**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 application of Genetic Algorithm is **Scheduling** different events like University classes schedule, meeting 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.

1. **Problem Statement:**

Implement Genetic Algorithm for Football Fixture problem. It is a scheduling problem which will create fixtures of the **English Premier League**. Input the 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.

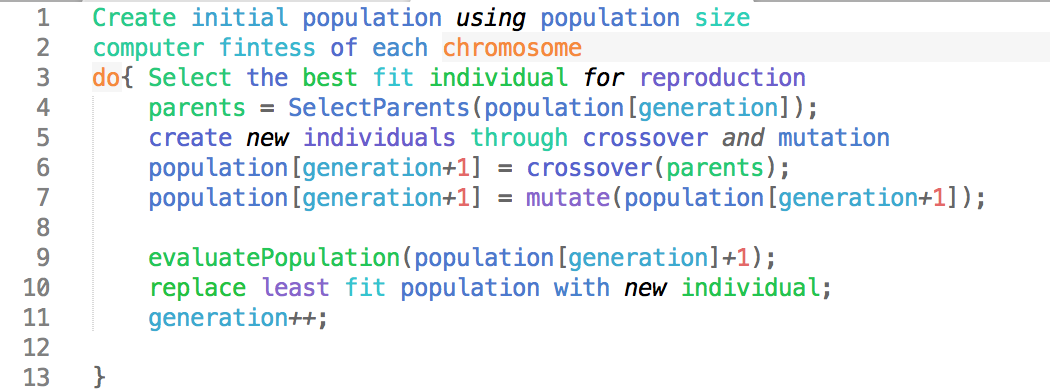
Here,

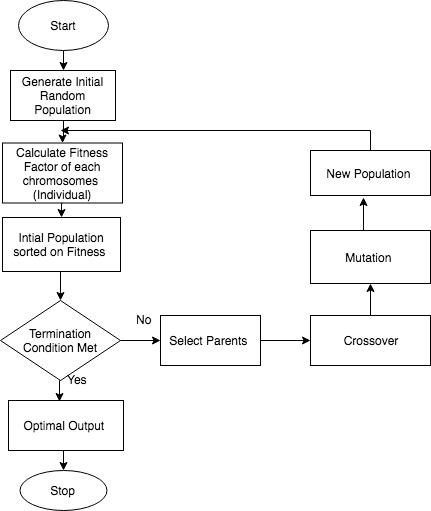
**Total number of matches in a tournament = ((Number of teams) \* (Number of teams - 1) \* (Number of Rounds)) / 2**

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
7. **Implementation Details:**

**Basic Algorithm for GA:**

**Flow Chart Of Genetic Algorithm:**

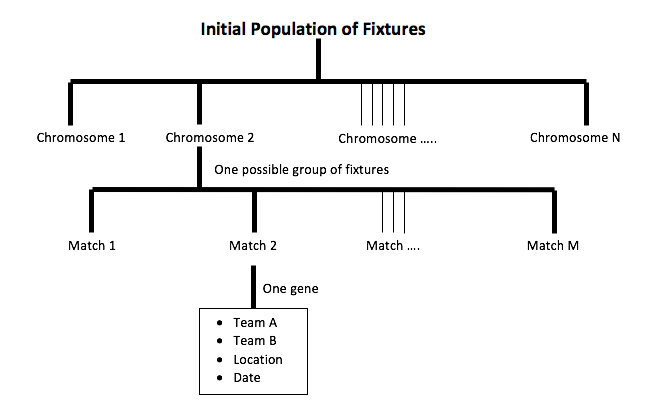
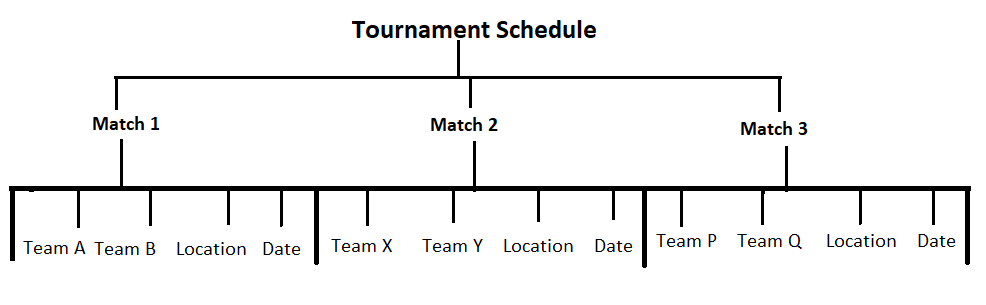


Figure 1: Basic layout of population, chromosome and gene

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.

Ultimate Goal of genetic Algorithm is to solve a optimisation problem.

## Helper classes for Sports Scheduling Implementation:

We have made separate class for Match and Team .

**Match**

In match class we have Information about two teams which are playing against each other, match date and the match location.

**Team**

In Team class we have Team name and the Home stadium associated with the team

**B. Main Classes:**

**Chromosome**

CalculateFitness :

Once our Population has been initialised, we need to evaluate the fitness of each individual in population.We assign fitness score to every chromosomes which is present in Population and see how close it to the desired solution.

we are calculating fitness according to the number of conflicts that arises inside the chromosomes. Conflicts is increased every time there is violation of the hard constraints.

There is a inversely proportional relationship between the Fitness and conflicts.

FITNESS =

**FixturesMain:**

This is our main class this class interact with other classes to make our genetic algorithm program work. It calls configuration class to initialise the data, configuration class assigns Teams, Match Location and dates.

It calls the population class to create initial population and then calls GenetiAlgorithm class to run Genetic Algorithm on the population.

**Termination Condition**

We have used two termination condition for terminating our program

1. When fitness factor get equal to 1. The program get terminated if fitness factor of any individual in the population becomes 1.
2. When maximum generation reaches to MAX\_GENERATION constant value, this help us to limit the maximum no of generation on which we have to run the algorithm for solution.

**Constants**

In this class we have initialised different Constants Involved in genetic algorithms.

POPULATION\_SIZE

NUMBER\_OF\_ROUNDS

ELITE\_FACTOR

K\_FACTOR

MUTATION\_FACTOR

CROSSOVER\_RATE

MAX\_GENERATION

**GeneticAlgorithm**

There are different methods that we have created inside this class

**Parent Selection**

Natural Selection is the main inspiration of Genetic algorithm.In nature the most fittest Individual (Chromosomes), the one having highest fitness factor has high chances of mating and this causes their genes to contribute more in production of the next generation.

There are Different Techniques of Selecting Parents in Genetic Algorithm.

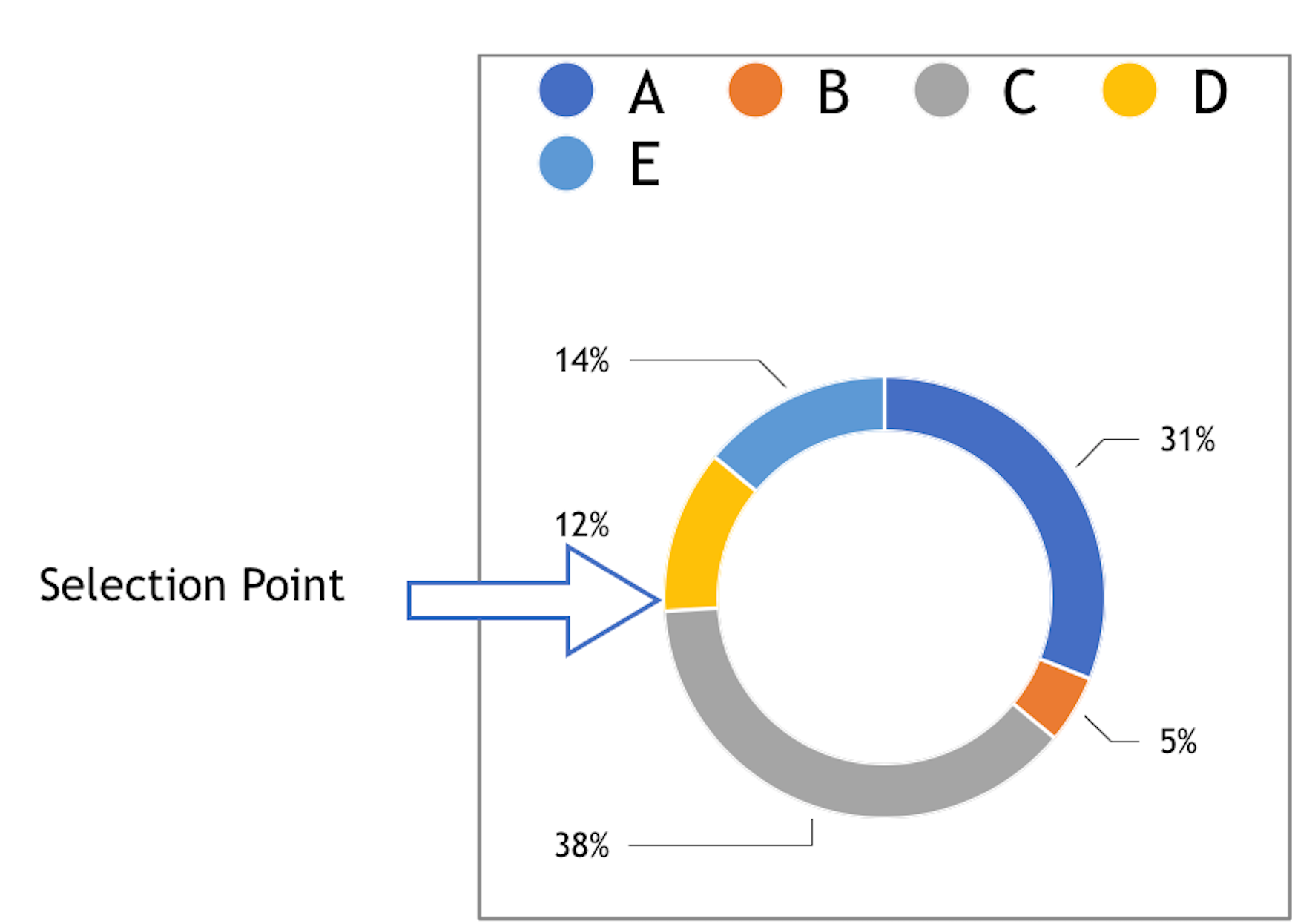
* **Roulette Wheel Selection**

Roulette wheel have segments of different sizes and these segment sizes are decided according to fitness level of individual. So one advantage of this method is that even the weaker individual have some chances of getting selected in mating and increasing diversity in genetic pool.Chances of getting selected depend upon the share a individual is having on the Wheel, The higher fitness factor mean a higher portion of share on wheel which further means higher probability of getting selected for mating.A fixed point know as Selection point is taken and the region of when that comes in front of selection point is taken and this process is repeated for second parent.

|  |  |
| --- | --- |
| Chromosomes | Fitness Value |
| A | 6.82 |
| B | 1.11 |
| C | 8.48 |
| D | 2.57 |
| E | 3.08 |

In this process probability of individual(i) of getting selected is computed as:

P(choice = i) =



**Fig : Roulette wheel for above chromosome values**

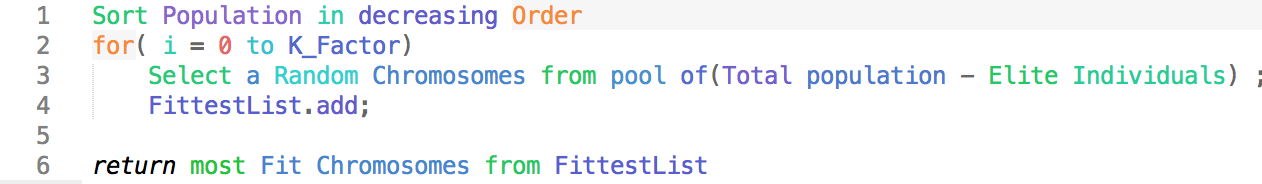
* **Tournament selection**

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

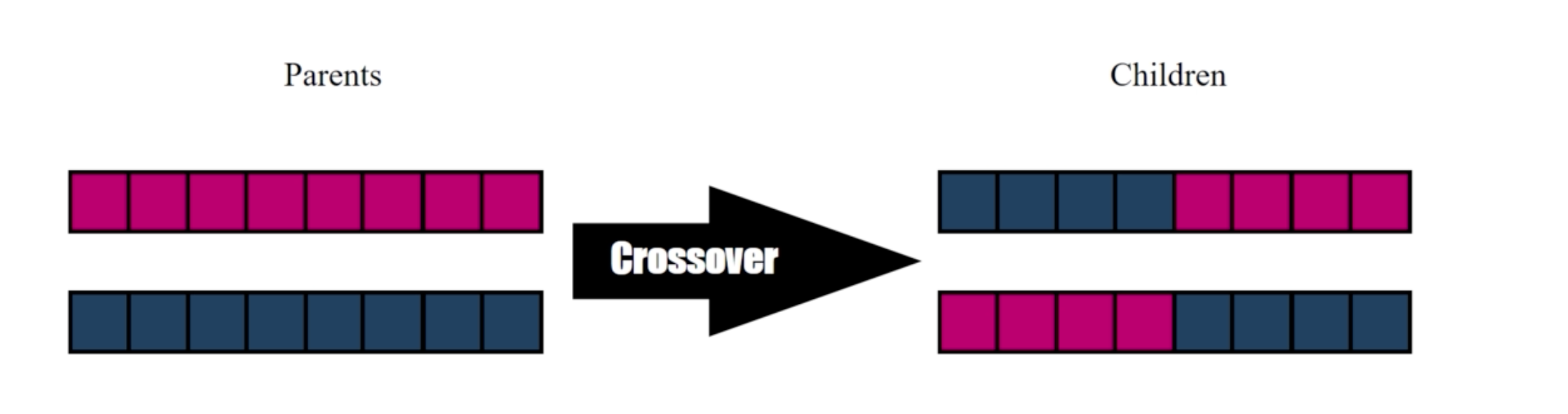
* **Rank Selection**

In this selection, Individuals are sorted by fitness and then we assign raking to individuals. Then every individual is allocated probability with respect to its rank in population. This selection procedure also works for negative value. It also overcomes the scaling problems like premature convergence and stagnation.

We have used **K-tournament selection** for parent Selection.

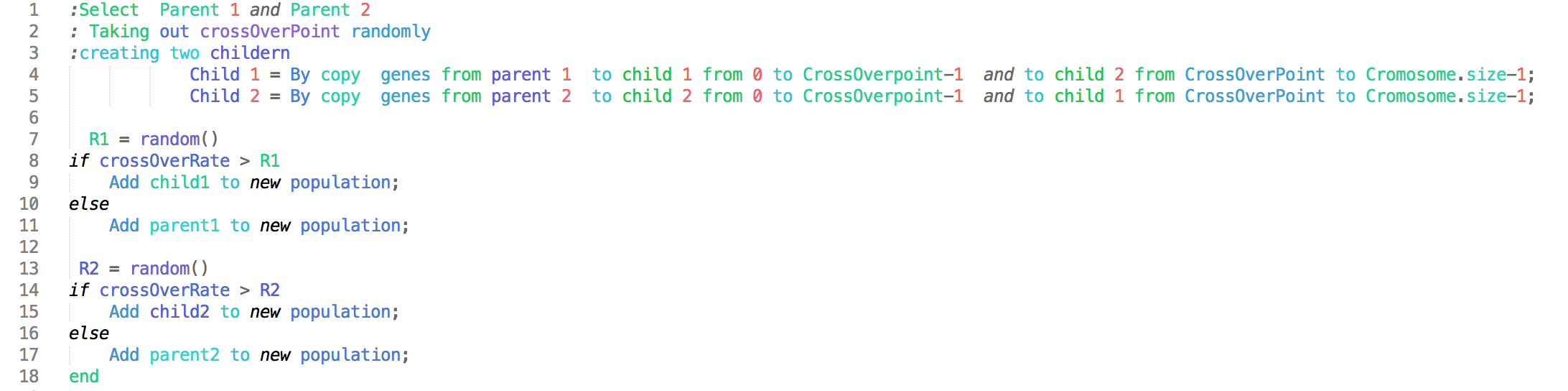
Lets look at the pseudo code for the K-Selection that we have implemented in the project.

**Cross-Over Function**

After selection of two parents we do Cross over. The process of exchanging genes between two chromosome, in this genes between the two parents are exchanged to form a new chromosome or child. Cross-over can be done in multiple ways One point crossover, Multi-point crossover Uniform crossover.

**CrossOver PseudoCode**

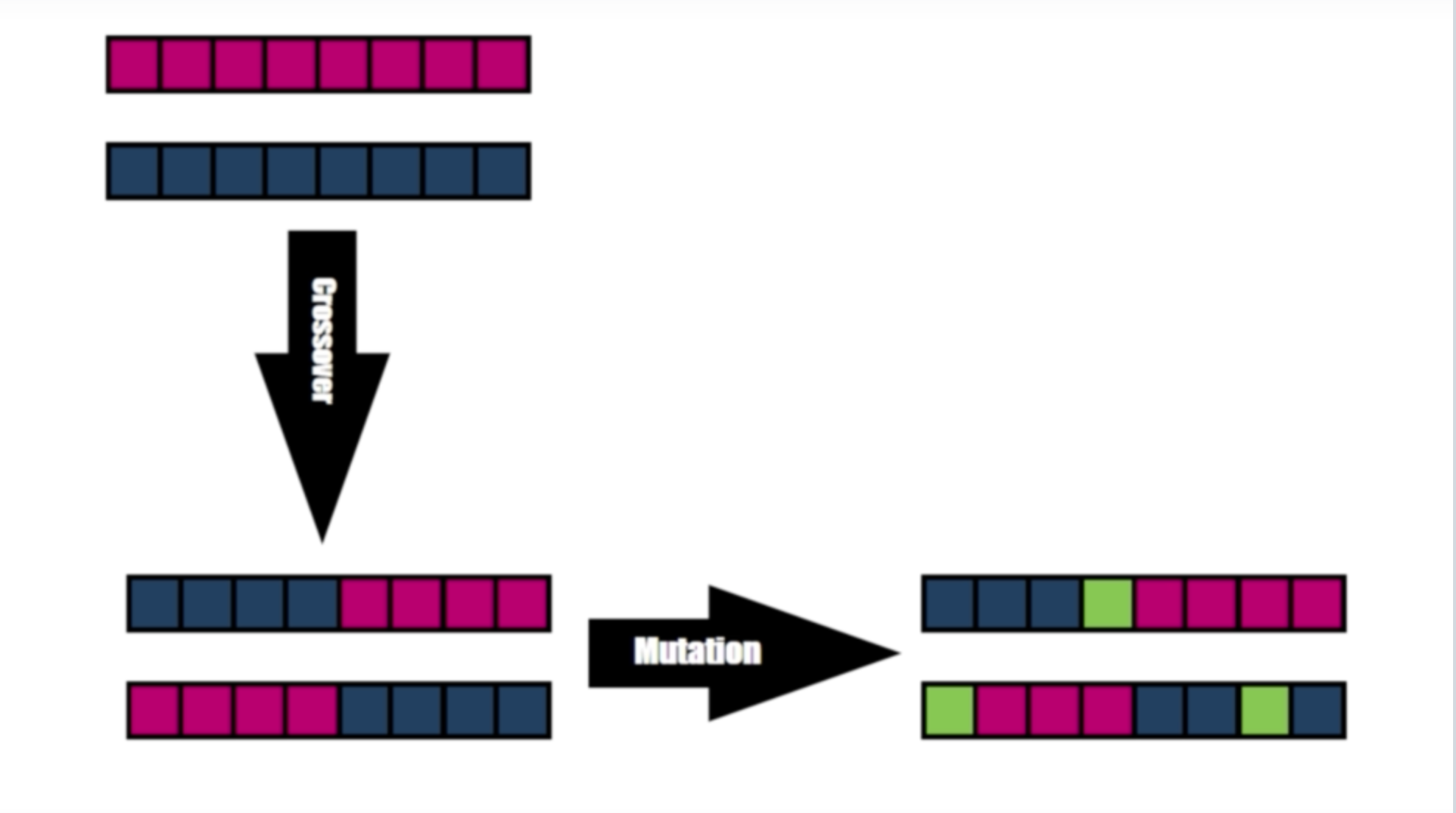
Lets look at the pseudo code for cross over which we have implemented in the project. We are using **single point crossover,** in which **crossover point is computed randomly** to increase diversity of genes in children generations.

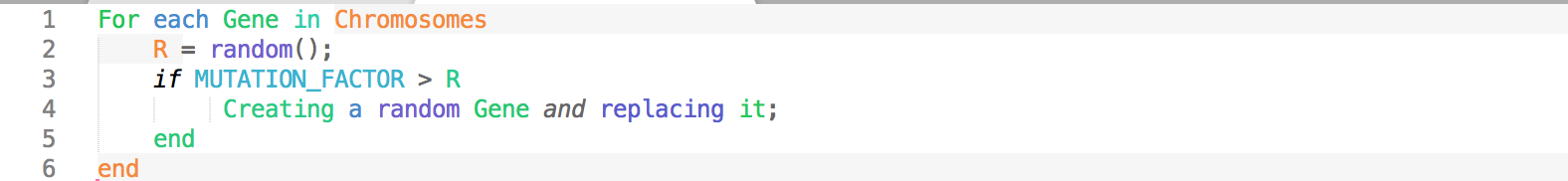


**Mutation**

In order to create more diverse chromosome (individual), Chromosomes goes through mutation.

In mutation some of Genes face random changes in genes after recombination. These changes are completely random and can be good or bad for individual fitness factor.



Lets look at the pseudo code for the Mutation which we have implemented in the project.

**Elitism**

Genetic Algorithm often looses its best Individuals over generation because of CrossOver and Mutation operator. To avoid this we use Elitism Operator to preserve the best genes of the population.

This operator saves the small portion of best individual and call them elite and this elite individuals are copied to next generation without being unaltered.This way the fittest individual are no longer lost between generations.

Number of individual to keep depend upon the Genetic Algorithm problem. We use Elite\_Factor to decide the percentage of population that should be kept and transferred unchanged to the next generation. This parameter is any number between interval of 0 to 1. For example elitism\_factor is 0.08 that means 0.08\*(Population\_Size) Individual should be transferred unchanged to the next generation.

We have Implemented Elitism in project, in our selection process.

public static Chromosome k\_wayParentSelection(Chromosome[] chromosomes)

{//some code

for (int i=0; i< Constants.K\_FACTOR ;i++){

Select a Random Chromosomes from pool of(Total population - Elite Individuals) ;

FittestList.add;

}

//some code

}

**Optimization**

To make our genetic algorithm fast in getting optimal solution, we have used several optimisation technique. These techniques helped us to get result more faster.

we have used

1. Parallel Processing
2. Using Hash-Map for storing values
3. Using knuth-shuffle