DNA Fragment Assembly Using Genetic Algorithm

Tameem Suliman Alsulmi

Table of Content

Abstract

The importance and difficulty of the project DNA fragment assembly problem has been increasing that a fast and accurate assembly is a major part of DNA sequencing projects, the Deoxyribonucleic acid or called “DNA” is an atom made out of two polynucleotide chains that curl around one another to frame a twofold helix carrying genetic instructions, many different algorithms have been built to solve it but not all of the algorithms have a good effect an excellent or good solution close to the exact solution for that we want to select and to choose the perfect algorithm for this hard DNA fragment assembly problem and what can we improve that using crossovers, mutations and many things that we will be covering in this project, an exact solutions cannot be obtained easily Because of the difficulty and complexity of this problem and since this problem is NP-hard. This project aims to give a very comprehensive study, Experiments and analyzes of complex solutions, extracted from the Genetic algorithm, several different crossovers, and several different mutations and stopping conditions for the DNA fragment assembly problem , providing a good read for the biologists and practitioners in that field, we will review the Genetic algorithm, crossovers and mutations in our solutions later on, and we will show the data sets used, and the tests for the Genetic algorithm. We will discuss a very comprehensive and many tests that we done using genetic algorithm and comprehensive compare between the crossovers and mutations and stop conations that has been used in this DNA fragment assembly problem project.

1. Introduction

1.1 Introduction

Our project the DNA Fragment Assembly problem has become a difficult task to any DNA the deoxyribonucleic acid sequencing research, a fast and accurate technique to assemble all DNA fragments is needed. And since this problem is NP-hard, and exact solution is very difficult to achieve. NP-hardness is a non deterministic polynomial time which hardness is meaning the property of class of problems that are informally[38], it is unlikely such algorithm is exist to solve np-hard problem in polynomial time and that why our project DNA fragment assembly problem is considered np-hard problem. NP-hard for our problem is that couldn't within the polynomial time able to find out the time for the algorithm[39], And You couldn't find out the solution within the polynomial time, polynomial itself is very big, and time complexity within the polynomial time is very high, the class P, in which all problems can be solved in polynomial time. In the end that mean the genetic algorithm for our problem couldn't calculate the complexity time.

Our motivation for doing this project is to provide a good study of the Genetic Algorithm used to assemble DNA fragments so biologists and practitioners in this field have a good and clear idea of each algorithm and its weaknesses.

The DNA Fragment Assembly is a procedure that takes a large number of DNA fragments and attempts to reconstruct the original DNA [1], and each of these fragments can be more than a hundred base-pairs(bps) long.

Current technologies cannot accurately and quickly reassemble DNA molecules that long, for example human DNA is more than 3 billion nucleotides long and it is impossible to read them at once.

First, we duplicate the DNA molecule, then we cut the DNA molecules into fragments eliminating the limitation of reading the DNA molecule at once, then we apply the recent algorithms. The cutting and reassembling procedure is called shotgun sequencing [2].

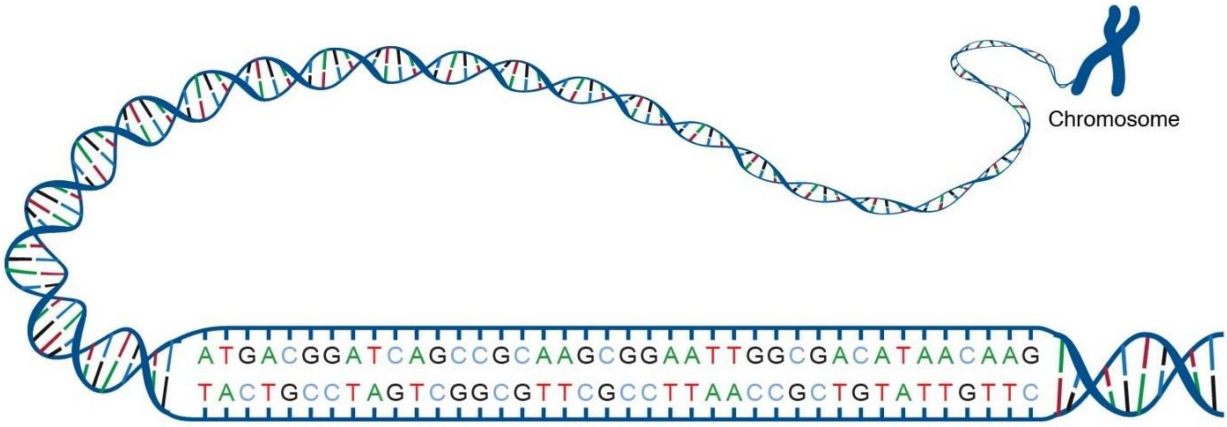


Figure 1: DNA Sequence [7]

To give an example of how DNA fragments should be assembled let us say we have the following DNA fragments as an input: TCGG, GCAG, ATCG, CAGC and GATC, two possible final reconstructions of the fragments are:

GATCGGCAGC (Length = 10)

CAGCAGATCGC (Length = 11)

The first sequence is shorter than the first making it easier to study and analyze.

Terminologies will be used throughout the research:

f: a fragment from a DNA molecule.

Prefix(f): a substring from the beginning of the fragment f.

Suffix(f): a substring from the end of the fragment f.

Overlap(f1,f2): a common sequence from the suffix of a fragment and the prefix of another.

1.2 Applications

DNA Sequence can be used in numerous fields such as forensics, paternity test, diagnosis, biotechnology, Molecular biology and many other fields.

Analyze and diagnose various maladies including different cancers by Contrasting solid and mutated DNA sequences, portray counter antibody repertoire, also can be utilized to manage treatments for patients.

Having a speedy method to succession DNA considers quicker and more individualized clinical consideration to be regulated, and for additional organisms to be cataloged and identified [6].

- **Molecular biology**

Molecular biology uses DNA sequence to know psychiatric illness and genetic diseases as well. Genetic diseases include heart diseases, blood diseases and diabetes. Therefore, DNA includes important information about human body [3].

- **Forensics**

DNA sequence is used in crimes where DNA is taken from blood, hair, and from many other things. This helps to know if the accused person is convicted or innocent [4].

- **Paternity**

One of the uses of DNA sequence is paternity test. From such a test, the real father of a person can be determined. It also can be used to know the relation between a person and another [5].

- **Evolutionary biology**

This science studies how living creatures relate to each other through generations and how they evolve. There studies can be done by using DNA Sequence [6].

1.3 Research Scope

The research will provide a comprehensive study about comparison between the crossovers, mutations and stop conditions by using Genetic algorithm that has been used in Artificial Intelligence in various implementations and studies for our project used to reassemble the DNA fragments to aide biologists and practitioners in biology field by reassembling the DNA fragments into a sequence that can be easy to analyze. An access to digital libraries is needed to review the literature and related works. We will use Java programming language to implement the genetic algorithm for the comparison.

1.4 Aims and Objectives

The aims and objectives of this project are:

- Study the proposed genetic algorithm, crossovers and mutations in the past works.
- Design and implement them using Java programming language.
- Provide a comprehensive analysis for comparison for the implemented genetic algorithm, crossovers, mutations and stop conditions.
- Mack a comprehensive comparison between the selected crossovers, mutations and stop conditions.

1.5 Methodology

- **Defining and analyzing the problem**

  Study the problem using research papers related to it.

  Define the problem, explain terminologies used and the aim of the research.

  Review some applications of the problem on the related fields.

  Review Literature.

  Analyze the proposed contributions and how they solved the problem.

- **Design and implementation**

  Implementation of the Genetic algorithm using Java programming language.

  Implementation of the many different Crossovers using Genetic algorithm.

  Implementation of the many different mutations that can work with the crossovers using Genetic algorithm.

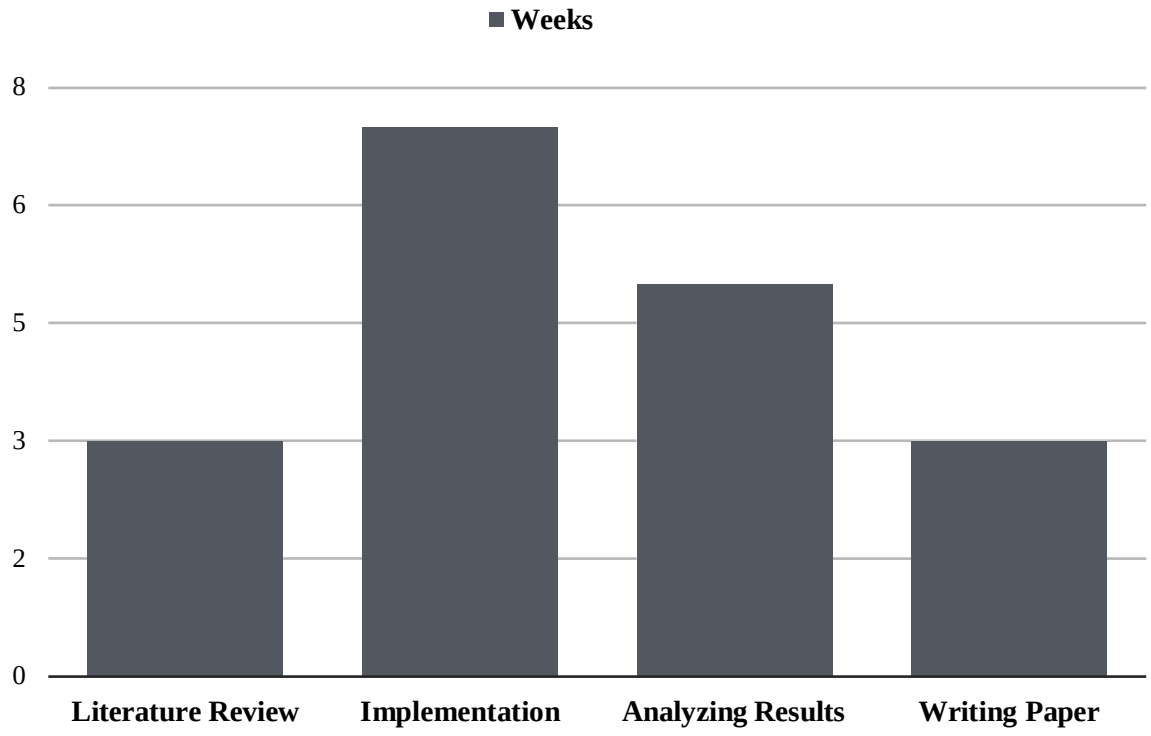
- **Analyzing and compare the Crossovers**

  Analyzing the performance and result of each Crossover we implemented.

  Analyze the genetic algorithm that we will selected.

  Analyzing each of Mutations we implemented.

1.6 Research Plan



1.7 Team Qualifications

Tariq Alashaikh	Took an AI course, Algorithms design and analysis and Java programming language course.
Abdulmajeed Alsufayan	Took an AI course, Algorithms design and analysis and Java programming language course.
Tameem Alsulmi	Took an AI course and Java programming language course.

1.8 Conclusion

We described how important the DNA fragment assembly problem is and how a good algorithm could shorten the target DNA sequence providing by that the ease to study and analyze DNA sequences to find diseases and abnormalities. And implementing it and analyzing the results should give us the good and bad of each crossover and mutation, in the Applications we how important is this project like for the forensics, then research scope of this project is comprehensive study using the genetic algorithm also aims and objectives study the genetic, crossovers and mutations after that the design and implementation for comprehensive comparison. Methodology defining and analyzing the problem and the design implementation using Java also how analyzing and compare the crossovers.

2. Literature Review

2.1. Introduction

We studied, analyzed and understood many Algorithms and ways that can be used for our project DNA fragments assembly problem, we will discuss that in 2.3 Related works section. for understanding and to find the efficient way for our problem by searching many different researches that have same problem DNA fragments assembly problem or similar problem to help us find better Algorithms and ways, then we decided to use Genetic Algorithm for our problem after many discussions, because it is the best algorithm for our problem. Our problem it is important to biological researchers and human health and we will talk about that more in 2.2 Background next section.

2.2. Background

Our project DNA fragments Assembly problem it's very important because it is NP-hard problem and important in human health and biological filed by helping biological researchers to be able to simplify the human DNA sequences, organism contains "the molecule of life"[8], named deoxyribonucleic acid or DNA, Encoded inside this DNA are directions of Characteristic of a person as assorted as the shape of the head, color of the eyes and the manner by which the bacteria infect a specific cell. The DNA found in almost all living cells the, reasons for simplifying the DNA to help biological researchers they then can understand human DNA more that is very good for all humans to find the secrets of our DNA and can use that for our benefit to human health and finding way to make the immune system stronger also invent cures to many sick people that is important to us and all kind of people we hope for that to improve the general health. almost all organisms have their genetic material codified as DNA, each person has unique DNA, every DNA sequence consist of these four nucleotides, [G] Guanine, [T] Thymine, [A] Adenine and [C] Cytosine.

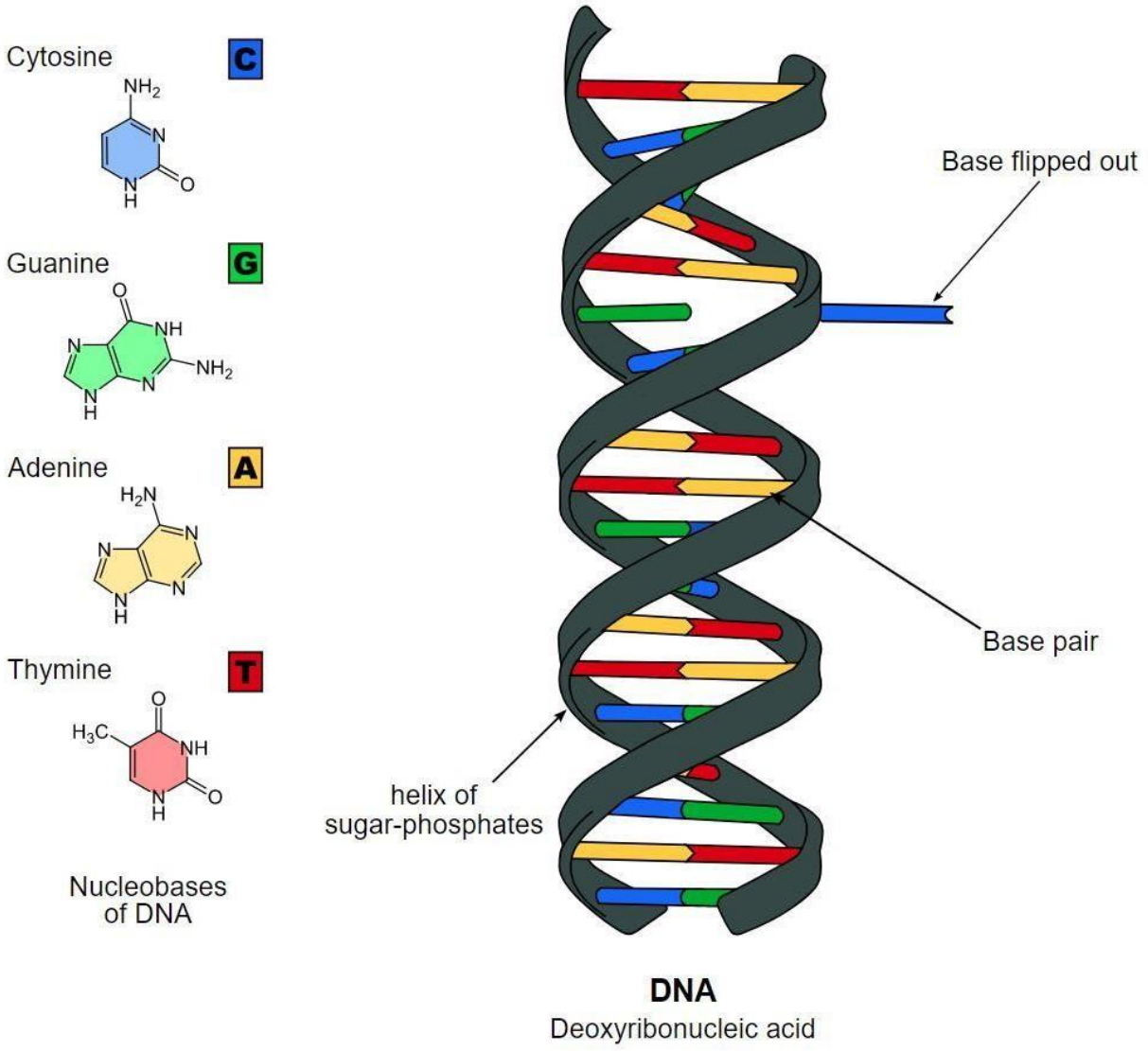


Figure 2: Structure of the DNA [37]

Simplify the long which is complicated DNA sequences for the biological researchers and scientists to comprehend the functioning of different organisms by cutting the DNA into fragments and Assembly them, can do that by using one of many different Algorithms that leads to rebuild the first original sequence of DNA from an enormous fragments number that are a few hundred base length. The deoxyribonucleic acid or DNA transfer genetic information that is stored in it from both the parents to the child like the characteristics of parents, eye skin colors, how hair looks like the color of it, blood type and many different inherited characteristics also diseases can be inherited. The mutation definition is the alternate that is permanent in the DNA sequence what

makes the gene, mutation can be inherited from parents or can be abnormal gene from none of the parents the problem is the mutation maybe affect seriously of child health and become disability to child or more.

2.3. Related Works

In this section we will presents the algorithms that has been used to solve our problem DNA fragment assembly problem in other research papers, the algorithm that we used and why, and what we implement and mention the research papers. The algorithms that has been used and some information about the algorithms:

- Greedy Algorithm

  The Greedy Algorithm was invented by Huffman. In 2004, they use the Greedy Algorithm to calculate and compute contig overlaps instead of fragment overlaps there is two advantages for that [9], so that empower us to pick just evident overlaps to make record and it grant a superior assurance for finding the fragments orientation. And the significant steps of Greedy Algorithm are step number one is constructing the Best Set of Maximum Weight Contigs [BSC] and step number two is ordering the Maximum Weight Contigs [MWC]. since the earliest reference of DNA fragment greedy algorithm has been utilized for occurrence how by picking a random fragment then associate with it the one that has the longest overlap, this step is redoing it until no overlap fragment are accessible. These days this strategy would be hard to implement since these fragments are excessively shot which regularly creates a few unique fragments with a similar size of overlap. Because of that we did not used this greedy algorithm to solve our problem after understanding it and reading in the research papers.

- Simulated Annealing Algorithm

The Simulated Annealing Algorithm was invented by Kirkpatrick 1983, Aarts 2005, Simulated Annealing Algorithm based on hyper-heuristics [1][9][10]. The early concept of Simulated Annealing Algorithm was derived from tempering of a solid. It is a stochastic enhancement procedure, which has its starting point in factual mechanics. It is in view of cooling methodology utilized in manufacture. This methodology warms the material to a high temperature with the goal it turns into fluid then atoms can move generally free. Then the temperature is gradually brought down so that at every temperature the atoms can move enough to start receiving the steadiest design. On basic level, on a fundamental level, when the material is cooled gradually, then atoms can arrive at the steadiest (ideal) design. This process the soft cooling is recognized as annealing. The Simulated Annealing is widely applied to tackle complex combinatorial issues by using an irritation searching methodology. The simulated annealing algorithm is broadly used in to solve combinatorial issues with what it is called neighborhood search methodology and starts with underlying random solution that was generated by utilizing temperature esteem then another state is produced from a present status utilizing an irritation strategy and the Eq is expressed caused change in the energy, delta E equal to the f new minus the f current and the f new is new state value of the new objective, f current is right now the best arrangement esteem, new state is replaces and accepted solution of the current is when the delta E less than zero nonetheless if the value is not better than present best solution the new state can be accepted if the it is satisfying the Boltzmann distribution. The Simulated annealing procedure stops when a temperature arrives at zero or iteration number is surpassed.

- Clustering Heuristic Algorithm

The Clustering Heuristic Algorithm was invented in 1950s. The classic three stages are layout, consensus, and overlap, are utilized in this algorithm. Using the semiglobal alignment algorithm so can locate every conceivable pairwise overlaps [8]. At point when the covers are resolved, using a greedy heuristic in the layout stage to discover the numerous sequence alignment between a lot of fragments. Taking the fragments pair at the beginning stage with most elevated overlap. The layout is built at progressively way including the most elevated overlap fragment with the amassed fragments. The thought is in view of the clustering principle. It's implies to the recent included fragment to the alignment that has outstanding overlap between the first fragment or the last fragment in the present alignment. Each fragment is dynamically included into the current alignment till no other fragment is available. The clustering heuristic algorithm was used in "A Comparison of DNA Fragment Assembly Algorithms" [8] research paper that was comparing various of algorithms one of them is the clustering heuristic algorithm, the clustering heuristic is slower and it is decent in the small data sets of our DNA fragment assembly problem but not in the larger data sets and it is less accurate in the results.

The algorithm that we pick to use for our project is Genetic Algorithm after many researches and discussions we decided to implement using the genetic algorithm for the DNA fragment assembly problem project.

- Genetic Algorithm

The Genetic Algorithm was invented in 1970 by John Holland, the Genetic Algorithm stand on the Theory of Darwin Evolution. Back in 2005 Genetic Algorithm was related to the utilization of a binary representation yet these days in 2005 different types representations can be found and Genetic Algorithm typically set on two tentative solutions a recombination operator [1]. A Genetic Algorithm can be utilized for improvement problems with numerous parameters and various objectives. This algorithm is usually used to handle NP-hard problem like our problem the DNA fragments assembly project. There will be more details, information and how the implementation of the genetic algorithm in the next chapters.

2.4. Conclusion

The main points in the background is to know and understand the main concepts and definitions of the human DNA that we need to know for our problem and how important is that. The main points in related work is to analysis DNA fragment Assembly problem researches that has been done and understand the using algorithms and how its work in our problem. The importance of our project is very high because, it is very helpful for the health of our world, by helping the biological scientists that are trying their best to improve and increase the medicine and global health to protect humans from disease. The next chapter will present to the methodology of our project and the theoretical and experimental design.

3 Methodology

3.1 Introduction

In order to achieve our goal which is to study the proposed solutions that has been used to reassemble DNA fragments into a shorter yet complete sequence that can help biologists analyze the sequence and discover diseases and abnormalities. we present in this chapter the methods we will use to achieve our goal in this project.

3.2 Theoretical formalization and algorithms design

An algorithmic approach will be used in this project, every sequence will be evaluated using the *Overlap(f1, f2)* function that calculates the overlapped segments from Suffix(f) of a fragment and the Prefix(f) of another to determine how more or less optimized this sequence than the other sequences different algorithms can produce. Every solution must be complete meaning that every DNA fragment must appear in the solution.

3.2.1 Existing Solutions

Current packages used to assemble DNA fragments are PHRAP [11], TIGR assembler [12], STROLL [13], CAP3 [14], Celera assembler [15], and EULER [16]. These packages tackle most of the DNA fragment assembly challenges, but none of them solves them all. These packages use a variety of algorithms and most of them are greedy based.

- Greedy Algorithm

The Greedy Algorithm is based on the Best Set of Maximum Weight Contigs Approach [21]. The algorithm takes in consideration unknown orientation of fragment, since a fragment can be read either from 5' to 3' or 3' to 5', and missing fragments. The first step of this algorithm is to construct Best Set of Maximum Weight Contigs (BSC) and this step takes $O(n^2l^2)$, where n is the total number of fragments and l is the average length of a fragment. This algorithm takes a set of fragments as the input and outputs the best sets of contigs stored in a vector template, and a linked list containing the overlap score in a decreasing order. The second step is ordering the contigs, this step's complexity is

$O(m^2l^2)$, where m is the total number of contigs, and l is the average fragment's length. It takes the first step's output as the input and outputs a list of the ordering of the contigs.[17]

- Simulated Annealing

Simulated Annealing [18] is a stochastic technique based on a cooling procedure used in industry. This technique starts by heating the material with high temperature so it turns into liquid making the atoms move freely, then the material is slowly cooling down by lowering the temperature, what happens is that by cooling the material slowly the atoms start adopting the most stable configuration continuously, and with that once the material turns into solid the structure of the atoms will have the optimum configuration, and that slow cooling process is called annealing. Figure 1 shows the structure of Simulated Annealing [1]. T is the temperature and s0 is the starting solution (lines 2-4), a new solution will be accepted as a current solution only if $\delta = f(s1) - f(s0) < 0$. Also accepting solutions that increases the objective function value with a probability $exp(-\delta/T)$ *if* $\delta > 0$ prevents this technique from getting stuck in a local maximum. The number of iterations controlled by *Markov_Chain_length* parameter, which means a sequence of accepted solution is a sequence where each state depends on the previous one. The entire process is repeated until we reach a solid state.[1]

```
1   t = 0
2   initialize(T)
3   s0 = Initial_Solution()
4   v0 = Evaluate(s0)
5   repeat
6       repeat
7           t = t + 1
8           s1 = Generate(s0,T)
9           v1 = Evaluate(s0,T)
10          if Accept(v0,v1,T)
11              s0 = s1
12              v0 = v1
13      until t mod Markov_Chain_length == 0
14      T = Update(T)
15  until 'loop stop criterion' satisfied
```

Figure 2: Scheme of Simulated Annealing

- Clustering Heuristic Algorithm

A semiglobal alignment algorithms is used to find all pairs. After that a greedy heuristic is used to find the multiple sequence alignment in a set of fragments. The pair with the highest overlap score is taken as a starting point. The layout is built by continuously adding the fragment that has the top overlap score with the already constructed layout either from the start or the end of it, taking into consideration unknown orientations. The first step is to construct a table with the overlap score of all possible pairs considering unknown orientations. The second step is to sort the score in decreasing order and inserting all pairs with overlap score above some threshold into a linked list, a positive score means that the two fragments are from the same strand and a negative score means otherwise. The third step is to choose the first node in the linked list to be out starting point. The fourth step is to go traverse the linked list and check whether the pair of fragments can be inserted in the front or back of the current layout. Otherwise we put the pair of fragments in a temporary linked list. The fifth step is to create a new contig and remove the nodes that are in the fragment set from the temporary linked list and use the temporary linked list as the current linked list and continue from the third step until no fragments are left.[17]

Here we explain the design of the genetic algorithm as a whole and all the algorithms used in the genetic algorithm

3.2.2 Genetic Algorithm

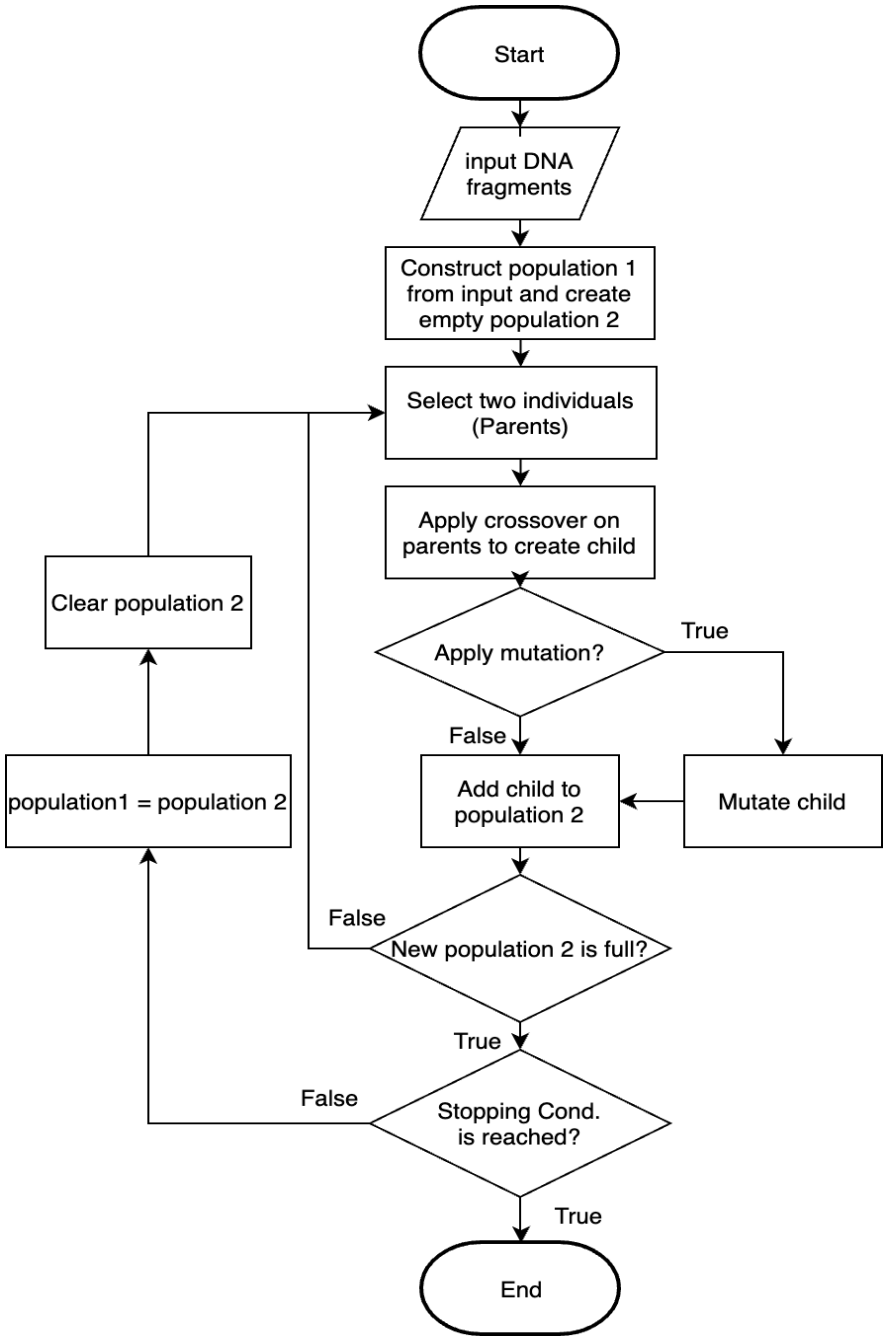


Figure 3: Flowchart of Genetic Algorithm

3.2.3 Fitness Function

The fitness function determines the quality of an individual in the population, and in this problem, we calculate the fitness function by finding the amount of overlap between each fragment and the one after it for the whole individual.

1. *initiate overlap = 0*
2. *for i from 0 to individual length – 2 do:*
3. *check for the mutual substring from the end of fragment individual[i] and the beginning of fragment individual[i+1]*
4. *add the length of the substring to overlap*
5. *return overlap*

Figure 4: Fitness Function

3.2.4 Selection

The selection process selects the best (most fit) two parents from the population, we do that by first sorting the whole population for the first time in descending order by calculating the fitness function for all individuals in the population. Then going through the whole population giving each individual %50 chance of getting selected. This ensures some variety in the generated population instead of always selecting the best two.

3.2.5 Crossover

Crossover is the operation of obtaining the child by applying crossover algorithms on the selected parents.

Order-based Crossover (OX2):

This crossover selects random positions in the first parent and re-order the selected elements of the first parent in the second parent based on the order of appearance in the first parent which produces the first child, and the process is done again with the same positions by switching the parents to produce the second child.

For example, we have these parents:

Parent 1

A	B	C	D	E	F	G	H
1	**2**	**3**	**4**	**5**	**6**	**7**	**8**

Parent 2

H	E	D	C	A	G	B	F
1	**2**	**3**	**4**	**5**	**6**	**7**	**8**

Consider picking the positions 2, 4 and 7 in Parent 1.

Parent 1

A	B	C	D	E	F	G	H
1	**2**	**3**	**4**	**5**	**6**	**7**	**8**

Parent 2

H	E	D	C	A	G	B	F
1	**2**	**3**	**4**	**5**	**6**	**7**	**8**

Now we re-order the same elements in the Parent 2 based on the order they appear in Parent 1, and because B comes first then D second and G last, we get this order (Child 1).

Child 1

H	E	B	C	A	D	G	F
1	2	3	4	5	6	7	8

We repeat the process by switching the parents and using the same positions.

Parent 1

A	B	C	D	E	F	G	H
1	2	3	4	5	6	7	8

Parent 2

H	E	D	C	A	G	B	F
1	2	3	4	5	6	7	8

Now we re-order the same elements in the Parent 2 based on the order they appear in Parent 1, and because E comes first then C second and B last, we get this order (Child 2).

Child 2

A	E	C	D	B	F	G	H
1	2	3	4	5	6	7	8

The two processes are repeated until we reach the number of the population.

Sequential Constructive Crossover (SCX):

SCX gave excellent results in TSP problems [19] and since the DNA fragment assembly problem is similar to TSP and QAP [20][21] this makes it a good crossover for this problem. SCX construct the child by inserting the element that have the best value to the child; the element that increases the fitness function of the child. SCX does not rely only on the structure of the parents.

Step 1: Select the first element of the first parent, naming it element n and inserting it in child.

Step 2: Sequentially go through the elements of both parents after the selected element and find the first element in each parent that is not in the child, if no element was found in a parent, go through the original sequence (The sequence to be optimized) and find the element.

Step 3: Assuming elements a and b were found, we calculate the fitness function between the two element and the last element in the child.

Step 4: If f(element n, element a) is better than f(element n, element b), then add element a, otherwise add element b. If the child is complete, then stop, otherwise the inserted element becomes element n and go to step 2.

Example: Assume we these parents.

Parent 1

A	B	C	D	E	F	G	H
1	**2**	**3**	**4**	**5**	**6**	**7**	**8**

Parent 2

H	E	D	C	A	G	B	F
1	**2**	**3**	**4**	**5**	**6**	**7**	**8**

The first element selected is A from Parent 1. Going sequentially through the parents we have B in Parent 1 and G in Parent 2.

Parent 1

A	B	C	D	E	F	G	H
1	2	3	4	5	6	7	8

Parent 2

H	E	D	C	A	G	B	F
1	2	3	4	5	6	7	8

Suppose that f(A, B) is better than f(A, G). Then we insert B in the child.

Child

A	B						
1	2	3	4	5	6	7	8

The next selected element will be B, and going through the parents starting from B, we get C from Parent 1 and F from Parent 2.

Parent 1

A	B	C	D	E	F	G	H
1	2	3	4	5	6	7	8

Parent 2

H	E	D	C	A	G	B	F
1	2	3	4	5	6	7	8

Suppose that f(B, F) is better than f(B, C), we insert F in the child.

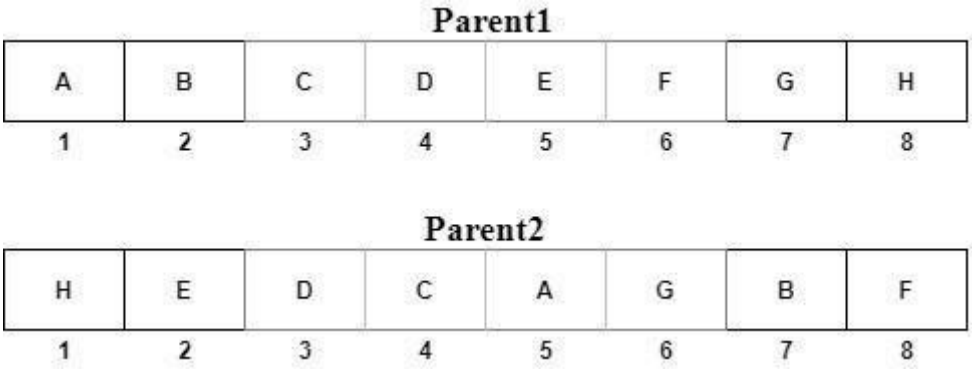
Child

A	B	F					
1	**2**	**3**	**4**	**5**	**6**	**7**	**8**

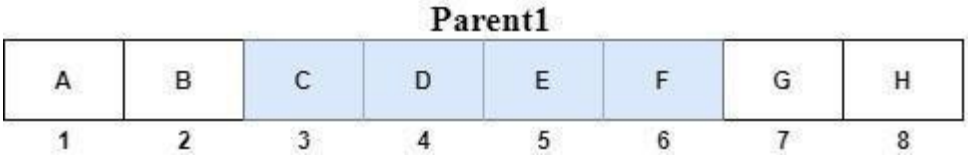
We repeat the process until the child is complete.

Partially Mapped Crossover (PMX):

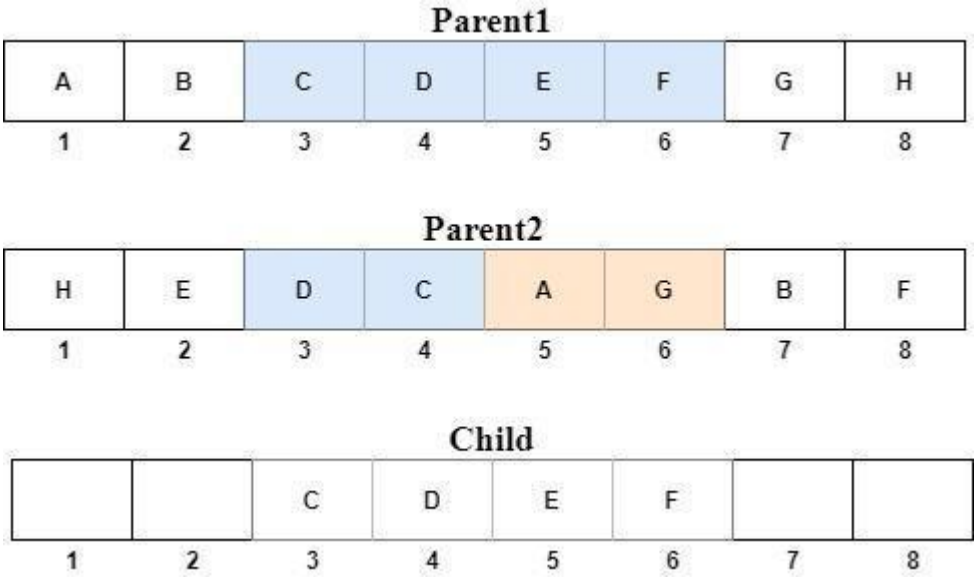
The PMX was proposed by Goldberg and Lingle in 1985. Its goal is to create an offspring from the parents where every position is occupied by a corresponding element from one of the parents [22]. For example, we assume we have two parents (Parent1, Parent2).



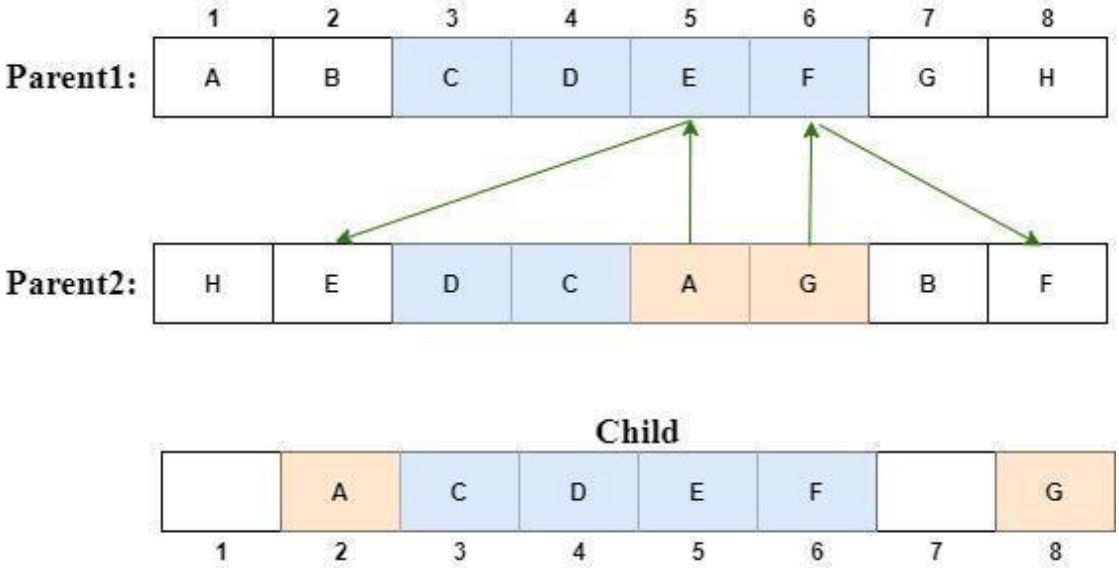
Substring will be randomly chosen from Parent1. Index 3 to index 6 of the first parent was randomly selected, as shown in the following figure.



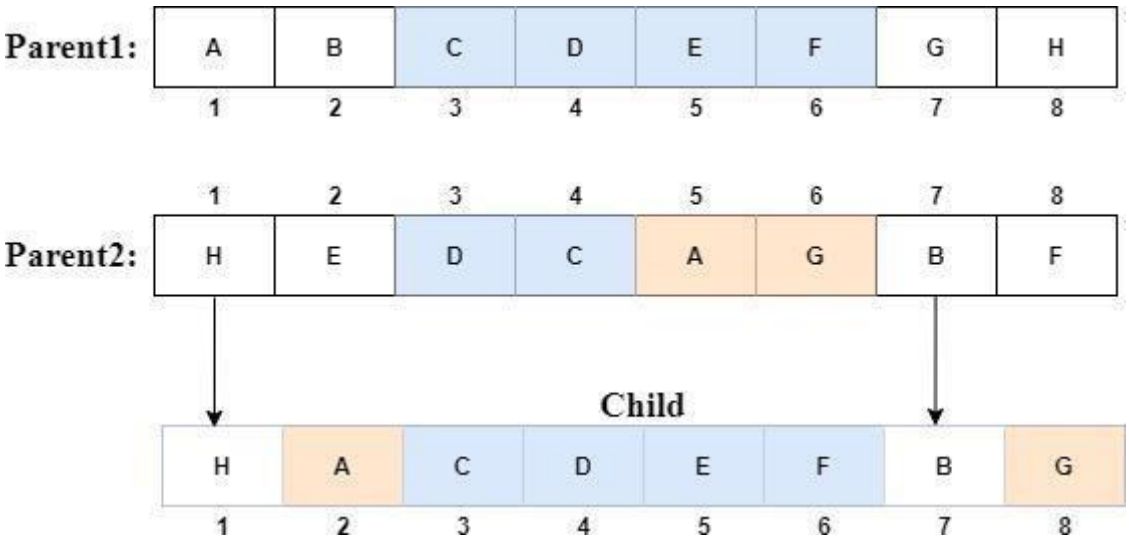
The selected Substring is copied from parent 1 to child, then index 3 to index 6 of parent 2 is viewed. We search for items that were not copied from parent 1 and are present in SUBSTRING in the second parent (A, G), and ignore the elements that were copied to the child (D, C) as shown in the following figure.



Now we will search for a place (A, G), A in parent 2 corresponds to element E in the first parent, E field in the second parent is 2, now A is copied to the child in field 2. We do the same process for element G. This process is illustrated in the following figure.



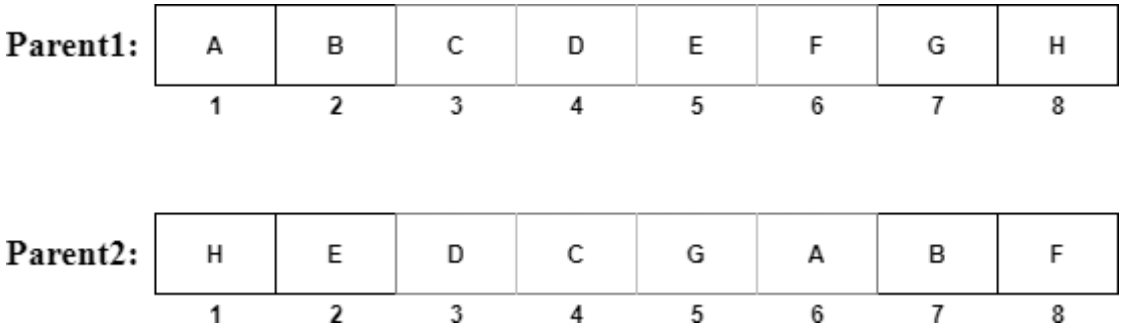
The remaining items are copied from Parent 2 in the same index. The process is repeated with parent 2 instead of parent 1 to create another child.



The PMX works well with Stop condition B. In general, it is considered one of the best crossover we have used in our project. PMX features, gives good solutions, gives diversity in solutions, and ease in application of its idea.

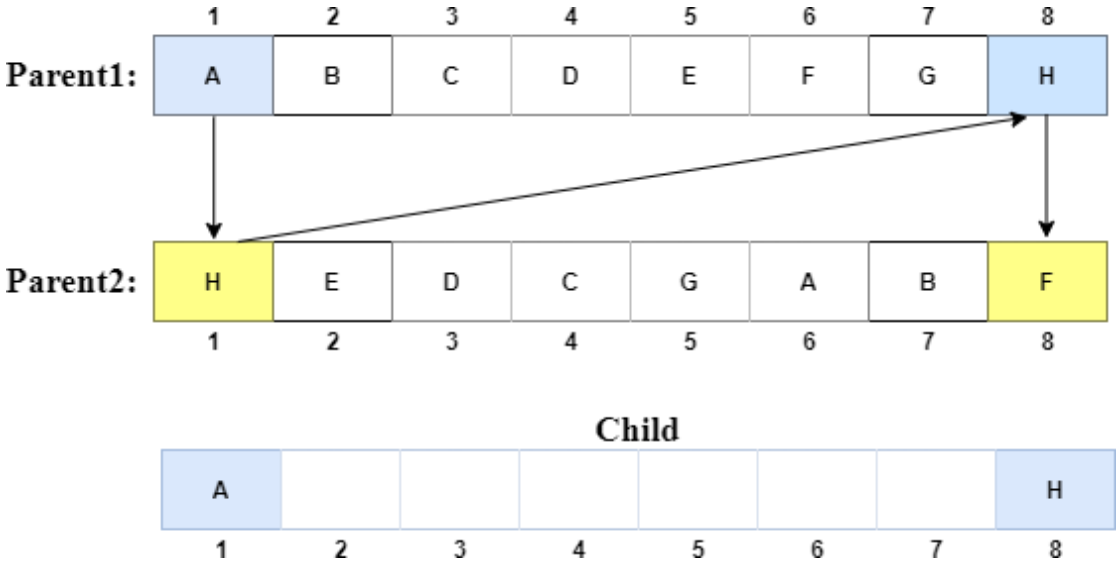
Cycle Crossover (CX):

The cycle crossover operator was suggested by Oliver et al. in 1987. It aims to create a child from the parents where every position is occupied by a corresponding element from one of the parents [22]. For example, we assume we have two parents (Parent1, Parent2).

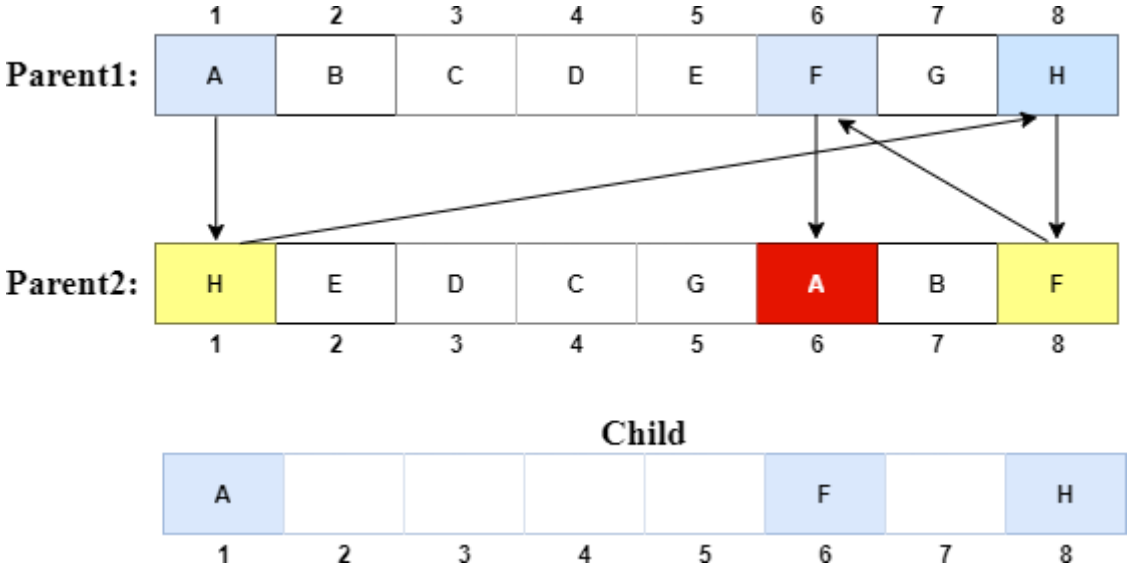


Now we select the first index from Parent 1 "A", and we store it in "Index 1" in the child, then we look at the corresponding component in Parent 2. In this example the corresponding element is "H".

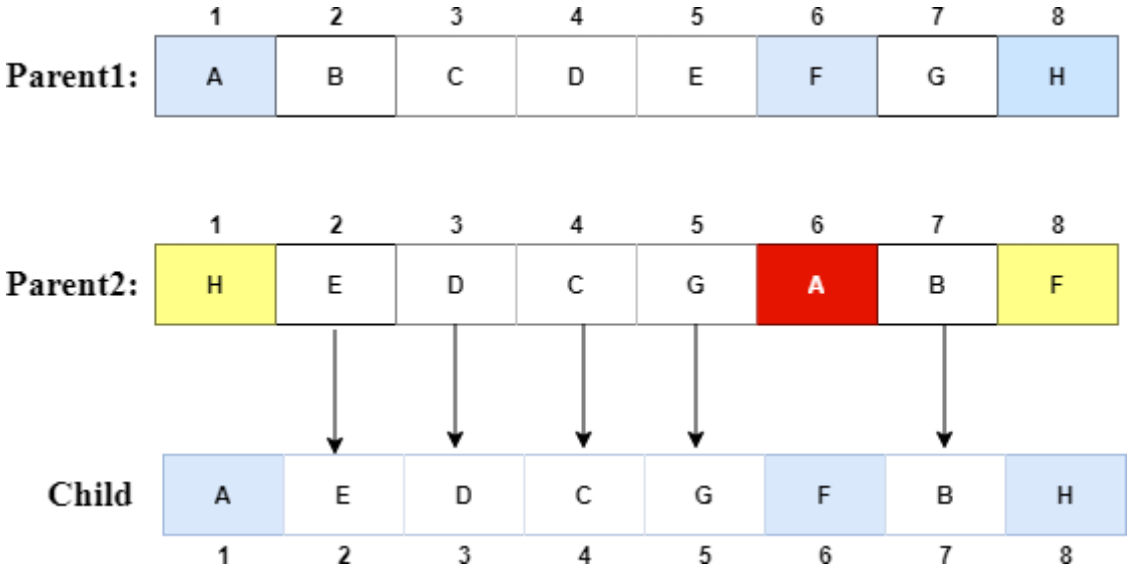
Now we are searching for the corresponding "H" element in parent 1. The item "H" appears in index 8, and we store it in "Index 8" in the child. As in the following figure.



We proceed on the previous process until we reach the first element that has been added. In our example, the first element is "A". Then we pause as it is in the following figure and move on to the next step.



Now we store items without color in the same index in the child as it is in the following figure. After this step, we reverse the role of the parents to get another child.



The cycle crossover operator gives good results, but there are better crossovers operators ones. It is easy to apply, this is an advantage, but sometimes the child may appear to us like his parents.

Edge Recombination Crossover (ER):

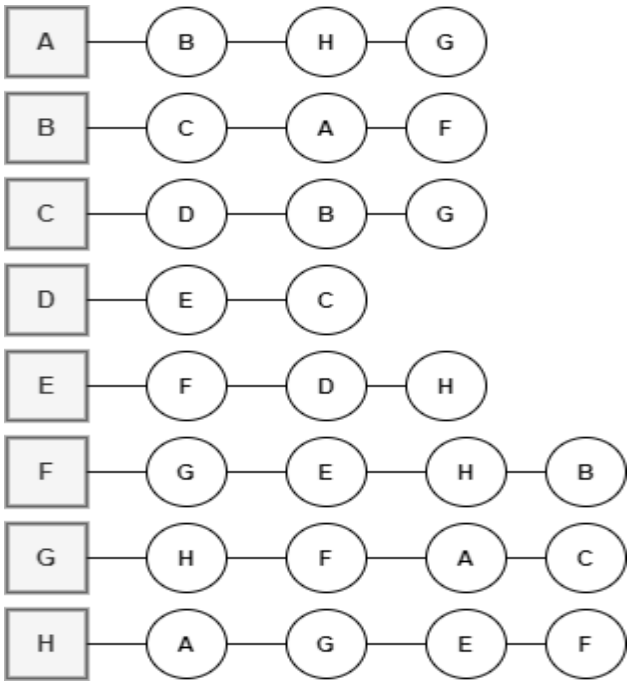
ER which is the edge recombination operator is an operator that creates an individual that is similar to a set of existing parents, by looking at the edges rather than the vertices.[23]. To clarify the idea assume we have two parents (Parent1, Parent2).

Parent1:	A	B	C	D	E	F	G	H
	1	2	3	4	5	6	7	8

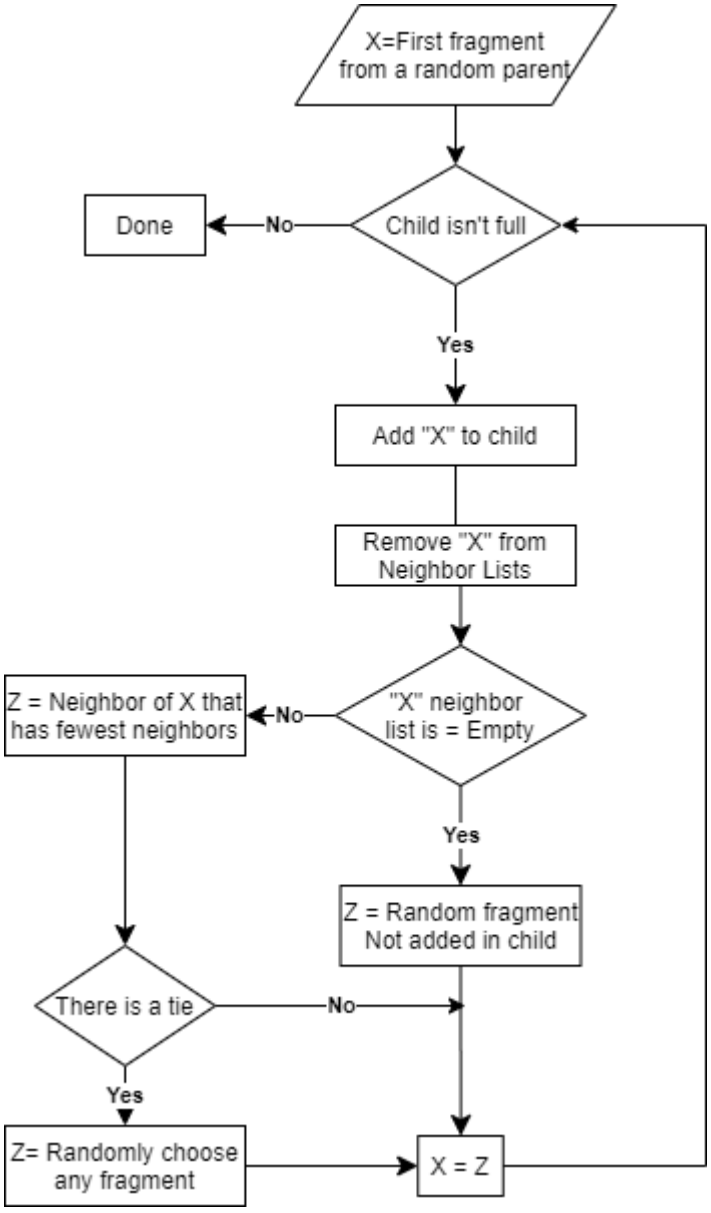
Parent2:	H	E	D	C	G	A	B	F
	1	2	3	4	5	6	7	8

Initially it will Generate Neighbor List, by knowing the neighbors of each element, for example: The neighbors of Element A in Parent 1 are (B, H), and the neighbors of Element A in Parent 2 are (B, G).

Now we delete the repetition (B, H) ∪ (B, G). The neighbors of "A" are (B, G, H). We do the same process for all the remaining elements as shown in the following figure.



We will demonstrate the idea of ER using flowchart. With the steps in the flowchart and with the help of the "Neighbor List" we will get a new child.



ER does not work well with the stopping condition A, because the similarity in all individuals of the population is difficult to achieve with the use of ER, and also does not give improvement in solutions well, and it difficult to apply compared to other crossover.

Position Crossover (PX):

PX was developed by Syswerda [23], date in 1991. Then was assessed by Barbulescu, her goal is to schedule problems by analyzing and inspecting the PX activity to see comparable operators. The position crossover (PX) technique is similar to the techniques in the (OX) and (OX2) crossover. How this crossover works by choosing various random locations along with parent1 strings and the rest from parent2 stings in the same order. For example, we assume we have to parents (parent1, parent2).

Parent1

A	B	C	D	E	F	G	H
1	2	3	4	5	6	7	8

Parent2

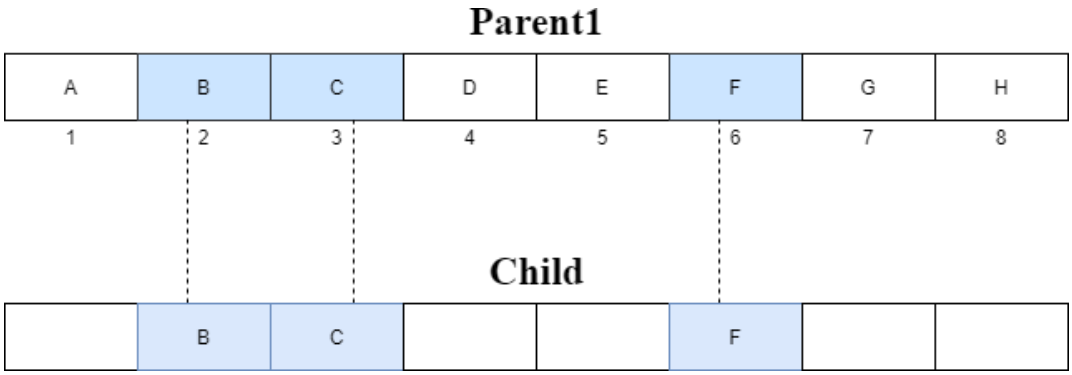
B	A	G	E	D	H	C	F
1	2	3	4	5	6	7	8

Locations will be chosen randomly in the number of locations and the position of them from parent1, as shown in the following figure.

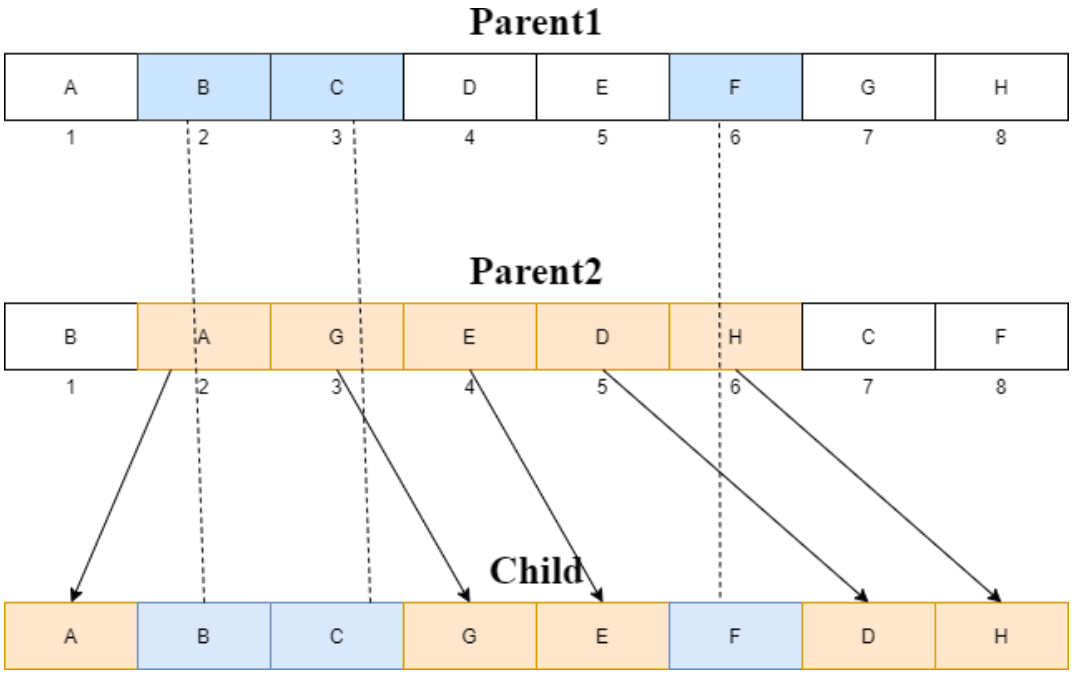
Parent1

A	B	C	D	E	F	G	H
1	2	3	4	5	6	7	8

This chosen locations is copied from parent1 to child at the same location, then the rest will be from parent2 at the same order in parent2 but not over the locations that have been copied and chosen parent1 also no duplicates from parent2, in below figure will show the child after copied from parent1.



The remains will be inherited from parent2 without duplicates like index 2,3,6 (B, C, F) in the empty locations index 1,4,5,7,8 in the child at the same order of its positions in parent2 as shown below in the figure.



The position crossover (PX) operator is basically a sort of permutation cross with a fix technique. It is nearer to the OX and OX2 operators where nodes are chosen in a randomly way.

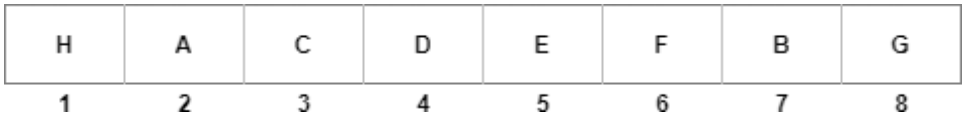
3.2.6 Mutation

Mutation operations are used to add a bit of variety in the population, by applying these operations on a certain chance (e.g. %0.05) the produced child will have his chromosomes changed and, in this problem, we used these mutation operations to change the permutation of the fragments of the child.

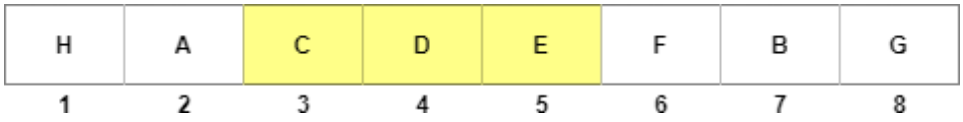
Inversion Mutation

Inversion Mutation works by choosing substring randomly, and then invert this substring [25].

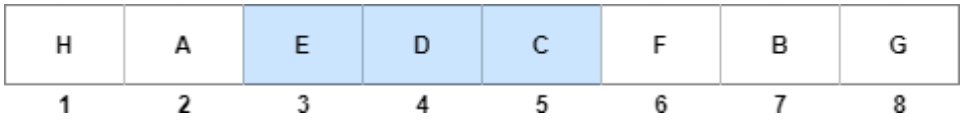
We assume we have the child in the following figure, before the mutation process:



We assume that the mutation occurred, and substring was randomly selected from index 3 to index 5 (C, D, E).



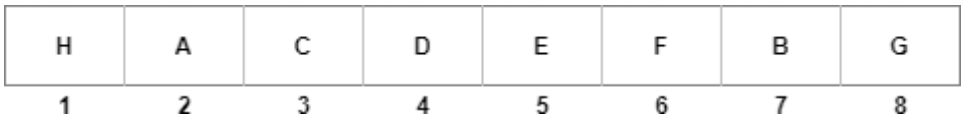
Now that the Inversion Mutation process has been completed, the child's shape after the Mutation will become like the following:



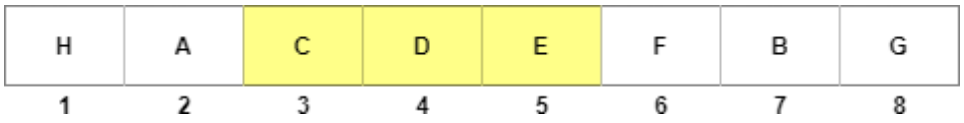
Displacement Mutation

Displacement Mutation works by choosing substring randomly, and then the substring is placed in a random place [25].

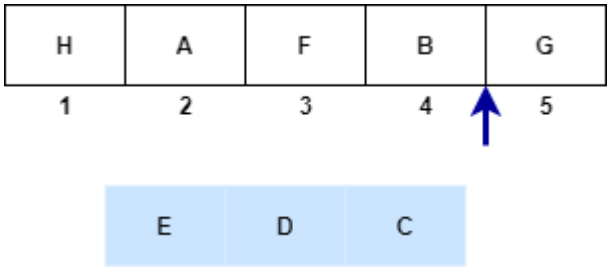
We assume we have the child in the following figure, before the mutation process:



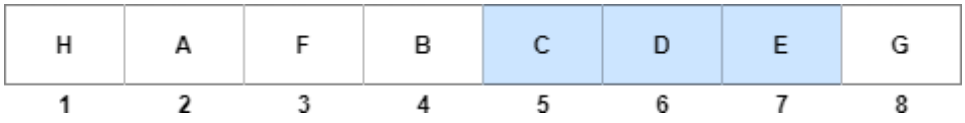
We assume that the mutation occurred, and substring was randomly selected from index 3 to index 5 (C, D, E).



following figure, the choice was made randomly between "B" and “G”.

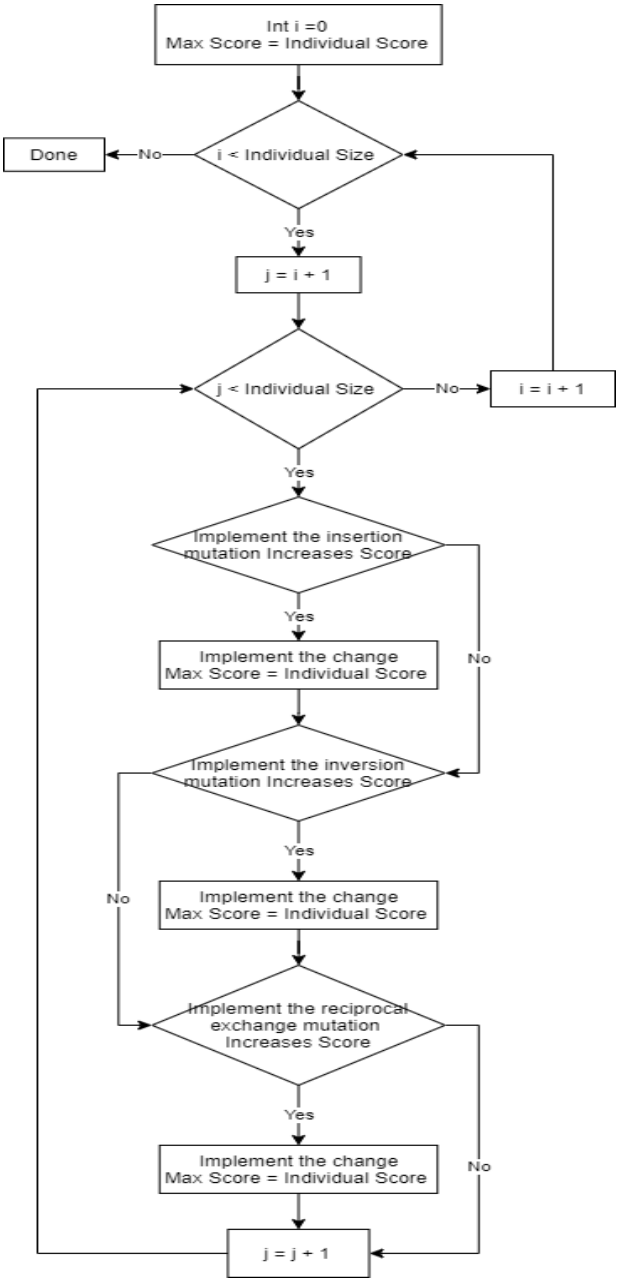


Now that the Displacement Mutation process has been completed, the child’s shape after the Mutation will become like the following:



Combined Mutation

Combined Mutation is implemented to improve the quality of the individual, and it combines three mutations (insertion, inversion, reciprocal exchange). In the following figure, flow chart shows how it works [24].

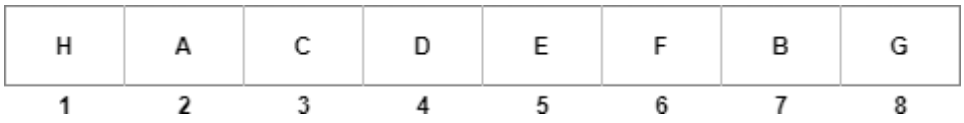


Through the tests we conducted, we noticed that Combined Mutation helps to find very excellent solutions, it is important to use them to improve the quality of the solutions.

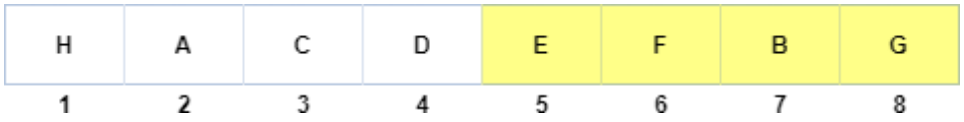
Scramble Mutation

Scramble Mutation works by choosing substring randomly, and then the substring is then randomly rearranged [26].

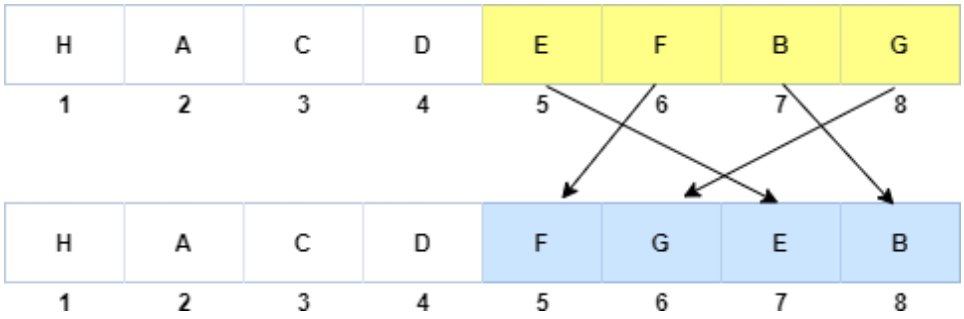
We assume we have the child in the following figure, before the mutation process:



We assume that the mutation occurred, and substring was randomly selected from index 5 to index 8 (E,F,B,G).



Now a random index is chosen for every element in substring, in our example, index 7 for element "E", "F" is selected in index 5. “B” in index 8 and “G” in index 6.



3.2.7 Stopping Condition

We have two stopping conditions Stop A and Stop B, in Stop B we record the fitness of the best individual in the population and we set a limit of 100 iterations of the genetic algorithm only if the fitness of the best individual in the new population is getting smaller consecutively, if the new individual is better than the last then we reset the limit count and continue otherwise we continue to 100 and stop. Stop A is to stop the algorithm if all the individuals in the generated population is the same since the probability of getting a new individual is extremely low.

3.3 Conclusion

We formalized and explained the existing solutions for the DNA fragments assembly problem. And our next step is to implement Genetic Algorithms used for this problem using Java programming language.

4. Results

4.1 Introduction

In this chapter, we will explain the method of the tests that we performed and review the results of these tests.

We will run tests on all crossovers, mutations, and all stop conditions that we have made, and these tests will run on many different data sets. We will also review the execution times of all the tests that they have conducted.

We ran all the tests with a Computer with an Intel i7-5500U 2.40 GHz processor, and with 8 GB RAM 1600 MHz

We took all the DNA sequences from the Gen bank [27]. Gen Bank is a bank that saves many of DNA sequences. It is available for all.

The dataset that we used in our project is:

- AF153612 [28]: Homo sapiens peroxisomal D3,D2-enoyl-CoA isomerase (PECI) mRNA, complete cds.
- BD393986 [29]: Novel Ecdysone Receptor-Based Inducible Gene Expression System.
- BX747047 [30]: XGC-gastrula Xenopus tropicalis cDNA clone TGas055e12 3', mRNA sequence.
- D11343 [31]: Rhizobium galegae gene for 16S rRNA, complete sequence, type strain: ATCC 43677.
- GU354832 [32]: Gelidocalamus sp. 2 Zeng & Zhang 06180 voucher Zeng and Zhang 06180 trnD-trnT intergenic spacer, partial sequence; chloroplast.
- JQ414563 [33]: Oryza granulata isolate CatB_mey_MAS3 catalase isozyme B (CatB) gene, partial cds.
- KM424533 [34]: Orthogonalys pulchella isolate 12538ort ultra conserved element locus uce-71 genomic sequence.
- KM424592 [35]: Orthogonalys pulchella isolate 12597ort ultra conserved element locus uce-3 genomic sequence.
- X60189 [36]: Human MHC class III region DNA with fibronectin type-III repeats.
- KX025132 [37]: Acipenser baerii HoxD10 (HoxD10) mRNA, complete cds.
-

4.2 Project Results

In this section we will explain the method of each test and present the results of these tests that we have performed. In each table, we will display the length of the DNA sequence for each dataset after implementing the genetic algorithm with the computational time in seconds. In the following table, we will display the name of each dataset with the length of each one before implementing the genetic algorithm.

Sequence name	AF153612	BD393986	BX747047	D11343	GU354832	JQ414563	KM424533	KM424592	KX025132	X60189
Sequence length	1348	925	867	556	1162	833	2782	1277	2094	3835

Table 1: Length of datasets

Test 1:

In this test (Table 2), we tested all crossovers, randomly selecting the mutation, making the algorithm work with stop condition A and making the population 10. We notice in this test that the best result came with the use of the SCX crossover, then it is followed by the PMX, CX and PX crossover in which the results are somewhat similar. Another option is the ER and OX2 crossover, and the results are also comparable.

					STOP A					
	AF153612	BD393986	BX747047	D11343	GU354832	JQ414563	KM424533	KM424592	KX025132	X60189
SCX	1065.5	750.5	684.5	454.9	944.4	695.2	2172	1015	1696.9	2880.7
ER	1215.4	862	798	494	1064.8	766.9	2555.5	1154.2	1924.2	3543.5
OX2	1240.8	847.1	793.2	492.6	1066	764.1	2587	1174.7	1935.9	3563.7
PMX	1136.9	765	729.3	443.9	956.8	683.9	2145	1019.7	1653.6	2914.3
CX	1159.9	800.3	712	467.8	1015	719.3	2249.4	1084.3	1780.1	3120.9
PX	1119.6	768.8	721.8	464.6	993.8	717.4	2407	1056.4	1743.4	3075.2

Table 2: Test 1 With Stop Cond. A

All numbers in the (Table 3) were calculated per second. We note that the SCX was not the best in terms of time as it was the best in terms of results. We note that the biggest time consumption came with the use of the ER crossover. The best results are in favor of PMX, CX and PX, so they are rather close.

					Time (s)					
	AF153612	BD393986	BX747047	D11343	GU354832	JQ414563	KM424533	KM424592	KX025132	X60189
SCX	44.13	24.92	35.17	12.67	38.77	27.38	122.3	1.96	5.27	39.05
ER	26.149	21.646	66.64	2.16	7.85	6.35	83.44	16.38	24.11	164.43
OX2	1.842	1.08	0.1	0.04	0.15	0.08	0.57	0.16	0.81	0.88
PMX	3.74	0.92	1.19	0.24	1.89	0.86	41.66	9.05	25.33	181.77
CX	2.68	4.23	0.88	0.16	4.96	0.42	90.57	2.51	9.4	85.52
PX	13.5	3.77	1.71	0.47	3.85	2.9	64.52	9.04	49.29	126.59

Table 3: Test 1 With Stop Cond. A

In this test (Table 4), we tested all crossovers, randomly selecting the mutation, making the algorithm work with stop condition B and making the population 10. We notice in this test a noticeable change from the previous test in the results and the best result came in the following table with the use of the SCX crossover, then it is followed by the PMX crossover and we notice the superiority of the PMX comes with the increase in the size of the data set. Then it comes in third place, as crossovers (CX and PX) together, because their results are similar. It comes last in ER and OX2.

					STOP B					
	AF153612	BD393986	BX747047	D11343	GU354832	JQ414563	KM424533	KM424592	KX025132	X60189
SCX	986	693.2	640.3	420.3	856.6	644.4	1943.5	943.2	1488.1	2534.1
ER	1020.8	708.5	659.3	426.5	880.6	655.6	2054.8	971.1	1563.5	2710.9
OX2	1028.9	713.2	666.7	429.6	891.9	660.3	2068.7	977.4	1570	2739.1
PMX	999.3	694.7	646.1	426.1	869.1	646.8	1971.66	950.2	1504.8	2588.8
CX	1000.6	696.1	657.1	425.1	874.6	648.1	1982.2	952.6	1512.4	2608
PX	1008.6	699	655.3	424.3	878.6	648.2	2014.6	959.5	1529.8	2655.9

Table 4: Test 1 With Stop Cond. B

In terms of time (Table 5), we notice a big change in time compared to the previous table, but this change came at the expense of the improvement in the results. As the dataset grows, the execution time increases significantly. As with the X60189 dataset, we note that the execution time is very long.

					Time (s)					
	AF153612	BD393986	BX747047	D11343	GU354832	JQ414563	KM424533	KM424592	KX025132	X60189
SCX	77.72	23.09	27.4	5.19	47.84	14.87	774.74	67.48	307.97	2034.58
ER	78.28	16.46	15.36	7.52	34.85	12.69	438.84	58.855	198.16	1276.48
OX2	49.72	16.15	13.53	4.21	39.68	6.31	415.24	68.24	172.62	661.58
PMX	41.4	16.98	15.87	4.56	30.04	12.71	766.31	49.78	282.64	2136.44
CX	58.48	19.49	13.99	4.49	57.61	16.37	702.03	61.56	1292.3	1517.37
PX	65.15	27.98	19.16	6.91	49.8	19.64	639.88	39.01	206.01	1396.56

Table 5: Test 1 With Stop Cond. B

Test 2:

In this test (Table 6), we tested all crossovers, randomly selecting the mutation, making the algorithm work with stop condition A and making the population 50. In this test we notice that with the condition of stopping "A" Crossover ER does not work and this is because the moment of reaching the point of similarity in all the individuals of the current generation is almost impossible even if all individuals, are generation of same parents. Because the rate of difference between one individual and the other is great, even if they are of the same parents. The difference may reach 60%. With this percentage, and with the population being 50, it will be difficult to reach a similar point. As for the results, they were good and close, except for the use of OX2.

					STOP A					
	AF153612	BD393986	BX747047	D11343	GU354832	JQ414563	KM424533	KM424592	KX025132	X60189
SCX	1018.6	695.6	658.3	422.6	859	651.3	2001.3	936.6	1500.6	2567.3
ER										
OX2	1205	810.6	795.6	472.6	1027.6	743.3	2513.3	1140.3	1891.3	3495.3
PMX	998.3	708	652.3	424.3	870	651.3	1978.6	952.3	1552	2590.6
CX	1082.6	721	655.3	436.3	905.3	654.6	2013	958	1564	2740
PX	1000.6	698.3	652	428.6	898.6	647.3	2008.6	953	1532	2593.3

Table 6: Test 2 With Stop Cond. A

And in terms of time (Table 7), we note that OX2 came first in terms of speed of implementation, while the rest of the crossovers were not the best in terms of implementation time, but they achieved good results.

					Time (s)					
	AF153612	BD393986	BX747047	D11343	GU354832	JQ414563	KM424533	KM424592	KX025132	X60189
SCX	23.72	18.25	7.62	3.41	19.74	7.91	224.37	21.46	59.55	744.73
ER										
OX2	8.56	1.94	2.432	2.0238	4.18	1.84	25.52	2.914	5.932	24.95
PMX	28.56	9.649	13.893	4.36	18.802	12.52	289.546	31.008	89.53	1028.727
CX	30.397	3.39	15.05	4.217	11.769	3.806	372.1968	17.928	67.86	1314.195
PX	44.884	19.99	7.767	2.06	12.61	8.24	351.021	46.033	121.207	1447.899

Table 7: Test 2 With Stop Cond. A

In this test (Table 8), we tested all crossovers, randomly selecting the mutation, making the algorithm work with stop condition B and making the population 10. In this test, SCX came first, then PMX came second. We note that results are close to CX, followed by PX in the last place, ER and OX2 come in close results.

					STOP B					
	AF153612	BD393986	BX747047	D11343	GU354832	JQ414563	KM424533	KM424592	KX025132	X60189
SCX	980	688.6	637	417	851.6	646	1940	933.6	1489.3	2527
ER	1005.3	698.3	654.3	426.3	866.3	648	2029.3	955	1545.3	2682.3
OX2	1021	695.3	650.6	422.3	883.3	649.3	2061.3	974.6	1555	2712.3
PMX	997.6	691	645.3	420.6	859	646.3	1964.6	940.6	1498.6	2575.3
CX	997.3	697.3	639.6	423.3	867.6	645.3	1971	947.6	1506	2598.3
PX	985	696.3	645.6	421.3	871.6	643	1964.6	948	1510.6	2590

Table 8: Test 2 With Stop Cond. B

In terms of time (Table 9), we notice that as the size of the dataset increases, the time increases dramatically. The largest number came with the largest data set, the X60189 and with the use of a PX crossover, as the test execution time took approximately 8463 seconds, which is a great time.

					Time (s)					
	AF153612	BD393986	BX747047	D11343	GU354832	JQ414563	KM424533	KM424592	KX025132	X60189
SCX	225.182	99.804	132.77	34.31	165.29	54.085	2816.902	196.889	821.797	4772.567
ER	279.658	70.14	85.167	23.576	187.161	65.11	2917.457	208.203	917.769	8028.452
OX2	218.536	101.433	85.55	25.22	140.301	79.72	2939.628	177.201	993.34	7938.509
PMX	332.28	167.423	73.624	21.86	298.41	75.56	2831.141	260.594	1094.717	5365.012
CX	238.49	80.674	83.009	31.86	155.37	57.315	2546.178	227.943	960.92	4235.232
PX	289.88	78.79	104.655	29.29	108.22	46.35	2950.415	205.92	994.187	8463.277

Table 9: Test 2 With Stop Cond. B

Test 3:

Here we tested all crossovers with stop condition A, using combined mutation, and chose the population 10. In this test (Table 10), we notice that the results are mixed, once the SCX is better and the time the PMX or CX, PX is better in terms of results. In the last place is OX2 and ER, and the results are close between them.

					STOP A					
	AF153612	BD393986	BX747047	D11343	GU354832	JQ414563	KM424533	KM424592	KX025132	X60189
SCX	1020.3	691	652	424	856.3	688.3	2016.3	949.3	1519.3	3099.6
ER	1216	855	793	483	1049.3	780	2553.3	1177	1926	3500
OX2	1237.6	834	770.3	491.3	1068	764.6	2581	1174.6	1948	3549.3
PMX	997.6	699.6	655.6	423.6	868.6	657.3	1977.3	955	1529	2569.3
CX	1006.6	725.6	664.6	446.6	944.3	652	2114.6	957	1673.3	2624.6
PX	1058.3	698.6	693	430.6	900.3	654.3	2202.6	962.3	1684.6	2997.6

Table 10: Test 3 With Stop Cond. A

In terms of time (Table 11), we noticed that the fastest implementation time came with the use of OX2. As for the rest of the crossovers, their results were somewhat similar.

					Time (s)					
	AF153612	BD393986	BX747047	D11343	GU354832	JQ414563	KM424533	KM424592	KX025132	X60189
SCX	21.32	9.84	13.61	3.19	8.9	2.8	246.37	19.38	114.04	361.5
ER	56.95	11.85	2.75	6.34	27.54	8.68	196.11	48.17	45.35	1057.81
OX2	0.57	1.18	0.95	0.26	1.18	0.82	10.96	0.46	0.82	25.88
PMX	63.02	12.54	8.18	2.31	19.01	7.74	583.33	38.09	157.36	2481.07
CX	28.44	7.69	8.15	1.07	5.87	11.79	261.64	22.06	84.71	551.46
PX	303.13	20.14	12.97	1.68	118.72	14.6	4815.39	168.2	683.04	5674.83

Table 11: Test 3 With Stop Cond. A

Here we tested all crossovers with stop condition B, using combined mutation, and chose the population 10. In this test (Table 12), we note that the results are excellent compared to previous tests, and with the use of SCX, we obtained the best results in this test. PMX also achieved excellent results after SCX. As for the rest of the crossovers, they are close to the results except for ER and OX2.

					STOP B					
	AF153612	BD393986	BX747047	D11343	GU354832	JQ414563	KM424533	KM424592	KX025132	X60189
SCX	979.3	689.6	644.3	422.3	865.3	644	1928.3	939.6	1488.6	2511.6
ER	1018.6	703.6	658.3	422.3	876	647.6	2033.3	959.6	1546	2682
OX2	1020.3	707	659.6	428.6	891.3	656.6	2055.3	696	1564	2726.3
PMX	993.6	697.3	649	425.6	876.6	649.6	1973.3	949.3	1502.3	2585
CX	1003	694.6	662	417.3	877.3	650.6	1995.6	943	1520.3	2616
PX	994.3	698	650.6	425.3	861	648.3	2004.6	958	1521	2650.3

Table 12: Test 3 With Stop Cond. B

In terms of time (Table 13), we noticed that the execution time increased, specifically with the use of SCX, as we have reached the highest execution time as it took with the largest dataset to 14134s. This is a very high time.

					Time (s)					
	AF153612	BD393986	BX747047	D11343	GU354832	JQ414563	KM424533	KM424592	KX025132	X60189
SCX	390.42	49.58	71.48	12.44	83.53	57.26	3068.68	193.54	1211.15	14134.25
ER	268.98	91.94	106.72	35.41	178.05	106.56	4604.02	163.31	1795.33	9604.46
OX2	328.02	90.3	103.06	14.86	164.01	29.12	4550.2	272.61	1389.73	7067.66
PMX	344.65	70.04	68.19	12.58	119.93	25.84	3997.97	212.21	1169.41	6374.3
CX	229.93	48.62	61.2	16.62	48.32	52.62	3175.58	150.59	970.7	6084.87
PX	323.64	74.3	87.52	10.1	154.26	73.75	3955.01	212.66	1543.08	8941.38

Table 13: Test 3 With Stop Cond. B

Test 4:

In (Table 14), we used only the displacement mutation with the six crossovers on a population of 10 individuals with stopping condition A. SCX has performed the best in terms of the length of resulted DNA sequence, with all rest performing somewhat similar giving the result of each one and CX being the worst.

					STOP A					
	AF153612	BD393986	BX747047	D11343	GU354832	JQ414563	KM424533	KM424592	KX025132	X60189
SCX	1148.3	811.6	746.6	471.6	993.3	732.6	2363.6	1095	1791.6	3207
ER	1239.3	857.6	790.3	505.3	1089	760.6	2604	1161.3	1923.6	3534.6
OX2	1247.3	842.6	786.6	493	1064	768.6	2590.6	1178.6	1943	3566
PMX	1235.3	846.6	780.3	496.3	1065	752	2617	1180	1929.6	3580.3
CX	1272	864	813.3	501	1096.6	780.3	2634.6	1204	1981	3613.3
PX	1253	851.3	787	493	1067	761.3	2579	1188.3	1932.3	3589

Table 14: Test 4 With Stop Cond. A

And in terms of time (Table 15), taken to find the sequence SCX did not achieve the best times in all datasets being similar to OX2, PMX and PX, giving that it gave better solutions. CX did achieve the best times beating SCX sacrificing the quality of the final sequence. Lastly, ER did the worst and by a big difference in relation of the other crossovers.

					Time (s)					
	AF153612	BD393986	BX747047	D11343	GU354832	JQ414563	KM424533	KM424592	KX025132	X60189
SCX	0.31	0.08	0.09	0.09	0.18	0.17	0.91	0.23	0.46	2.7
ER	6.5	15.23	9.16	2.2	15.12	3.56	42.03	7.56	42.21	147.5
OX2	0.27	0.1	0.13	0.04	0.13	0.12	0.53	0.16	0.36	0.78
PMX	0.28	0.07	0.09	0.041	0.17	0.111	0.19	0.17	0.51	0.50
CX	0.09	0.06	0.05	0.04	0.03	0.05	0.26	0.0902	0.15	0.18
PX	0.54	0.21	0.16	0.1	0.45	0.3	4.2	0.39	2.53	2.67

Table 15: Test 4 With Stop Cond. A

In (Table 16), we used only the displacement mutation with the six crossovers on a population of 10 individuals with stopping condition B. Both SCX and PMX did the best overall performance in terms of DNA sequence length. OX2 and ER did the worst since both of them do not diversify the generated population by much like the other crossovers.

					STOP B					
	AF153612	BD393986	BX747047	D11343	GU354832	JQ414563	KM424533	KM424592	KX025132	X60189
SCX	1138.3	783	734	462	997.3	720.3	2347	1064	1776	3183.6
ER	1231	825.6	786	499.3	1067.3	765	2572	1903.3	1917	3524
OX2	1235	844.3	779	489.3	1063.3	765.3	2577	1903.3	1929	3564.3
PMX	1136.6	782.3	723.3	455	976.3	707.3	2358	1075	1786.66	3258
CX	1179	787.3	764.6	477.6	1010.6	724.6	2543.33	1144.3	1816	3315.6
PX	1215.6	820.6	772	475.6	1041	737.6	2551.6	1158	1911.66	3527.3

Table 16: Test 4 With Stop Cond. B

And in terms of time (Table 17), OX2 did the best for the same reason it is the worst in terms of DNA sequence. And both SCX and PX are somewhat similar. ER, PMX and CX did the worst on this test.

					Time (s)					
	AF153612	BD393986	BX747047	D11343	GU354832	JQ414563	KM424533	KM424592	KX025132	X60189
SCX	2.27	1.545	1.12	1.39	2.85	1.37	9.093	1.84	5.62	12.07
ER	6.38	2.97	1.29	0.60	3.53	1.59	27.56	2.86	13.32	40.97
OX2	1.37	0.72	0.63	0.47	0.95	0.62	4.48	1.441	2.73	4.97
PMX	5.72	2.70	2.86	1.43	4.55	2.59	22.67	6.005	10.02	29.42
CX	6.55	5.71	2.85	1.35	6.34	4.78	16.16	4.8	21.29	84.4
PX	2.90	2.41	1.49	1.16	2.96	2.53	11.93	3.23	5.066	14.91

Table 17: Test 4 With Stop Cond. B

Test 5:

In (Table 18), we used only the inversion mutation with the six crossovers on a population of 10 individuals with stopping condition A. SCX performed the best in terms of DNA sequence length, while ER and CX performed the worst.

					STOP A					
	AF153612	BD393986	BX747047	D11343	GU354832	JQ414563	KM424533	KM424592	KX025132	X60189
SCX	1146.6	781.6	744.6	472	991.6	714	2400	1106	1751.6	3259.6
ER	1267.6	871.3	822.6	508.6	1074.3	771.3	2599.3	1189.3	1982	3609
OX2	1242.6	849	778.6	491.6	1073	762	2599.33	1180	1927.6	3561.6
PMX	1258	851.6	795.3	491.6	1081	764.3	2617.3	1191.3	1961.3	3599
CX	1268.6	867	805.6	512.6	1092	779.6	2627.6	1204	1968	3613.3
PX	1251	847	786.3	485	1059.6	755.6	2605.3	1180	1940.3	3572

Table 18: Test 5 With Stop Cond. A

In terms of time (Table 19), CX performed the best while ER performed the worst and the rest performed somewhat similar.

					Time (s)					
	AF153612	BD393986	BX747047	D11343	GU354832	JQ414563	KM424533	KM424592	KX025132	X60189
SCX	0.38	0.171	0.081	0.054	0.286	0.229	1.141	0.209	1.117	2.828
ER	18.51	27.34	9.018	2.587	10.14	9.324	144.48	13.996	13.28	56.76
OX2	0.282	0.112	0.13	0.048	0.16	0.128	0.54	0.17	0.37	0.85
PMX	0.15	0.102	0.05	0.055	0.062	0.073	0.21	0.068	0.15	0.28
CX	0.135	0.06	0.052	0.02	0.034	0.0234	0.208	0.074	0.09	0.1257
PX	0.223	0.194	0.118	0.077	0.319	0.138	1.16	0.197	0.501	2.58

Table 19: Test 5 With Stop Cond. A

In (Table 20), we used only the inversion mutation with the six crossovers on a population of 10 individuals with stopping condition B. SCX performed the best in terms of DNA sequence length while the rest performed somewhat similar.

					STOP B					
	AF153612	BD393986	BX747047	D11343	GU354832	JQ414563	KM424533	KM424592	KX025132	X60189
SCX	1079.3	751.3	697	442.6	946.3	695	2301.3	1044.3	1690	3063
ER	1249	851.6	789.3	502.6	1066.6	770.6	2579.6	1182.6	1940	3562.6
OX2	1226.3	835	787.6	487.3	1064.6	766.6	2576.66	1174.3	1934.6	3565.6
PMX	1214	818.3	773.33	484.3	1049.3	747	2569.3	1175	1919.3	3548.6
CX	1249.3	857.6	803	500.3	1084	769.6	2615.3	1196.6	1949	3601
PX	1207.33	824	765.6	474.3	1038	750.6	2552.3	1150	1901	3528.3

Table 20: Test 5 With Stop Cond. B

In terms of time (Table 21), SCX performed the worst in all datasets, while the rest performed somewhat similar except for dataset X60189.

					Time (s)					
	AF153612	BD393986	BX747047	D11343	GU354832	JQ414563	KM424533	KM424592	KX025132	X60189
SCX	16.04	4.61	4.39	1.78	6.59	2.38	26.4	6.8	31.84	98.95
ER	2.29	1.71	1.92	0.72	2.91	1.37	11.83	1.91	4.23	18.78
OX2	2.62	1.21	1.017	0.51	1.092	0.92	5.056	1.065	2.66	3.95
PMX	4.76	2.004	2.43	1.101	3.29	1.6055	8.74	2.29	5.408	9.059
CX	3.41	1.115	1.443	0.651	1.58	1.35	5.33	2.055	3.19	6.344
PX	4.66	1.65	2.28	1.53	3.48	1.39	9.868	2.87	7.772	19.006

Table 21: Test 5 With Stop Cond. B

Test 6:

In (Table 22), we used only the scramble mutation with the six crossovers on a population of 10 individuals with stopping condition A. The best result was by using the SCX, on the other hand all other crossovers ER, OX2, PMX, CX and PX had similar results in some datasets and little different in other datasets but in the end it is comparable.

					STOP A					
	AF153612	BD393986	BX747047	D11343	GU354832	JQ414563	KM424533	KM424592	KX025132	X60189
SCX	1149.6	787	730.3	459	1001.3	726.3	2368	1097.3	1776.6	3237.6
ER	1254.6	851.6	811.6	501	1070	771.3	2618.6	1204.6	1965.6	3595.6
OX2	1241	846.3	786.3	487.6	1064	763	2592	1169.6	1934.6	3567.6
PMX	1253	855.6	793.6	494.6	1072	769.6	2609.6	1195.3	1953.3	3594.3
CX	1267	865.6	811.3	503.6	1092	786	2621.3	1204.6	1974.6	3614.6
PX	1244.3	848.6	791	501.3	1071	769	2601.33	1186	1938.6	3577.3

Table 22: Test 6 With Stop Cond. A

In terms of time (Table 23), we noticed that CX was the best in time in almost every datasets we tested, on the other hand ER was the slowest by high difference in all datasets compared to other crossovers, the best time was CX, and it is somewhat similar between the SCX, OX2, PMX and PX.

					Time (s)					
	AF153612	BD393986	BX747047	D11343	GU354832	JQ414563	KM424533	KM424592	KX025132	X60189
SCX	0.215	0.13	0.072	0.0543	0.164	0.067	0.979	0.11	0.707	1.76
ER	71.5	20.46	21.5	3.46	15.42	11.92	94.4	23.94	56.22	362.83
OX2	0.276	0.09	0.107	0.062	0.147	0.131	0.65	0.14	0.434	0.762
PMX	0.163	0.059	0.058	0.0501	0.13	0.054	0.157	0.075	0.187	0.184
CX	0.09	0.021	0.069	0.026	0.471	0.033	0.26	0.069	0.119	0.274
PX	0.404	0.067	0.14	0.065	0.161	0.059	0.661	0.194	0.538	1.215

Table 23: Test 6 With Stop Cond. A

In (Table 24), we used only the scramble mutation with the six crossovers on a population of 10 individuals with stopping condition B. The best result was by using SCX crossover, and the worst result was CX but not by much compared to OX2, and all the other are somewhat similar.

					STOP B					
	AF153612	BD393986	BX747047	D11343	GU354832	JQ414563	KM424533	KM424592	KX025132	X60189
SCX	1086.3	755	694	442	944.3	688.33	2251.3	1056.3	1667.3	3041.3
ER	1232	833	782	487	1064.6	759	2577	1168.33	1918.6	3540
OX2	1237	844.6	788.3	491.6	1058	770.3	2580.6	1162.3	1926.6	3564.6
PMX	1213.6	813.66	773.3	471.6	1043.3	746.3	2560.6	1141.6	1916.6	3521.66
CX	1250.3	849.3	783.6	495	1068.6	768	2599.6	1181.6	1949.6	3584.6
PX	1219	822	757.6	474.3	1045.3	746.6	2556.6	1153	1896.3	3526

Table 24: Test 6 With Stop Cond. B

In terms of time (Table 25), We noticed a change compared to the previous test, we also notice OX2 was the best in time in almost every datasets, beside the last dataset the (X60189) the PX was better, on the other hand the SCX crossover was not good, then the other crossovers ER, PMX, CX and PX are somewhat similar.

					Time (s)					
	AF153612	BD393986	BX747047	D11343	GU354832	JQ414563	KM424533	KM424592	KX025132	X60189
SCX	7.585	2.518	2.844	1.644	5.45	2.40058	24.306	4.29	18.307	49.96
ER	2.54	1.54	1.369	0.544	2.16	0.852	6.34	2.98	5.48	15.96
OX2	0.861	0.48	0.52	0.26	0.62	0.63	2.3	1.13	1.818	2.911
PMX	2.102	1.92	1.0436	0.58	1.79	1.116	4.638	3.069	3.540	7.307
CX	1.095	1.129	1.48	0.6085	1.461	0.843	4.46	1.92	2.31	6.143
PX	1.854	1.53	1.37	0.66	1.809	0.909	4.87	2.439	5.699	10.82

Table 25: Test 6 With Stop Cond. B

Test 7:

In (Table 26), we used only the swap mutation with the six crossovers on a population of 10 individuals with stopping condition A. The best result was by using the SCX crossover in all used datasets, then comes the two crossovers ER and OX2 was very similar in almost every dataset we used, on the other hand the CX was not good at this test, and the PMX and PX were in the middle.

					STOP A					
	AF153612	BD393986	BX747047	D11343	GU354832	JQ414563	KM424533	KM424592	KX025132	X60189
SCX	1150	797	743	476.6	995.6	722.6	2422.6	1093.3	1738.6	3243.6
ER	1219.3	849.3	785.6	501.6	1089.6	762.6	2559	1181.6	1946	3612
OX2	1235.6	849.6	797	488.6	1059.6	769	2589	1184	1935.3	3562.3
PMX	1259.3	855.6	800	494.6	1082.3	768.6	2614.3	1201	1950	3589
CX	1266.3	872	807.3	508	1090	784	2629.6	1203.6	1965.6	3635.3
PX	1238	848.6	781	494	1071	759	2617.3	1189.3	1939.3	3565

Table 26: Test 7 With Stop Cond. A

In terms of time (Table 27), we noticed that the CX crossover was faster in many datasets we used and the slowest was the ER crossover by high margin between all other crossovers.

					Time (s)					
	AF153612	BD393986	BX747047	D11343	GU354832	JQ414563	KM424533	KM424592	KX025132	X60189
SCX	0.302	0.09	0.092	0.046	0.183	0.11	0.508	0.16	0.62	1.609
ER	17.88	6.65	12.1	3.36	8.073	4.011	57.709	16.3	25.68	87.58
OX2	0.18	0.11	0.08	0.053	0.123	0.086	0.5564	0.188	0.35	0.834
PMX	0.07	0.048	0.04	0.033	0.073	0.08	0.184	0.071	0.16	0.351
CX	0.04	0.033	0.041	0.013	0.0301	0.039	0.162	0.089	0.178	0.115
PX	0.21	0.0913	0.177	0.076	0.236	0.121	0.48	0.154	0.572	1.6

Table 27: Test 7 With Stop Cond. A

In (Table 28), we used only the swap mutation with the six crossovers on a population of 10 individuals with stopping condition B. the best result was by using the SCX crossover in all datasets we used by a visible difference, and the PMX, CX and PX were pretty close and comparable but butter than ER and OX2.

					STOP B					
	AF153612	BD393986	BX747047	D11343	GU354832	JQ414563	KM424533	KM424592	KX025132	X60189
SCX	1125	782.3	720	458.6	974.6	717.3	2326.6	1086.6	1784.6	3145.6
ER	1232	852.6	784	481	1059.3	755.3	2591	1169.3	1937.3	3543.6
OX2	1228.6	837	788.6	489.6	1059	760.3	2601	1167.6	1941.6	3547.3
PMX	1180.6	793.6	751.6	472.3	1016.6	733	2456.6	1119	1831.6	3304
CX	1174	800	765.6	470	1019.3	743.6	2469.3	1114	1856.6	3394.6
PX	1187	790.6	746.3	469.3	1015.3	720.3	2522.6	1141.3	1867.6	3423.6

Table 28: Test 8 With Stop Cond. B

In terms of time (Table 29). We notice that a very big difference in time by using the OX2 crossover the fastest in this test not comparable or close to other crossovers in time, on the other hand the slowest in this test was the CX and PX in some datasets, PMX was not consistent compared to other crossovers.

					Time (s)					
	AF153612	BD393986	BX747047	D11343	GU354832	JQ414563	KM424533	KM424592	KX025132	X60189
SCX	2.886	1.59	2.42	0.68	3.65	1.5	5.68	2.24	3.909	13.65
ER	3.39	1.12	1.56	0.7	2.48	2.34	8.538	3.054	2.91	15.8
OX2	0.84	0.81	0.59	0.345	1.048	0.48	1.95	0.961	1.42	2.97
PMX	4.754	2.63	2.203	0.82	4.038	2.41	16.8	4.193	13.2	44.63
CX	7.247	3.34	2.13	1.279	4.87	2.13	22.82	6.217	11.03	40.451
PX	3.69	3.55	02.07	1.207	4.21	2.743	10.919	2.65	7.8	31.226

Table 29: Test 8 With Stop Cond. B

4.3 Conclusion

In this chapter, we explained the test methods and review the results of these tests. We also made all possible tests to make the comparison process between the results easier and clearer.

In the next chapter, we will discuss the tests that we did, discuss the results that we got, and compare these results to reach the best possible way to solve the problem.

5. Discussion

5.1 Introduction

As we saw in the previous chapter, we did many tests on the genetic algorithm on all crossovers, mutations and stopping conditions that we made.
Given that the problem of assembling DNA fragments is very complex, it is difficult to find an optimal solution. Therefore, we chose the genetic algorithm, designed and implemented many crossovers, mutations and stopping conditions to reach the best possible solution. We have also selected many different data sets for testing.
In this chapter, we will explain and discuss all the results of the tests we performed and how we came to the best possible result.

5.2 Discussion of the Results

Initially, we tested all crossovers and stopping conditions with randomly selecting mutations and chose the population 10.
In this test, we note that the results of stop condition "B" are better than those of condition "A". But stop condition "B" consumes a lot of time to show results opposite to stop condition "A". Also, the best results were when using the SCX crossover with a stop condition "B", with a noticeable difference from the rest. As for time, all times were close.
In the second test, we repeated the first test with the change of population to 50. In this test we notice that with the condition of stopping "A" Crossover ER does not work and this is because the moment of reaching the point of similarity in all the individuals of the current generation is almost impossible even if all individuals, are generation of same parents.
Because the rate of difference between one individual and the other is great, even if they are of the same parents. The difference may reach 60%. With this percentage, and with the population being 50, it will be difficult to reach a similar point.
We also note that the speed of time is in favor of the stop condition "A", but in terms of the results it is superior to the stop condition "B". We also note that the results of the second test are better than the first test.
In the third test, we tested all crossovers with all stop conditions with the use of combined mutation and we chose the population 10.
With the use of the combined mutation, the time taken to generate results increases. But this increase comes at the cost of good results. With comparison with the second test, the best results are fairly close.
Good results were seen with the use of the SCX Crossover with "B" stop condition.
In the fourth test we repeated the third test but with the change of the type of mutation to displacement mutation. Here we noticed a big change in results and time. In terms of results, the results were not the best we got compared with the first test, the second test, and the third test, and the best results came in the stop condition (B). But in terms of time, we notice a big change, and the results are appearing faster than previous tests.

In the fifth test, we tested all crossovers with all stop conditions with the use of the inversion mutation and we chose the population 10. We did not get better results than the previous tests. As we noticed in the fourth test, there is an observed change in time and results. Likewise, the preference of results goes with the condition of stopping "B". As for the time preference, it goes with the stop condition "A".
In the sixth test, we tested all crossovers and stop conditions with a scramble mutation and the population was set to 10. We note that the results are close to the fourth and fifth test, and we did not get a better result than the previous tests. Also, with repeating the same test with the change of the type of mutation into a swap mutation we will observe the speed of execution of the algorithm with stop condition "A". But in terms of results, there is no improvement over previous tests.

5.3 Conclusion

After performing all these tests and testing all possibilities. We note that the best results came with the second and third tests, namely with the use of SCX Crossover with the stop condition "B".
The results were almost close between the two tests. As for the fourth, fifth, sixth and seventh tests, they did not give us good results, but the execution time in them was fast. Thus, we conclude that using the SCX crossover without a combined mutation is not sufficient to give good results. We also conclude that with the use of the stop condition "B", it improves the results, but it takes longer time than the condition "A". PMX Crossover comes second after SCX crossover, which gave the best results in the second test with the use of the stop condition "B". Next, the PX crossover also gave good results in the second test with a "B" stopping condition.
In the end, we can say that using a SCX crossover with combined mutation and with a stop condition "B" and making the population 50, we will get very excellent results. But these excellent results will come at the expense of time. The results of the best test are shown in the following table.

					STOP B					
	AF153612	BD393986	BX747047	D11343	GU354832	JQ414563	KM424533	KM424592	KX025132	X60189
SCX	980	688.6	637	417	851.6	646	1940	933.6	1489.3	2527
					Time (s)					
	AF153612	BD393986	BX747047	D11343	GU354832	JQ414563	KM424533	KM424592	KX025132	X60189
SCX	225.182	99.804	132.77	34.31	165.29	54.085	2816.902	196.889	821.797	4772.567

6. Conclusion

We discussed in chapter 1 the idea of DNA fragment assembly and the obstacles that made solving this problem by hand a difficult task, and how using Genetic Algorithm would speed up the process and give better results. And in chapter 2, we reviewed the literature and the contributed work that solves this problem using different algorithms other than the Genetic Algorithm and we mentioned the applications of this problem and how important it is. In chapter 3 we discussed the design and implementation of the existing solutions and the design of the Genetic Algorithm and each stage in it (e.g. Selection and Crossover). In chapter 4 and 5 we presented the comprehensive tests on all the algorithms we implemented by measuring the length of the final sequence and the time it took to find that sequence, and the discussion about the results with a comparison between the implemented algorithms and giving the best variation of the Genetic Algorithm.

This research should provide the necessary information about the Genetic Algorithm and how it solves the DNA fragment assembly, and the various crossover and mutation algorithms with the comprehensive tests to give a better idea of the performance of each algorithm.

The main achievements of this research are to give a comprehensive study about the DNA fragment assembly problem and the implementation of the Genetic Algorithm and how it used to tackle this problem.
This work could be improved with applying local search algorithms to create a hybrid algorithm where the final sequence would be used as an input to the second algorithm.

References

[1] G. Luque and E. Alba, "Metaheuristics for the DNA Fragment Assembly Problem", International Journal of Computational Intelligence Research, vol. 1, no. 2, pp. 98-108, 2005.

[2] A. Motahari, G. Bresler and D. Tse, "Information Theory of DNA Shotgun Sequencing", IEEE Transactions on Information Theory, vol. 59, no. 10, pp. 6273 - 6289, 2013.

[3] A. Osman, A. A. Alwawi and O. M. A. Abdallah, "Using DNA sequencing in diagnosis of hereditary diseases in Al-Qunfudah, KSA," 2017 Joint International Conference on Information and Communication Technologies for Education and Training and International Conference on Computing in Arabic (ICCA-TICET), Khartoum, 2017, pp. 1-5.

[4] J. Prahlow, T. Cameron and A. Arendt, "DNA testing in homicide investigations", Medicine, Science and the Law, vol. 57, no. 4, pp. 179-191, 2017. Available: 10.1177/0025802417721790.

[5] N. DJ, "Paternity tests", 2019.

[6] "DNA sequencing", En.wikipedia.org, 2020. [Online]. Available: https://en.wikipedia.org/w/index.php?title=DNA_sequencing&oldid=922319768.

[7] (NIH) National Human Genome Research Institute www.genome.gov › genetics-glossary › acgt

[8] L. Li and S. Khuri, "A Comparison of DNA Fragment Assembly Algorithms", 2004.

[9] K. Huang, J. Chen, C. Yang and C. Tsai, "A memetic gravitation search algorithm for solving DNA fragment assembly problems", Journal of Intelligent & Fuzzy Systems, vol. 30, no. 4, pp. 2245-2255, 2016. Available: 10.3233/ifs-151994.

[10] J. Blazewicz, E. Burke, G. Kendall, W. Mruczkiewicz, C. Oguz and A. Swiercz, "A hyper-heuristic approach to sequencing by hybridization of DNA sequences", Annals of Operations Research, vol. 207, no. 1, pp. 27-41, 2011. Available: 10.1007/s10479-011-0927-y.

[11] http://www.phrap.org/.

[12] G.G. Sutton, O. White, M.D. Adams, and A.R. Kerlavage. TIGR Assembler: A new tool for assembling large shotgun sequencing projects, Genome Science & Tech., pp.9–19,1995.

[13] T. Chen and S.S. Skiena. Trie-based data structures for sequence assembly. In The Eighth Symposium on Combinatorial Pattern Matching, pp. 206–223, 1998.

[14] X. Huang and A. Madan. CAP3: A DNA sequence assembly program, Genome Research, 9, pp. 868–877, 1999.

[15] E.W. Myers. Towards simplifying and accurately formulating fragment assembly, Journal of Computational Biology, 2(2), pp. 275–290, 2000.

[16] P.A. Pevzner. Computational molecular biology: An algorithmic approach. The MIT Press, London, 2000.

[17] L. Li and S. Khuri, "A Comparison of DNA Fragment Assembly Algorithms", 2004.
[18] S.Kirkpatrick, C.D.Gelatt, andM.P.Vecchi.Optimization by simulated annealing, Science, 220(4598), pp. 671–680, 1983.
[19] H. Bennaceur and E. Alanzi, "Genetic Algorithm For The Travelling Salesman Problem using Enhanced Sequential Constructive Crossover Operator", International Journal of Computer Science and Security (IJCSS), vol. 11, no. 3, 2017.
[20] L. Wang, J. Zhang and H. Li, "An Improved Genetic Algorithm for TSP – IEEE Conference Publication", Ieeexplore.ieee.org, 2007. [Online]. Available:https://ieeexplore.ieee.org/document/4370274/.
[21] H. Azarbonyad and R. Babazadeh, "A Genetic Algorithm for solving Quadratic Assignment Problem (QAP)", Arxiv.org, 2014. [Online]. Available:https://arxiv.org/pdf/1405.5050.
[22] P. Larrañaga, C. Kuijpers, R. Murga, I. Inza and S. Dizdarevic, Artificial Intelligence Review, vol. 13, no. 2, pp. 129-170, 1999. Available: 10.1023/a:1006529012972.
[23] U. A.J. and S. P.D., "CROSSOVER OPERATORS IN GENETIC ALGORITHMS: A REVIEW", ICTACT Journal on Soft Computing, vol. 06, no. 01, pp. 1083-1092, 2015. Available: 10.21917/ijsc.2015.0150.
[24] Z. Ahmed, "A hybrid algorithm combining lexisearch and genetic algorithms for the quadratic assignment problem", Cogent Engineering, vol. 5, no. 1, 2018. Available: 10.1080/23311916.2018.1423743.
[25] K. Deep and H. Mebrahtu, "Combined Mutation Operators of Genetic Algorithm for the Travelling Salesman Problem", International Journal of Combinatorial Optimization Problems and Informatics, vol. 2, no. 3, pp. 1-23, 2011.
[26] "Study of Various Mutation Operators in Genetic Algorithms", International Journal of Computer Science and Information Technologies, 2014.
[27] https://www.ncbi.nlm.nih.gov/genbank/
[28] B. Geisbrecht, D. Zhang, H. Schulz and S. Gould, "Characterization of PECI, a Novel Monofunctional Δ3,Δ2-Enoyl-CoA Isomerase of Mammalian Peroxisomes", Journal of Biological Chemistry, vol. 274, no. 31, pp. 21797-21803, 1999. Available: 10.1074/jbc.274.31.21797.
[29] PALLI SUBBA REDDY, KAPITSKAYA MARIANNA ZINOVJEVNA, CRESS DEAN ERVIN. NOVEL ECDYSONE RECEPTOR-BASED INDUCIBLE GENE EXPRESSION SYSTEM. World Intellectual Property Organization Publ.of the Int.Appl. without Int.search REP. WO2001US09050. 21 Mar 2001.
[30] C. Showell and F. Conlon, "Decoding development inXenopus tropicalis", genesis, vol. 45, no. 6, pp. 418-426, 2007. Available: 10.1002/dvg.20286.
[31] K. Sano and M. Miyata, "The gyrB gene lies opposite from the replication origin on the circular chromosome of Mycoplasma capricolum", Gene, vol. 151, no. 1-2, pp. 181-183, 1994. Available: 10.1016/0378-1119(94)90653-x.
[32] C. Zeng, Y. Zhang, J. Triplett, J. Yang and D. Li, "Large multi-locus plastid phylogeny of the tribe Arundinarieae (Poaceae: Bambusoideae) reveals ten major

lineages and low rate of molecular divergence", Molecular Phylogenetics and Evolution, vol. 56, no. 2, pp. 821-839, 2010. Available: 10.1016/j.ympev.2010.03.041.

[33] B. Ai, Z. Wang and S. Ge, "GENOME SIZE IS NOT CORRELATED WITH EFFECTIVE POPULATION SIZE IN THEORYZASPECIES", Evolution, vol. 66, no. 10, pp. 3302-3310, 2012. Available: 10.1111/j.1558-5646.2012.01674.x.

[34] B. Faircloth, M. Branstetter, N. White and S. Brady, "Target enrichment of ultraconserved elements from arthropods provides a genomic perspective on relationships among H ymenoptera", *Molecular Ecology Resources*, vol. 15, no. 3, pp. 489-501, 2014. Available: 10.1111/1755-0998.12328.

[35] K. Matsumoto, M. Arai, N. Ishihara, A. Ando, H. Inoko and T. Ikemura, "Cluster of fibronectin type III repeats found in the human major histocompatibility complex class III region shows the highest homology with the repeats in an extracellular matrix protein, tenascin", Genomics, vol. 12, no. 3, pp. 485-491, 1992. Available: 10.1016/0888-7543(92)90438-x.

[36] D. Langenau et al., "Molecular cloning and developmental expression of Tlx (Hox11) genes in zebrafish (Danio rerio)", Mechanisms of Development, vol. 117, no. 1-2, pp. 243-248, 2002. Available: 10.1016/s0925-4773(02)00187-9.

[37] "File:Difference DNA RNA-EN.svg - Wikimedia Commons", Commons.wikimedia.org, 2020. [Online]. Available: https://commons.wikimedia.org/wiki/File:Difference_DNA_RNA-EN.svg.

[38] "NP-hardness", En.wikipedia.org, 2020. [Online]. Available: Wikipedia,https://en.m.wikipedia.org/wiki/NP-hardness.

[39] Doctors, dr.p Ganesh Kumar, dr.Yassen, in Imam Muhammaf ibn Saud Islamic University.