17:00:00 EDT 2018      Manchester City            Liverpool          Etihad stadium
  Thu Aug 16 17:00:00 EDT 2018      Manchester City    Tottenham Hotspur          Etihad stadium
  Mon Aug 20 17:00:00 EDT 2018    Tottenham Hotspur              Chelsea         Wembley stadium
  Wed Aug 22 17:00:00 EDT 2018              Chelsea      Manchester City        Stamford stadium
  Fri Aug 24 17:00:00 EDT 2018            Liverpool              Chelsea                 Anfield
  Fri Aug 24 17:00:00 EDT 2018    Tottenham Hotspur            Liverpool         Wembley stadium
  Sun Aug 26 17:00:00 EDT 2018              Chelsea      Manchester City        Stamford stadium
  Tue Aug 28 17:00:00 EDT 2018    Tottenham Hotspur              Chelsea         Wembley stadium
  Thu Aug 30 17:00:00 EDT 2018            Liverpool    Tottenham Hotspur                 Anfield
  Sat Sep 01 17:00:00 EDT 2018            Liverpool      Manchester City                 Anfield

Time in milliseconds to get solution: 3854
Done implementing GA
```

5 teams in a league:



```
Optimal Solution:
                   Match date             Home Team          Opponent Team         Match location
  Fri Aug 10 17:00:00 EDT 2018    Tottenham Hotspur              Arsenal         Wembley stadium
  Sun Aug 12 17:00:00 EDT 2018              Chelsea            Liverpool        Stamford stadium
  Tue Aug 14 17:00:00 EDT 2018      Manchester City            Liverpool          Etihad stadium
  Sat Aug 18 17:00:00 EDT 2018            Liverpool    Tottenham Hotspur                 Anfield
  Wed Aug 22 17:00:00 EDT 2018              Arsenal            Liverpool        Emirates stadium
  Wed Aug 22 17:00:00 EDT 2018      Manchester City    Tottenham Hotspur          Etihad stadium
  Fri Aug 24 17:00:00 EDT 2018      Manchester City              Chelsea          Etihad stadium
  Sun Aug 26 17:00:00 EDT 2018              Chelsea              Arsenal        Stamford stadium
  Tue Aug 28 17:00:00 EDT 2018      Manchester City              Arsenal          Etihad stadium
  Tue Aug 28 17:00:00 EDT 2018            Liverpool            Liverpool                 Anfield
  Thu Aug 30 17:00:00 EDT 2018      Manchester City            Liverpool          Etihad stadium
  Thu Aug 30 17:00:00 EDT 2018    Tottenham Hotspur              Chelsea         Wembley stadium
  Wed Sep 05 17:00:00 EDT 2018      Manchester City              Arsenal          Etihad stadium
  Wed Sep 05 17:00:00 EDT 2018            Liverpool              Chelsea                 Anfield
  Fri Sep 07 17:00:00 EDT 2018    Tottenham Hotspur              Arsenal         Wembley stadium
  Sun Sep 09 17:00:00 EDT 2018              Chelsea    Tottenham Hotspur        Stamford stadium
  Thu Sep 13 17:00:00 EDT 2018      Manchester City    Tottenham Hotspur          Etihad stadium
  Sat Sep 15 17:00:00 EDT 2018              Arsenal              Chelsea        Emirates stadium
  Sat Sep 15 17:00:00 EDT 2018            Liverpool            Liverpool                 Anfield
  Mon Sep 17 17:00:00 EDT 2018    Tottenham Hotspur            Liverpool         Wembley stadium

Time in milliseconds to get solution: 5986
Done implementing GA
```

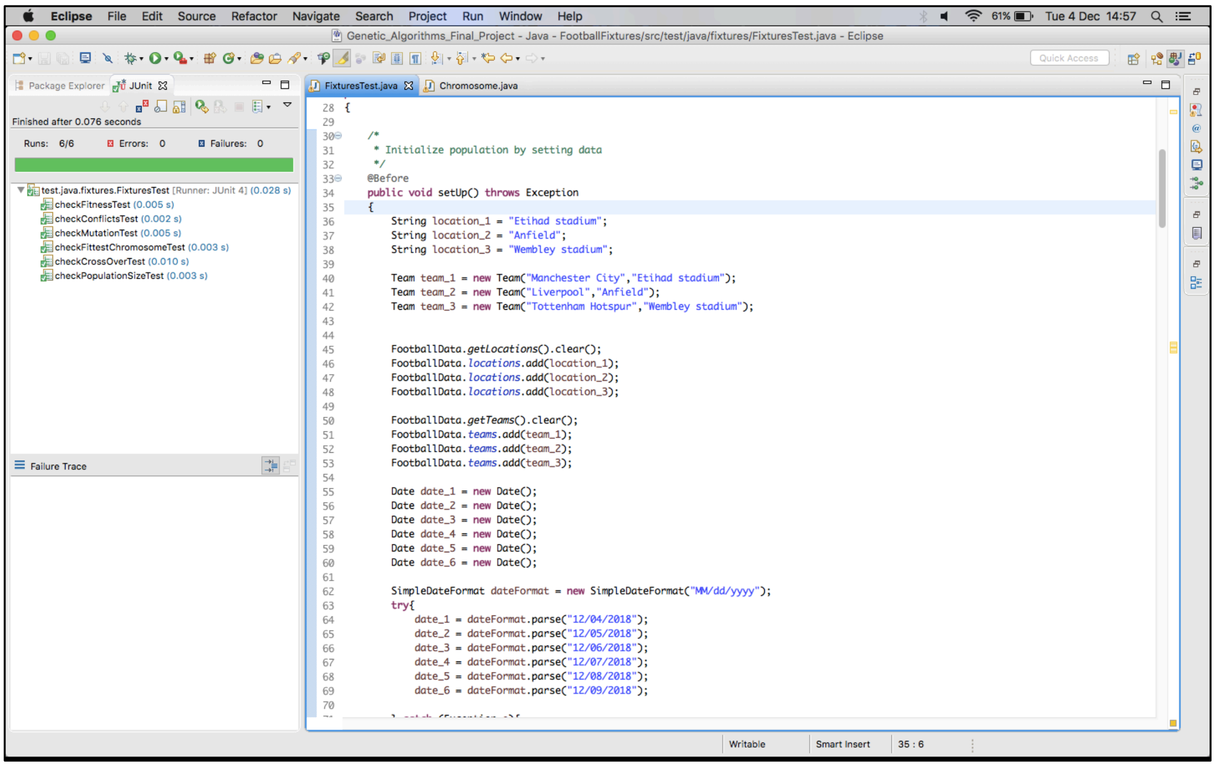
6 teams in a league:



```
Optimal Solution:
                     Match date                Home Team          Opponent Team          Match location
  Fri Aug 10 17:00:00 EDT 2018                   Everton               Liverpool           Goodison park
  Sun Aug 12 17:00:00 EDT 2018                   Arsenal       Tottenham Hotspur        Emirates stadium
  Thu Aug 16 17:00:00 EDT 2018                   Chelsea               Liverpool        Stamford stadium
  Thu Aug 16 17:00:00 EDT 2018                   Everton       Tottenham Hotspur           Goodison park
  Sat Aug 18 17:00:00 EDT 2018           Manchester City       Tottenham Hotspur          Etihad stadium
  Mon Aug 20 17:00:00 EDT 2018         Tottenham Hotspur                 Chelsea         Wembley stadium
  Mon Aug 20 17:00:00 EDT 2018                 Liverpool               Liverpool                 Anfield
  Fri Aug 24 17:00:00 EDT 2018                   Arsenal         Manchester City        Emirates stadium
  Sun Aug 26 17:00:00 EDT 2018                 Liverpool       Tottenham Hotspur                 Anfield
  Sun Aug 26 17:00:00 EDT 2018           Manchester City               Liverpool          Etihad stadium
  Tue Aug 28 17:00:00 EDT 2018                   Everton                 Arsenal           Goodison park
  Tue Aug 28 17:00:00 EDT 2018                   Everton       Tottenham Hotspur           Goodison park
  Sat Sep 01 17:00:00 EDT 2018         Tottenham Hotspur         Manchester City         Wembley stadium
  Sat Sep 01 17:00:00 EDT 2018                 Liverpool       Tottenham Hotspur                 Anfield
  Mon Sep 03 17:00:00 EDT 2018         Tottenham Hotspur                 Everton         Wembley stadium
  Mon Sep 03 17:00:00 EDT 2018                   Everton       Tottenham Hotspur           Goodison park
  Sun Sep 09 17:00:00 EDT 2018                   Arsenal                 Everton        Emirates stadium
  Mon Sep 17 17:00:00 EDT 2018                   Chelsea               Liverpool        Stamford stadium
  Wed Sep 19 17:00:00 EDT 2018                   Chelsea                 Arsenal        Stamford stadium
  Fri Sep 21 17:00:00 EDT 2018                 Liverpool                 Arsenal                 Anfield
  Sun Sep 23 17:00:00 EDT 2018                   Chelsea               Liverpool        Stamford stadium
  Sun Sep 23 17:00:00 EDT 2018           Manchester City                 Arsenal          Etihad stadium
  Thu Sep 27 17:00:00 EDT 2018                 Liverpool                 Chelsea                 Anfield
  Thu Sep 27 17:00:00 EDT 2018                   Arsenal       Tottenham Hotspur        Emirates stadium
  Sat Sep 29 17:00:00 EDT 2018                   Chelsea         Manchester City        Stamford stadium
  Wed Oct 03 17:00:00 EDT 2018                 Liverpool                 Everton                 Anfield
  Wed Oct 03 17:00:00 EDT 2018                   Chelsea                 Arsenal        Stamford stadium
  Fri Oct 05 17:00:00 EDT 2018                 Liverpool         Manchester City                 Anfield
  Sun Oct 07 17:00:00 EDT 2018                   Arsenal               Liverpool        Emirates stadium
  Sun Oct 07 17:00:00 EDT 2018         Tottenham Hotspur                 Chelsea         Wembley stadium

Time in milliseconds to get solution: 6314
Done implementing GA
```

The perfect schedule was found for the following parameters of genetic algorithms;
Initial Population Size: 100000
Maximum Number of Generation: 1000
Cross Over factor: 0.8
Mutation Factor: 0.8
K Factor: 100
Elite Factor: 0.04 * Population Size

J-Unit Test Cases: Tests were run for testing all the modular functions of the projects. Fitness function, cross over, mutation and sorting chromosome were tested. The code passed all the tests.



Genetic_Algorithms_Final_Project - Java - FootballFixtures/src/test/java/fixtures/FixturesTest.java - Eclipse

```
Finished after 0.076 seconds

Runs:  6/6        Errors:  0        Failures:  0

test.java.fixtures.FixturesTest [Runner: JUnit 4] (0.028 s)
    checkFitnessTest (0.005 s)
    checkConflictsTest (0.002 s)
    checkMutationTest (0.005 s)
    checkFittestChromosomeTest (0.003 s)
    checkCrossOverTest (0.010 s)
    checkPopulationSizeTest (0.003 s)
```

```java
{

    /*
     * Initialize population by setting data
     */
    @Before
    public void setUp() throws Exception
    {
        String location_1 = "Etihad stadium";
        String location_2 = "Anfield";
        String location_3 = "Wembley stadium";

        Team team_1 = new Team("Manchester City","Etihad stadium");
        Team team_2 = new Team("Liverpool","Anfield");
        Team team_3 = new Team("Tottenham Hotspur","Wembley stadium");


        FootballData.getLocations().clear();
        FootballData.locations.add(location_1);
        FootballData.locations.add(location_2);
        FootballData.locations.add(location_3);

        FootballData.getTeams().clear();
        FootballData.teams.add(team_1);
        FootballData.teams.add(team_2);
        FootballData.teams.add(team_3);

        Date date_1 = new Date();
        Date date_2 = new Date();
        Date date_3 = new Date();
        Date date_4 = new Date();
        Date date_5 = new Date();
        Date date_6 = new Date();

        SimpleDateFormat dateFormat = new SimpleDateFormat("MM/dd/yyyy");
        try{
            date_1 = dateFormat.parse("12/04/2018");
            date_2 = dateFormat.parse("12/05/2018");
            date_3 = dateFormat.parse("12/06/2018");
            date_4 = dateFormat.parse("12/07/2018");
            date_5 = dateFormat.parse("12/08/2018");
            date_6 = dateFormat.parse("12/09/2018");

```

Figure 5.1: Screenshot of J Unit test cases passing

6. Code Optimizations:

Benchmarking was done to check the efficiency of the program. The following code optimization were done to increase the efficiency of the program:

- **<u>Parallel processing</u>**: Parallel processing was implemented while initializing the population. As the initial population was high, creation of such a large amount of data was taking substantial time. To reduce that, the initial population was split into small portions to generate random chromosomes. The **CompletableFuture functionality of Java 8** was used to implement this function.

  Note: After benchmarking the program, it was observed that time reduced by half after implementing parallel processing.

- **<u>Use of Symbol Table:</u>** For **efficient search operations** and **improved indexing**, symbol tables were used instead of other data structures. **Hash – maps** were used to store the various matches and its location to determine the fitness of a chromosome.

- **<u>Use of Knuth Shuffle:</u>** Knuth shuffle was before sorting the population to avoid any element of bias/pattern in the collection to be sorted

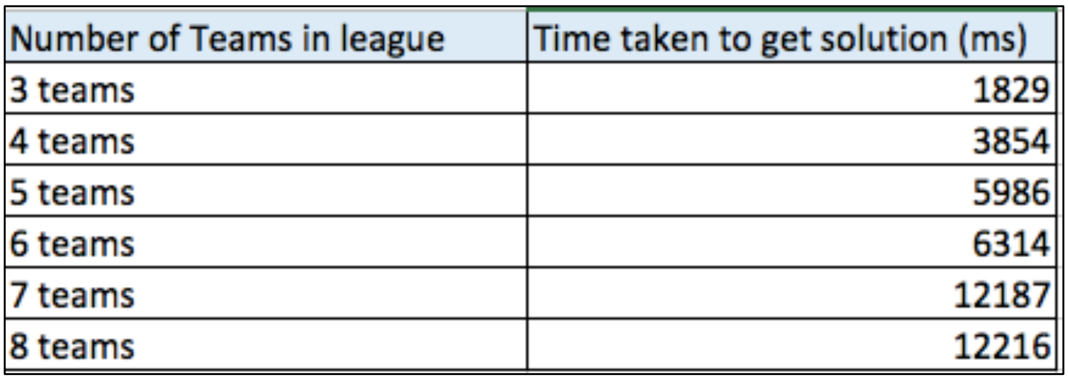
- **<u>Use of Comparable Interface:</u>** The comparable interface was implemented in the Chromosome class to **compare 2 chromosomes based on its fitness**. This was helpful in determining the fittest chromosome which creating a new generation.

- **<u>Use of Hash-Code and equals method:</u>** These methods were implemented in the Match class to compare 2 matches. All the attributes of the match were considered for generation of both the methods.

7. Observations and Conclusions:

The following observations were made on the running of the genetic algorithm for drawing football fixtures:

- As the input complexity increased, the probability of getting the perfect solution decreased. This is evident from the screenshots posted in the Program Output section.
- The number of conflicts increased as the number of teams increased. This led to increase in the number of generations taken to reach the optimal solution.
- As the input complexity increased, the time complexity of the program also increased.



Number of Teams in league	Time taken to get solution (ms)
3 teams	1829
4 teams	3854
5 teams	5986
6 teams	6314
7 teams	12187
8 teams	12216

Figure 7.1: Readings of number of teams VS execution time to get solution

- The fitness factor increased as the number of generations increased till it reached its peak of 1.0

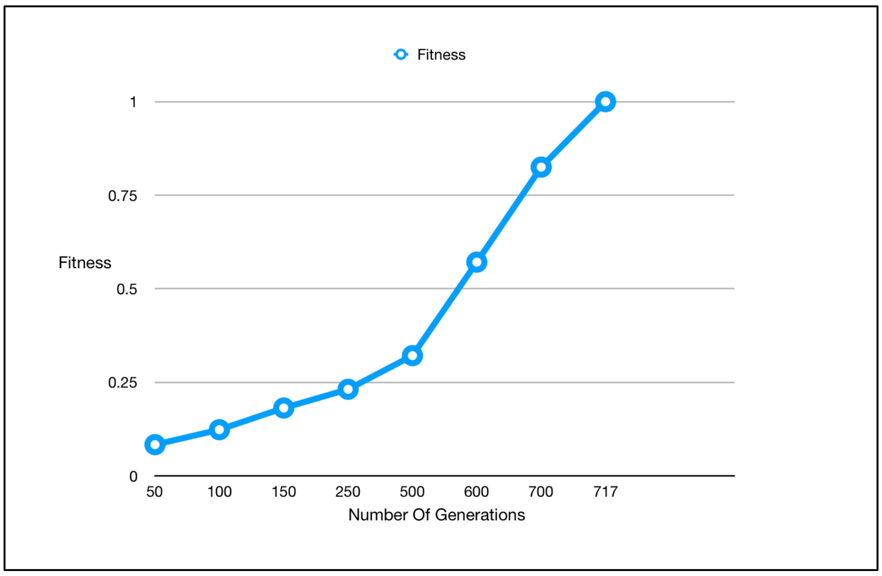


Figure 7.2: Graph of fitness of a generation VS number of generations for 3 teams

- The genetic algorithms parameters were varied and output was noted each time. It was observed that as the mutation and crossover rates were increased, the time taken for getting the optimal solution increased. The graph below shows the time taken for running the genetic algorithm by varying rates.

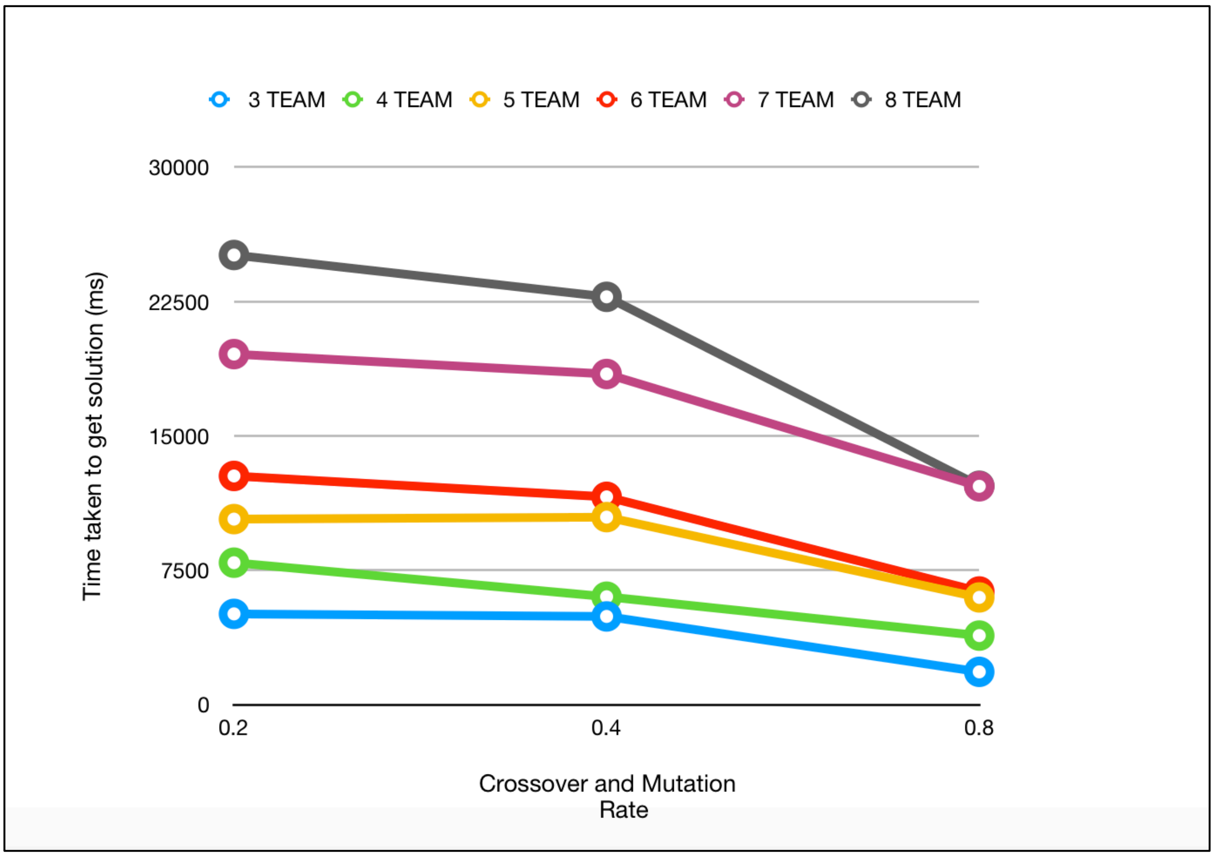


Figure 7.3: Graph of time taken for execution VS cross rate & mutation rate

8. References:

- https://www.tutorialspoint.com/genetic_algorithms/
- https://towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3
- Genetic Algorithms in Java Basics by Lee Jacobson (Author), Burak Kanber (Contributor)
- https://www.whitman.edu/Documents/Academics/Mathematics/2014/carrjk.pdf
- https://www.theguardian.com/football/2013/jun/15/fixtures-premier-league-football-league-compiled