Genetic Algorithms

Project for CSC 589B Course

Zahra Sasanian
V00715027

Table of Contents

1. Introduction

The idea of evolutionary computing was first introduced in the 1960s by I. **Rechenberg** in his work "***Evolution strategies***" (*Evolutionsstrategie* in original)[1]. His idea was then developed by other researchers. **Genetic Algorithms** (GAs) were invented by John **Holland** and developed by him and his students and colleagues [1].

Genetic algorithms (GAs) are general search and optimization algorithms inspired by processes normally associated with the natural world (natural selection). They are search techniques used in computing to find exact or approximate solutions to optimization and search problems. They are particularly well suited for hard problems where little is known about the underlying search space. Fig.1 shows the position of Genetic Algorithms among other search techniques. Genetic Algorithms have many applications in the computer and social sciences and in engineering [2].

There are many optimization problems which have very large search space. In these problems, usually it is impossible to use brute-force solutions or exhaustive search algorithms to find the optimum solution in the search space. On the other hand, sometimes we need to find a near optimal solution in a limited time; therefore, we need to use other kind of searches, one of which is Genetic Algorithm. We don't need to know any rule or regularity among solutions in search space to run a Genetic Algorithm on it, that's why this type of algorithms are very useful where our knowledge of the underlying search space is limited.

In this project, first we discuss the basic concepts corresponding Genetic Algorithms, then we will show an example of using a Genetic Algorithm in solving the biological multiple sequence alignment problem.

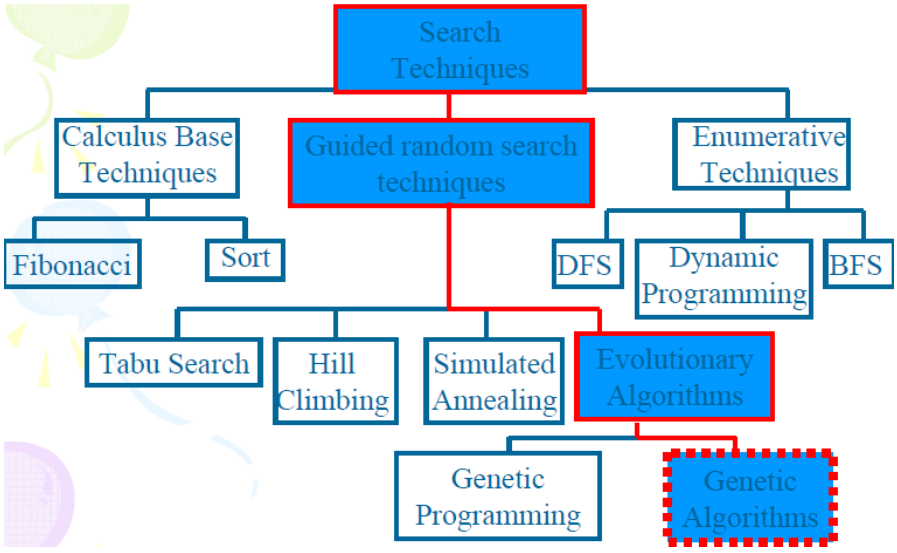

Fig.1. Classification of search techniques and position of GAs among them.

[1] http://en.wikipedia.org/wiki/Evolution_strategy

2. Genetic Algorithm

Genetic Algorithm works similar to what happens in nature as species evolve by natural selection. It starts with a set of solutions (represented by chromosomes) called population. Some solutions from one population are taken and used to form a new population. This is motivated by a hope, that the new population will be better than the old one. Solutions which are selected to form new solutions (offspring) are selected according to their fitness - the more suitable they are the more chance they have to reproduce. This process is repeated until some condition (for example number of populations or improvement of the best solution) is satisfied. Below we describe the parameters of the Genetic Algorithm more specifically.

2.1. Population

Each population consists of a set of individuals (chromosomes). Each chromosome is a solution to the problem which is being solved by Genetic Algorithm. Therefore, the first step to solve a problem using Genetic Algorithms is to encode its solutions to a structure similar to chromosome. Chromosomes can be strings, permutations, sets or any other data structure.

2.2. Search Space

In a numerical search or optimization problem, possible solutions are being searched in order to locate the solution that best describes the problem. The set (or list) of all possible solutions to the problem which are searched is called search space. In such a space, some measure of distance between solutions can be defined and each solution can be assigned a measure of success, or fitness within the problem. Better performing or fitter solutions will then occupy the peaks within the search space.

2.3. Fitness Function

In order to evaluate the solutions, we need some measure of optimality which is called Fitness Function. A fitness function is a particular type of objective function that prescribes the optimality of a solution (that is, a chromosome) in a genetic algorithm so that, that particular chromosome may be ranked against all other chromosomes. At each evolutionary step, new chromosomes are produced by fitter solutions (solutions which have higher fitness value) from the previous generation, which have more chance to survive during the evolution.

2.4. Selection

The purpose of the selection is to focus the search in promising regions of the search space. It is inspired by Darwin's theorem, "survival of the fittest". During each successive generation, a proportion of the existing population is selected to breed a new generation. Individual solutions are selected through a fitness-based process, where fitter solutions (as measured by a fitness function) are typically more likely to be selected. Certain selection methods rate the fitness of every solution and select the best solutions. Other methods rate only a random sample of the population, as this process may be very time consuming.

Selection can be performed from the previous population or from the individuals in the search space which are not in the previous population. The former is called exploitation while the latter is called exploration. There is a trade-off between exploration and exploitation. Genetic algorithms usually consider probabilities for each of these possibilities. Too much stress on exploration results in a pure random search whereas too much exploitation results in a pure local search. Clearly, intelligent search methods must reside somewhere in the continuous spectrum in between these extremes.

2.5. Crossover (Recombination)

For each new solution to be produced, a pair of parent solutions is selected for breeding using the selection operation described above. By recombining the sub-solutions of parents, a new solution is created which typically shares many of the characteristics of its parents. New parents are selected for each new child, and the process continues until a new population of solutions of appropriate size is generated. Although reproduction methods that are based on the use of two parents are more "biology inspired", some research suggests more than two parents are better to be used to reproduce a good quality chromosome.
This process ultimately results in building the next generation of chromosomes that is different from the initial generation. Generally the average fitness values of the population are increased during this procedure, since only the best organisms from the last generation are selected for breeding along with a small proportion of less fit solutions for avoiding local minimums. There are different types of crossovers such as: “one point” crossover, “two points” crossover, “cut and splice” crossover, etc.

2.6. Mutation

This operation simulates the mutation which occurs in biological selection. The purpose of mutation is to simulate the effect of errors that happen with low probability during duplication, and to preserve and introduce diversity. It results in moving to new areas in the search space or restoring lost information to the population.
The classic example of a mutation operator involves a probability that an arbitrary bit in a genetic sequence will be changed from its original state. A common method of implementing the mutation operator involves generating a random variable for each bit in a sequence. This random variable tells whether or not a particular bit will be modified. This mutation procedure is called single point mutation. Other types of mutation are inversion and floating point mutation. When the gene encoding is restrictive as in permutation problems, mutations are swaps, inversions and scrambles.

2.7. Termination Condition

Creating new generations continues until a termination condition is satisfied. This condition can be a pre-determined number of generations or amount of elapsed time. It also can be achieving a satisfactory solution, or having no improvement in solutions quality over a pre-determined number of generations.

2.8. Overall Structure

Fig.2 shows the general structure of a Genetic Algorithm. At first step, the initial population is generated from the solutions in the search space. Then the fitness or optimality of each individual is evaluated by a fitness function and a fitness value is assigned to each individual of the initial population.
In evolutionary loop, while the termination condition is not satisfied, the following steps are performed. First using the selection mechanism, a set of fitter individuals is selected for reproduction. These set maybe selected from the previous population or the search space. Then recombination (crossover) is applied to the selected set and new offspring are created. Then with a low probability the mutation operation is applied on each new offspring. After that, the fitness values of the new individuals are evaluated and assigned to them. The next population is built using new offspring and old population. It is common to select the upper 50% part of the sorted population using fitness values containing both new individuals as well as old ones. The evolution continues until the termination condition is met.

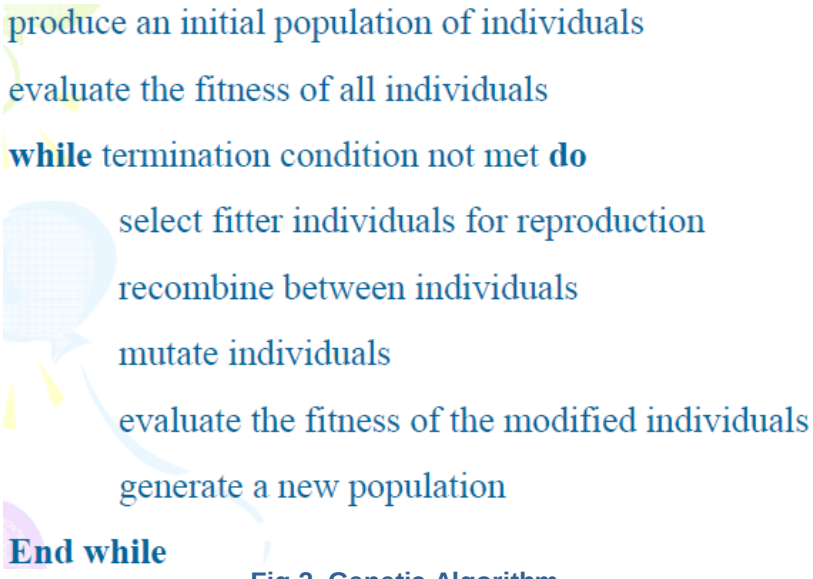

Fig.2. Genetic Algorithm

3. Biological Example: A Genetic Algorithm for Multiple Sequence Alignment

As an example of genetic algorithms, we use an algorithm introduced by Jorng-Tzong et. al. [3] which solves the multiple sequence alignment problem in biology using genetic algorithms.

3.1. Introduction to Multiple Sequence Alignment (MSA)

Multiple sequence alignment is an important analytical process which helps biologists to discover patterns among the chromosomes of different species. There are different types of

biological sequences. The DNA sequence is composed of four kinds of elements called bases, denoted as A, T, C, and G. Protein sequence has 20 alphabets C, S, T, P, A, G, N, D, E, Q, H, R, K, M, I, L, V, F, Y, W. Multiple sequence alignment can help compare the structureal relationship between sequences by simultaneously aligning multiple sequences to construct connections between the elements in different sequences.
While many algorithms have been proposed for solving the multiple sequence alignment problem, there is still a need to find better solutions since the problem is NP-Complete. Dynamic programming can be useful to solve the MSA problem for short length sequences but for long sequences it is not the best solution since it consumes system resources. This paper introduces a genetic algorithm for solving MSA problem. For simplicity and without loss of generality, we avoid some mathematical representations in paper and try to describe them verbally or by showing examples.

3.2. Chromosome Encoding

A chromosome is defined as a set of strings of numbers (say number-string) with fixed lengths that represent aligned sequences, including gaps. Each number in a number-string is unique and corresponds to a position of a certain character in a sequence. A chromosome X of an alignment whose length is N is composed of sequences X_1, X_2, ..., X_k denoted as $X_1\#\ X_2\#...\#\ X_k$ can be represented by:

$$X_1 \# X_2 \# \dots \# X_K = (x_{1,1}, x_{1,2}, \dots, x_{1,m_1})(x_{2,1}, x_{2,2}, \dots, x_{2,m_2}) \dots (x_{k,1}, x_{k,2}, \dots, x_{k,m_k})$$

where $(x_{i,1}, x_{i,2}, \dots, x_{i,m_i})$ is a number-string represented for the i-th sequence in an alignment. The symbol # is represented for concatenation, and $x_{i,j}$ is a number indicating the position of a nucleotide at the i-th row and j-th column ($x_{i,j} < x_{i,j+1}$). m_i is the length of the i-th number-string. The N value limits the longest length of alignments that chromosomes can represent. It is important to select the N value considering the size of search space. If the N value is too small, optimal alignments of less similar sequences cannot be found. If the N value is too large, it needs more time to find the optimal alignment of highly similar sequences. In this paper, N is determined by the equation $N = n_{max} \times (1 + r_{sp})$ in which r_{sp} is space ratio and n_{max} is the maximum length of the sequences. As an example, consider the alignment of Fig.3. In this figure $n_{max} = 10$ and r_{sp} is set to 0.2. The chromosome that corresponds to this alignment is:

(0,11) (0, 3, 5, 8, 11) (1, 8, 10,11) (6, 8, 11)

```
-ATCCGCTTAC
-CT-C-CT-AG
A-TCCGCT-A-
TCTCCG-T-AC
```

Fig.3. Sequence Alignment

3.3 System Process Flow

For solving the MSA problem for DNA sequences, the procedure of Algorithm1 is applied. In Algorithm 1, the symbol **|P|** represents the size of the population. The concept of the system process flow is based on the architecture of the genetic algorithms described in section 2.8.
The first population is generated randomly. The numbers in the *i*-th number-string of the chromosome *X* are generated by randomly picking m_i unique numbers ($m_i < N$) and then sorting these numbers increasingly. The method of (Goldberg 1989) is used to select the chromosomes of the next population from the set of offspring and the original chromosomes. The evolution is repeated until the following termination conditions are satisfied.

- The number of generations exceeds the maximum denoted as g_{max}.
- The best fitness is not improved over a certain number of generations denoted as b_{max}.

Algorithm 1 The Flow of Our Approach

```
generate the initial population P
let n be multiplied population size by
crossover rate
while not satisfy the termination condition do
  for i = 1 to n do
    select two chromosomes x and y
    from population at random
    let X' = Ẋ and Y' = Ẏ
    mutation (X')
    mutation (Y')
    crossover (X', Y')
    add (X') and (Y') into mating pool
  end for
  select the best top |p|chromosomes to replace
  the original population
end while
```

Fig.4. Genetic Algorithm for MSA

3.4 Fitness

The sum-of-pairs function (Setubal and Meidanis 1997) is used to evaluate the fitness of the generated chromosomes. When computing the fitness, a chromosome must be converted to the alignment form. The sum-of-pairs score is defined as the sum of pair wise scores of all pairs of sequences. Three kinds of scores (match, mismatch, and gap) are defined for DNA sequences.

3.5 Crossover

In the crossover process, two parent chromosomes, denoted as *X* and *Y* are randomly selected in order to produce two offspring chromosomes. Crossover points (cutting points) are randomly selected in parent chromosomes. The blocks among the cutting points are called crossover blocks. The crossover blocks are selected randomly. The definition of the crossover blocks is given below. Let the parent chromosomes *X* and *Y* be represented by:

$$X_1 \# X_2\# \dots \# X_K = (x_{1,1}, x_{1,2}, \dots, x_{1,m_1})(x_{2,1}, x_{2,2}, \dots, x_{2,m_2}) \dots (x_{k,1}, x_{k,2}, \dots, x_{k,m_k})$$

and

$$Y_1 \# Y_2\# \dots \# Y_K = (y_{1,1}, y_{1,2}, \dots, y_{1,m_1})(y_{2,1}, y_{2,2}, \dots, y_{2,m_2}) \dots (y_{k,1}, y_{k,2}, \dots, y_{k,m_k})$$

respectively.

$$\text{If } \forall i = 1 \dots k, \left\{ \begin{array}{c} (\exists a_i,\ x_{i,a_i} < p < x_{i,a_{i+1}} \text{ and } y_{i,a_i} < p < y_{i,a_{i+1}}) \\ \text{and} \\ (\exists b_i,\ x_{i,b_i} < p < x_{i,b_{i+1}} \text{ and } y_{i,b_i} < p < y_{i,b_{i+1}}) \end{array} \right\},$$

the $(x_{1,a_1+1}, \dots, x_{1,b_1}) \dots (x_{k,a_k+1}, \dots, x_{k,b_k})$ and $(y_{1,a_1+1}, \dots, y_{1,b_1}) \dots (y_{k,a_k+1}, \dots, y_{k,b_k})$ are defined as crossover blocks $A_{p,q}$ and $B_{p,q}$ for *X* and *Y* respectively.

An example of the functionality of the crossover algorithm is depicted in Fig.5. In this example, the parent chromosome *X* has two crossover blocks $A_{0,12}$ and $A_{12,22}$, and the parent chromosome *Y* has two crossover blocks $B_{0,12}$ and $B_{12,22}$. So, the parent chromosomes are broken to their crossover blocks and the recombination process merges their blocks resulting creation of two new chromosomes $A_{0,12}, B_{12,22}$, and $B_{0,12}$, $A_{12,22}$. The fitness value (SP-Score) of the $A_{0,12}B_{12,22}$ chromosome is better than the fitness value of $B_{0,12}, A_{12,22}$. So, it is the best child among two children.

X=(1,4,5,7,8,9,10,15,16,18)(0,3,6,7,10,14,16,17,
20)(3,6,8,10,16,20,21)
Y=(2,4,5,6,8,9,11,14,17,20)(3,5,6,8,11,13,15,17,
20)(3,7,9,11,15,17,18)
$A_{0,12}$=(1,4,5,7,8,9,10)(0,3,6,7,10)(3,6,8,10)
$B_{0,12}$=(2,4,5,6,8,9,11)(3,5,6,8,11)(3,7,9,11)
$A_{12,22}$=(15,16,18)(14,16,17,20)(16,20,21)
$B_{12,22}$=(14,17,20)(13,15,17,20)(15,17,18)
best child = (1,4,5,7,8,9,10,14,17,20)
(0,3,6,7,10,13,15,17,20)(3,6,8,10,15,17,18)
random child = (2,4,5,6,8,9,11,15,16,18)
(3,5,6,8,11,14,16,17,20)(3,7,9,11,16,20,21)

Fig.5. Crossover example

3.6. Mutation

Algorithm 2 shows the mutation process. Four kinds of mutation operators are defined in this paper, namely, MergeSpace, MoveSpaceCol, FullSpaceCol and MoveRowSpace. During the mutation process, each chromosome is mutated *n* times by several operators randomly selected from the four mutation operators. The frequency of applying each mutation operator is controlled by its probability.

Algorithm 2 Mutation (a chromosome X)
n = (the longest length of the number-string)/
  $1000*r_{sp}$
for i=1 **to** n **do**
  randomly select one operator from four
  mutation operators to alter the chromosomes
end for

Fig.6. Mutation Algorithm

The value *n* is in proportional to the longest length of number-strings in chromosomes. The purpose of the MergeSpace operator is to merge two or three gaps together. The MoveSpaceCol operator is aimed to move gaps in given columns to the neighborhood columns. The FullSpaceCol operator changes a space column to a full space column. In this paper, by full space column they mean a full column without any spaces! The MoveRowSpace operator rearranges the gaps of some sequences within given continuous columns. Below, the four mutation operators are discussed in details.

3.6.1. MoveRowSpace Operator

The MoveRowSpace operator rearranges positions of spaces the columns of some sequences. This operator is based on the basic two-sequence alignment using dynamic programming with some modification in the dynamic programming table. Algorithm 3 shows this operator.

Algorithm 3 MoveRowSpace (a chromosome X)
Randomly pick some columns which begin with c_1
and end with $C_{q'}$
s.t. $\exists$ a space column $C \in (C1, Cq)$
is not a full space column
Find the sub-number-string $(x_{i,a1}, x_{i,a2}, \ldots, x_{i,ai})$
in each number-string of X s.t.$c_1 \leqq x_{i,a1} \leqq c_q$
Make the template sequences s
According to the template sequence s,
recomputed the numbers in the
 sub-number-strings selected at random

Fig.6. Mutation operator1: MoveRowSpace

An example of applying the above operator is given in Fig.7. Fig.7a shows an input chromosome *X*. First, the columns from 20 to 49 are selected randomly. Fig.7b shows the corresponding sub-alignment of the selected part of chromosome *X* which begins at 20 and ends at column 49. A template sequence is made in which symbols have the highest frequency rate in each column of the sub-alignment. Then, a target sequence is produced by deleting the spaces from a sequence randomly selected in the sub-alignment. In Fig.7c, *t* is the target sequence produced from the first sequence in the sub-alignment and *s* is the template sequence.
Next, the target and the template sequences are aligned using a dynamic programming algorithm under the condition that no new space is allowed to be inserted into the template sequence. After the aligning process, new spaces in the target sequence are produced (see Fig.7d). Finally, the number-string of the generated subsequence is replaced in the number-string of the chromosome *X*.

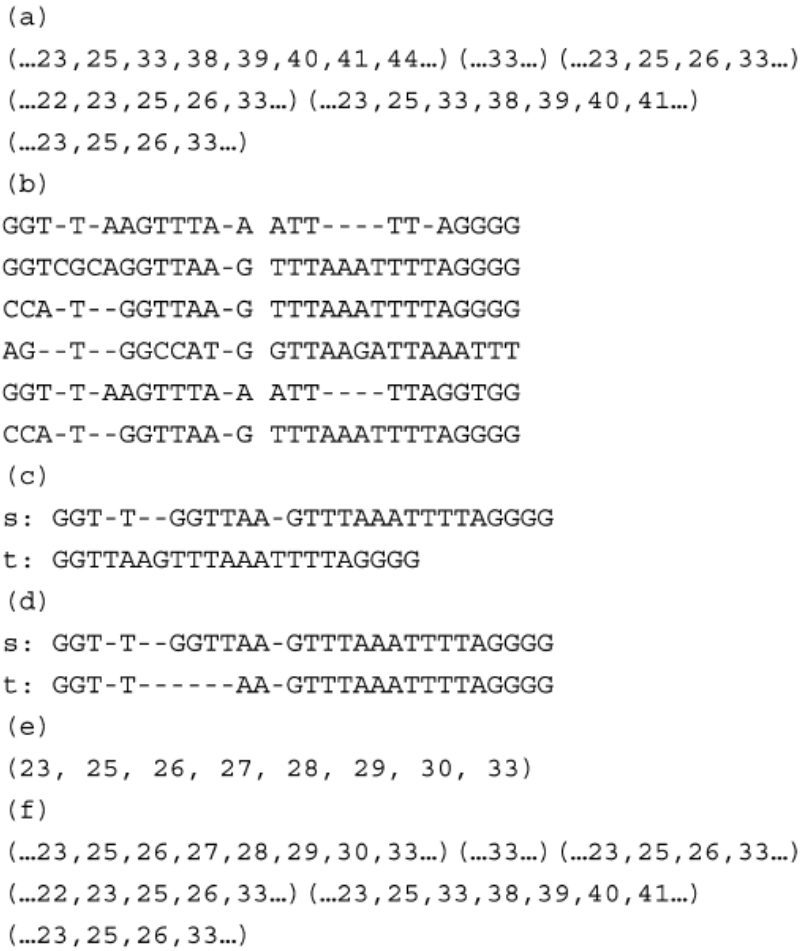

Fig.7. Example of MoveRowSpace

3.6.2. MergeSpace Operator

The MergeSpace operator merges some spaces of a sequence together and then shifts to other columns. The details of MergeSpace operator is given in Algorithm 4.

Algorithm 4 MergeSpace(a chromosome X)

Pick a number-string $x^*_i=(x_{i,1},x_{i,2},\ldots,x_{i,mi})$ in X at random
Choose two or three spaces to apply this operator
if Randomly select two numbers $x_{i,g}$ and $x_{i,g+1}$ **then**
  Select two numbers h and $h+1 \notin x^*_i$
  Replace the numbers $x_{i,g}$ and $x_{i,g+1}$ to h and $h+1$ respectively
else if Randomly select three numbers $x_{i,g}$, $x_{i,g+1}$, $x_{i,g+2}$ **then**
  Select three numbers h, h+1 and h+2 $\notin$ x*I
  Replace the numbers $x_{i,g}$, $x_{i,g+1}$, $x_{i,g+2}$ to h, $h+1$ and $h+2$ respectively
end if
Sort the numbers in x^*_i by increasing

Fig.8. Mutation operator2: MergeSpace

In this operator, two or three spaces are selected randomly and move to other columns. An example is given in Fig.9a. The spaces at positions 7, 9 and 10 in the third number-string are selected to shift and merge together; therefore, the new spaces are generated at positions 13, 14 and 15.

Y=(2,4,5,6,8,9,10,14,17,20)(3,5,6,8,10,14,15,17,20)(3,7,9,10,16,17,18)
(a) Y=(2,4,5,6,8,9,10,14,17,20)(3,5,6,8,10,14,15,17,20)(3,**13,14,15**,16,17,18)
(b) Y=(2,4,5,6,8,9,10,14,17,20)(3,5,6,8,10,14,15,17,20)(3,7,9,10,**14**,17,18)

Fig.9. Example of MergeSpace

3.6.3. FullSpaceCol Operator

The FullSpaceCol operator is depicted in Algorithm 5 of Fig.10. This operator generates full space columns (i.e. columns without any spaces) by grabbing and moving the neighborhood spaces. Fig.9b shows an example of the FullSpaceCol operator. In this example, the column 14 is selected to become a full column. The first two sequences have values on position 14 but the last one has a space. Among the neighbors, positions 9 and 16 are not completely full and not completely empty, so we can move one of them to 14 to fill column 14. The number 16 is replaced by 14 since it is the nearer to 14 comparing to 9.

Algorithm 5 FullSpaceCol(a chromosome X)
Randomly pick a column u s.t. C_{score}
$(X^*_{1,u}, X^*_{2,u}, \ldots, X^*_{k,u}) < U_{thrd}$
for all number-string $x^*_i = (x^*_{i,1}, x^*_{i,2}, \ldots, x^*_{i,mi})$ **do**
  if $u \notin x^*_i$ **then**
    Find a number p from the nuber-string
    x^*_i s.t. p is a nearest neighborhood
of u and then column p is not a full space column
    Sort the number-string $x * I$ by increase
  end if
end for

Fig.10. Mutation operator3: FullSpaceCol

3.6.4. MoveSpaceCol Operator

The MoveSpaceCol operator moves a set of spaces in a specified range to the neighborhood columns. The details of this mutation operator are given in Algorithm 6. First, a set of columns $A = \{a_1, a_2, \dots, a_p\}$ is selected such that satisfies the following conditions.

- Columns which are in the range a_1 to a_p but are not in the set A must be full space columns.
- The scores of all columns in A are less than a threshold, which is the score of the column in which one third of sequences have spaces and the others have the same base symbols.
- Number of elements in A is less than or equal to twenty.

Then, the spaces in the column set A will be moved to the neighbor columns of A. The MoveSpaceCol operator does not destroy the full space columns of the chromosomes.

Algorithm 6 MoveSpaceCol(a chromsome X)

Find a column set $A = \{a_1, a_2, \dots a_p\}$

  under some conditions

Choose the direction for moving the spaces in A

if the direction is left-hand side **then**

  $q = a_1 - 1$

  for $u = 1$ **to** 2 **do**

    for $i = p$ to 1 **do**

  Find a column c, s.t. the number c is the

  largest number and is less than or equal to q

  and the column c is not a full space column

  Replace all the a_i in number-strings to c,

  if c is not exists in the corresponding

  number-string

    $q = c - 1$

    end for

  end for

else if the direction is right-hand side **then**

  The process is similar to the process

  of moving spaces to left-hand side

end if

Fig.11. Mutation operator4: MoveSpaceCol

4. Results

The number of evolutions in experiments is set to 30. Table 1 shows the experimental results for average and best values obtained by running the genetic algorithm on a set of test data. The 'Score' column indicates SP-Score of alignments. The 'M.C.' column denotes the number of match columns. The numbers of generations and CPU time elapsed in seconds are represented in the 'Gen#' and the 'Time' columns respectively. The same data is applied to the best current alignment using ClustalW [4]. For comparison, the scoring method used in this approach is used to evaluate the ClustalW results as well. As it is shown in the table, more than half of the results from this algorithm are better than the ones generated by ClustalW, and the rest are close to ClustalW.

Table 1. Results of our Genetic Algorithm in comparison with ClustalW

	Average				Best				ClustalW	
ID	Score	M.C.	Gen#	Time	Score	M.C.	Gen#	Time	Score	M.C.
D1	18627	198	475	0.633	18627	198	441	0.591	18627	198
D2	25343	449	602	0.731	25343	449	530	0.621	25343	449
D3	47889	109	492	0.591	47889	109	456	0.521	47889	109
D4	79605	94	3439	7.784	82762	108	3002	6.830	81544	107
D5	33194	1610	693	3.266	33235	1614	583	2.804	33211	1612
D6	26737	2253	1653	24.908	27018	2283	1982	29.793	27315	2306
D7	58004	879	3023	21.561	59235	902	3601	25.507	59499	904
D8	87658	1440	928	4.630	87693	1442	760	3.986	87690	1440
D9	30923	2879	3785	259.949	43971	3847	2231	146.861	51018	4414
D10	18812	907	843	2.62	18878	909	723	1.783	18845	905
D11	57813	999	998	2.868	57870	1001	749	2.243	57825	999
D12	36808	1124	1190	4.522	37177	1136	1601	6.69	37078	1122
D13	51756	1115	2336	15.257	53016	1164	1928	12.968	53796	1193
D14	14602	676	2910	7.397	17094	828	3914	9.794	17496	855
D15	79439	2638	736	10.672	79473	2642	733	10.706	79371	2638
D16	143906	2167	3096	89.256	150480	2318	2527	72.784	151308	2352
D17	514741	5939	5585	1208.683	529892	6229	6025	1302.874	534044	6315

5. Conclusion

In this project, we introduced Genetic Algorithms as one of the possible solutions to solve optimization problems and discussed their applications and importance. Then, the general structure and parameters of genetic algorithms were explained. As an example, we described a genetic algorithm which is introduced to solve the Multiple Sequence Alignment (MSA) problem in biology. The results show improvement over the best available solution.

6. References

[1] John H. Holland, "Adaption in Natural and Artificial Systems," *MIT Press*, 1975.

[2] David A. Coley, “An Introduction to Genetic Algorithms for Scientists and Engineers,” *World Scientific Publishing*, 1999.

[3] Jorng-Tzong Horng, Li-Cheng Wu, Ching-Mei Lin, and Bing-He Yang, “A genetic algorithm for multiple sequence alignment,” *Soft Computing Journal*, 2005. (http://rsdb.csie.ncu.edu.tw/tools/msa.htm)

[4] Thompson J, Higgins D and Gibson T, “CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position specific gap penalties and weight matrix choice,” *Nuc Acids Res,* 1994.