A Binary Integer Programming model for computing DNA Sequence Alignment

Nawar N. Qubat
College of Computer Sciences and Mathematics
University of Mosul

Received on:24/8/2008 **Accepted on:14/10/2008**

ABSTRACT

DNA Sequence Alignment is an important problem in computational biology and is useful for comparing genomes and finding genes, for determining evolutionary linkage of different biological sequences. Dynamic Programming Problems is discussed and applied to solve this problem. This paper is concerned with computing DNA Sequence Alignment firstly by formulating a Binary Integer Programming model to compute the string sequence in Edit Distance Problem then re-formulating this model to be suitable to compute this alignment. By this model we gave a good role for Operations Researches field to prove it's efficient to solve problems of molecule of life. The suggested model is applied to solve an example in Edit Distance Problem then used again after re-formulating it for an example in DNA Sequence Alignment Problem.

Keywords: DNA Sequence Alignment, Edit Distance, Binary Integer Programming, Dynamic Programming.

نموذج برمجة ثنائي صحيح للحساب محاذاة تسلسل الحمض النووي

نوار نجم قباط

كلية علوم الحاسوب والرياضيات، جامعة الموصل

تاريخ الاستلام: 2008/8/24 **تاريخ القبول: 2008/10/14**

الملخص

اصطفاف سلسلة الـ DNA تعتبر مسألة مهمة في علم الأحياء الحسابي ومفيدة في مقارنة المورثات وإيجاد الجينات وفي تقرير الترابط التطوري من السلاسل الحيوية المختلفة. مسائل البرمجة الديناميكية قد نوقشت وطبقت في حل هذه المسألة. هذا البحث اهتم في حساب اصطفاف سلسلة الـ DNA أولاً بصياغة نموذج برمجة صحيحة ثنائية لحساب سلسلة الكلمة في مسألة تحرير المسافة ثم بعد ذلك تم إعادة صياغة هذا النموذج ليكون مناسباً لحساب هذا الاصطفاف. بواسطة هذا النموذج أعطينا دور جيد لحقل بحوث العمليات في حل مسائل جزيئة الحياة. النموذج المقترح طبق في حل مثال في مسألة تحرير المسافة ثم بعد ذلك استخدم مرة أخرى بعد إعادة صياغته في حل مثال في مسألة اصطفاف سلسلة الـ DNA.

الكلمات المفتاحية: محاذاة تسلسل الحمض النووي ، تحرير المسافة ، البرمجة الثنائية الصحيحة ، البرمجة الديناميكية.

1. Introduction:

Sequence Alignment is used in several different fields including molecular biology, string editing, speech processing, and codes and error control. In computational biology this alignment known as DNA Sequence Alignment which is an important problem for comparing genomes and finding genes, for determining evolutionary linkage of different biological sequences.[9]

DeoxyriboNucleic Acid (DNA) is the hereditary molecule of life. Information encoded within DNA confers physical characteristics such as hair color, eye color, and susceptibility to certain diseases. By unraveling the secrets encoded within DNA, numerous biological and medical discoveries have been and continue to be made.

A DNA sequence is a string formed from a four-letter alphabet {Adenosine (A), Thymidine (T), Guanosine (G), Cytidine (C)} of biological macromolecules,for example:ATTCGGATCGGAATCGTAGCC represents a string of the nucleotides A, G, C and T.

One basic technique to glean information from a DNA molecule is to compare it to other DNA molecules. This comparison, generally called an alignment, is fundamental to the emerging field of bioinformatics revealing where sequences are the same and where are they different. The differences could be invaluable, for example, in helping to explain why individuals have different susceptibility to disease. [12]

DNA Sequence alignment reveals the relations between the characters in different sequences, and there are the reverse complement relations between the characters in DNA double strand.

The measuring of the similarity by Sequence Alignment Problem between two DNA sequences is basically depends on Edit Distance Problem concept for finding the similarity between two string.

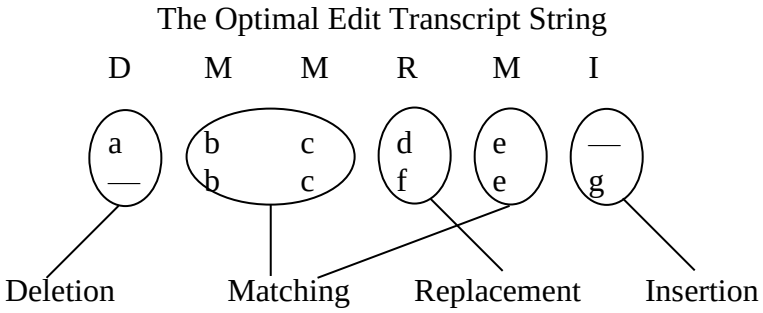
Our study represents introducing on Edit Distance Problem in section 2 and then formulating a binary integer programming model for computing the string sequence via this problem in section 3 and re-solving an example by this model after solving it by Dynamic Programming Problem. Section 4 gives more details on DNA Sequence Alignment Problem and re-formulating the binary model in section 3 to be suitable to solve this alignment and re-solving an example by this model after solving it by Dynamic Programming Problem with MATLAB codes in section 6. Finally, some conclusions were reached in section 5 on this model on it's effective for computing DNA Sequence Alignment.

2. Computing the string sequence:

2.1. Edit Distance Problem:

Edit Distance (or Levenshtein distance [10]) between two strings is defined as the minimal number of edit operations which must be performed between them, character by character to transform one string into another, the edit operations are: Replacement (R), Matching (M), Deletion (D) and Insertion (I) and the string over the (R, M, D and I) is called an edit transcript string of the two strings.[6]

Simply, to transform the string **abcde** to the string **bcfeg**, we can delete **a** then replace **d** by **f** and finally insert **g**, yielding to the string **bcfeg**. Also by the following description:



Edit Distance Problem (EDP) is to compute the edit distance between two given strings along with an optimal edit transcript that describes the transformation.

The symbol "—" indicated to a gap which occur of the deletion and insertion operations only, the minimal edit distance operations is the number of columns in which the characters differ. The aim of finding the minimal edit distance between two strings is for searching for strings of strings with most similarity.

2.2. Edit Distance Formula:

For finding the value of edit distance between two strings we have to use a general description of the problem which as follows:

Let S and T be a two strings with length m and n, respectively, we can define: $E(i, j)$ as the value of edit distance between S and T for $i = 1,..., m$, $j = 1,..., n$, $E(m, n)$ is the minimal edit distance between S and T, $E(m,0)$ or $E(0, n)$ is the minimal effort required to transform the string S into the null string or the string T into the null string.

In order to calculate $E(i, j)$ we must have an initial conditions:

$E(0,0)=0$,
$E(i,0)=i, 1\le i\le m$,
$E(0,j)=j, 1\le j\le n$, and recurrence relation used for $1\le i\le m, 1\le j\le n$ defined as:

$$E(i,j)=\min\begin{cases}E(i-1,j)+\sigma(S(i),-).\\ E(i,j-1)+\sigma(-,T(j)).\\ E(i-1,j-1)+\sigma(S(i),T(j)).\end{cases} \quad \ldots(1)$$

Where $\sigma(S(i),-)=1$ (deletion operation cost),
$\sigma(-,T(j))=1$ (insertion operation cost),

$$\sigma(S(i),T(j))=\left\{\begin{array}{l}0 \quad \text{if} \quad S(i)=T(j) \ \text{(matching operation cost)}\\ \\ 1 \quad \text{if} \quad S(i)\neq T(j) \ \text{(replacement operation cost)}\end{array}\right\} \ldots(2)$$

See ([2], [7], [9] & [11]).

We can called to $\sigma(s(i),-), \sigma(-,T(j))$ and $\sigma(S(i),T(j))$ by **scoring functions**. Thus, the minimal edit distance between the strings S=**abcde** with $m=5$ and T=**bcfeg** with $n=5$ is $E(m,n)=E(5,5)=3$.

2.3. Computing Edit Distance:

In order to calculate $E(m,n)$ we must be able to calculate $E(i,j)$ for any i and j, this calculation can be done recursively or through Dynamic Programming. Dynamic Programming Problem (DPP) is a powerful technique of calculating values more than once to be used to calculate other values later.

DPP for computing strings sequence depends completely on Edit Distance formula in (eqs. (1) & (2)) by filling a table from left to right and top to bottom starting by the initial conditions which are the sets of known values $E(i,0)$ and $E(0,j)$ and bottom-right most value will be the minimal edit distance $E(m,n)$. [2]

2.4. Example:

The table below shows the transformation of the string VINTNER to WRITERS by using DPP, the rows corresponds to a different edit operations, horizontal for insertion, vertical for deletion and diagonal for replacement and matching.

Table (2.1)

	0	1	2	3	4	5	6	7

$E(i,j)$		W	R	I	T	E	R	S
0		0	1	2	3	4	5	6	7
1	V	1	1	2	3	4	5	6	7
2	I	2	2	2	2	3	4	5	6
3	N	3	3	3	3	3	4	5	6
4	T	4	4	4	4	3	4	5	6
5	N	5	5	5	5	4	4	5	6
6	E	6	6	6	6	5	4	5	6
7	R	7	7	6	7	6	5	4	5

We chose an arbitrary edit distance calculations from table (2.1) such that:

$$E(2,3)=\min\begin{cases}E(1,3)+1=3+1=4\\E(2,2)+1=2+1=3\\E(1,2)+0=2+0=2\end{cases}=2$$

$$E(5,4)=\min\begin{cases}E(4,4)+1=3+1=4\\E(5,3)+1=5+1=6\\E(4,3)+1=4+1=5\end{cases}=4$$

And the terminal value $E(7,7)=5$ which gives the minimal edit distance of the two strings, such that:

$$E(7,7)=\min\begin{cases}E(6,7)+1=6+1=7\\E(7,6)+1=4+1=5\\E(6,6)+1=5+1=6\end{cases}$$

A MATLAB code presented in section 6 and has been used to solve the Edit Distance Problem with function (`EditDist`).

If we remove the last 3 columns and the last 2 rows from the table (2.1) of the optimal edit distance then the remaining columns and rows represent an optimal edit distance for the remaining substrings and this property called Prefix, see below:

V	I	N	T	N	E	R

W	R	I	T	E	R	S

Here, the minimal edit distance of the two substrings VINTN and WRIT will be $E(5,4)=4$. [9]

2.5. Recovering the string itself:

DPP described how to compute the edit distance between two strings and now we explain how the optimal string recover itself. The key idea is to retrace the optimal paths of the Dynamic Programming backwards, re-discovering the path of choices from $opt[m][n]$ into $opt[0][0]$, (opt means optimal edit distance) which depends on the following three possibilities:

I. If a character $S(i)$ up to $T(j)$ for $1 \le i \le m$, $1 \le j \le n$, then:

$$opt[i-1][j-1] = opt[i][j] \quad \text{if} \quad S(i) = T(j). \qquad \ldots(3)$$

$$opt[i-1][j-1] = opt[i][j] - 1 \quad \text{if} \quad S(i) \neq T(j). \qquad \ldots(4)$$

II. If a character $S(i)$ up with a gap for $1 \le i \le m$, $0 \le j \le n$, then:

$$opt[i-1][j] = opt[i][j] - 1. \qquad \ldots(5)$$

III. If a character $T(j)$ up with a gap for $1 \le j \le n$, $0 \le i \le m$, then:

$$opt[i][j-1] = opt[i][j] - 1. \qquad \ldots(6)$$

Now, by the (eqs. (3), (4), (5) and (6)) the recovering of the optimal paths of the strings in above example from $opt[7][7]$ into $opt[0][0]$ and according to the inversion of the arrows in the shaded squares we get:

$$opt[7][7] - 1 = 4 = opt[7][6] = opt[6][5] = opt[5][4]$$
$$opt[5][4] - 1 = 3 = opt[4][4] = opt[3][3]$$
$$- -$$
$$opt[3][3] - 1 = 2 = opt[2][2]$$
$$opt[2][2] - 1 = 1 = opt[1][1]$$
$$opt[1][1] - 1 = 0 = opt[0][0]$$
$$OR \; -$$
$$opt[3][3] - 1 = 2 = opt[2][3] = opt[1][2]$$
$$opt[1][2] - 1 = 1 = opt[0][1]$$
$$opt[0][1] - 1 = 0 = opt[0][0]$$
$$OR \; -$$
$$opt[3][3] - 1 = 2 = opt[2][3] = opt[1][2]$$
$$opt[1][2] - 1 = 1 = opt[1][1]$$
$$opt[1][1] - 1 = 0 = opt[0][0]$$

Therefore, the final transformations of the similarity of both strings VINTNER and WRITERS according to the recovering of the optimal paths will be:

V I N T N E R —

W R I T — E R S

OR,

— V I N T N E R —
W R I — T — E R S

OR,

V — I N T N E R —
W R I — T — E R S

Respectively.

The first path of transformation string VINTNER to WRITERS is the best one than the others because it has the least number of gaps (two gaps) and minimal length (8) than the two other paths. [13]

3. Binary Integer Programming model for Edit Distance:

3.1. Binary Integer Programming Problem:

A Binary Integer Programming Problem (BIPP) is kind of Integer Programming Problem (IPP) and is given by vector $c \in \Re^n$, $b \in \Re^m$ and matrix $A \in \Re^{n \times m}$. The goal of the problem is to find a vector $x \in \mathrm{B}^n = \{0,1\}$ solving the following optimization problem: [8]

Max $c^T x$

s.t. $Ax \le b$

$x \ge 0$

3.2. Formulating BIP model for computing Edit Distance:

Let S and T be a two strings with length m and n, respectively, we can formulate a binary integer programming model (BIPM) to find the similarity of S and T on finding a minimum distances between them which depends on the number of similar characters, such that:

$$\text{Min} \sum_i \sum_j d_{S_i T_j} x_{S_i T_j} \quad , \qquad \ldots(7.1)$$

$$\text{subject to} \quad x_{S_i T_j} = T_j \, , \qquad \ldots(7.2)$$

for $i = 1,\ldots,m$ and $j = 1,\ldots,n$, $S_i \in S$ and $T_j \in T$ (characters)

$$\text{Where } x_{S_i T_j} = \begin{cases} 1 & \text{if } x_{S_i T_j} \text{ has been taken (branched)} \\ 0 & \text{otherwise} \end{cases} \qquad \ldots(7.3)$$

And

$$\begin{cases} 0 & \text{if } S_i = T_j \end{cases}$$

$d_{S_i T_j} =$ …(7.4)

1 otherwise

3.3. Proposed algorithm to solve BIPM:

Step 1: (Starting) We choose the first node (N0) to be the variable $x_{S_m T_n}$ and also to be the first comparison between the character S_m from the string S and the character T_n from the string T with the equation $x_{S_m T_n} = T_n$ and distance $d_{S_m T_n}$.

Step 2: (Matching) If S_m matched (similar to) T_n then branch the node (N0) to the node (N1) which represent the next variable $x_{S_{m-1} T_{n-1}}$ and to be a new comparison between S_{m-1} and T_{n-1} with the equation $x_{S_{m-1} T_{n-1}} = T_{n-1}$ and distance $d_{S_{m-1} T_{n-1}}$, else:

Step 3: (Mismatching) S_m mismatched (not similar to) T_n then branch the node (N0) to the node (N1, N2 and N3) which represent the next following variables to be a new comparisons:

i) N1: $x_{S_m T_{n-1}}$ with the equation $x_{S_m T_{n-1}} = T_{n-1}$ and distance $d_{S_n T_{n-1}}$,

ii) N2: $x_{S_{m-1} T_{n-1}}$ with the equation $x_{S_{m-1} T_{n-1}} = T_{n-1}$ and distance $d_{S_{m-1} T_{n-1}}$,

iii) N3: $x_{S_{m-1} T_n}$ with the equation $x_{S_{m-1} T_n} = T_n$ and distances $d_{S_{m-1} T_n}$.

If one of the nodes (N1, N2 or N3) has a matched characters (as step 2) then *we need not* to branch the other two nodes.

Step 4: (Exception) If $m = n = 2$ then we directly branch the node with the equation $x_{S_2 T_2} = T_2$ and distance $d_{S_2 T_2}$ into the node with the equation $x_{S_1 T_1} = T_1$ even if S_2 is not similar to T_2 .

Step 5: (Ending) Repeat the process in steps 2 and 3 until reaching to the last variable $x_{S_1 T_1}$ with the equation $x_{S_1 T_1} = T_1$ and distance $d_{S_1 T_1}$.

By this algorithm we get a tree of nodes with the optimal solution of the Edit Distance Problem which represent the minimum number of edit operations to transform the string S into the string T with finding at least one of the optimal transformations paths.

3.4 Solving Example 2.4 according to BIPM:

We have two strings S and T, such that:

S:

S_1	S_2	S_3	S_4	S_5	S_6	S_7
V	I	N	T	N	E	R

T:

T_1	T_2	T_3	T_4	T_5	T_6	T_7
W	R	I	T	E	R	S

with length $|S| = m = 7$ and $|T| = n = 7$

By the algorithm with step 1, the first branched node (N0) is carry the variable $x_{S_7T_7}$, and by steps 2 and 3 we continue by branching the nodes, such that:

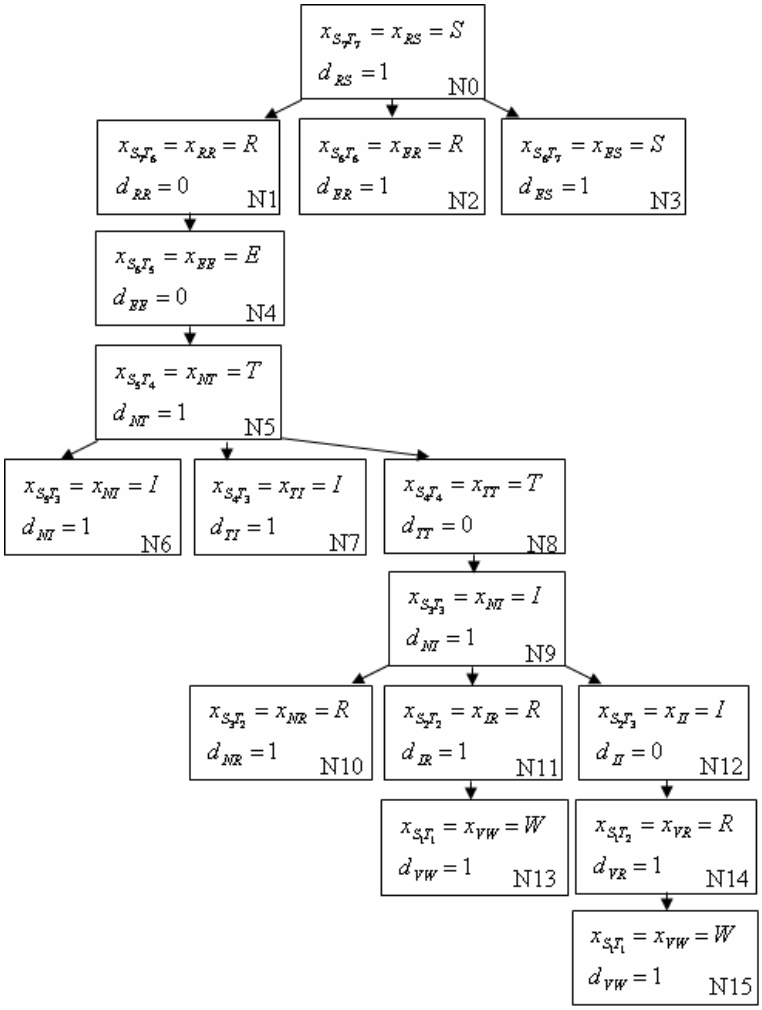


Figure (3.1)

Step 4 of the algorithm occurs in node (N11), although I is not similar to R but is branched only to one node (N13).

By following the path of the nodes: N15-N14-N13-N12-N9-N8-N5-N4-N1-N0, we have the value of the objective function:

$d_{VW}x_{VW} + d_{VR}x_{VR} + d_{II}x_{II} + d_{NI}x_{NI} + d_{TT}x_{TT} + d_{NT}x_{NT} + d_{EE}x_{EE} + d_{RR}x_{RR} + d_{RS}x_{RS}$

= 1 (1) + 1 (1) + 0 (1) + 1 (1) + 0 (1) + 1 (1) + 0 (1) + 0 (1) + 1 (1)

= 5

and we have two transformation of VINTNER to WRITERS (by fixing first the similar characters in the nodes: N1, N4, N8 and N12) such that:

V	—	I	N	T	N	E	R	—
W	R	I	—	T	—	E	R	S

OR,

—	V	I	N	T	N	E	R	—
W	R	I	—	T	—	E	R	S

And by following the path of the nodes: N13-N11-N9-N8-N5-N4-N1-N0 we have the value of the objective function:

$d_{VW}x_{VW} + d_{IR}x_{IR} + d_{NI}x_{NI} + d_{TT}x_{TT} + d_{NT}x_{NT} + d_{EE}x_{EE} + d_{RR}x_{RR} + d_{RS}x_{RS}$ =

1 (1) + 1 (1) + 1 (1) + 0 (1) + 1 (1) + 0 (1) + 0 (1) + 1 (1) = 5

and we have two transformation of VINTNER to WRITERS (by fixing first the similar characters in the nodes: N1, N4 and N8) such that:

V	I	N	T	N	E	R	—
W	R	I	T	—	E	R	S

4. Computing DNA Sequence Alignment:

4.1. DNA Sequence Alignment Problem:

Given two strings S and T with length m and n such that S=$(S_1,...,S_m)$ and T=$(T_1,...,T_n)$ respectively, the sequence alignment is defined as an assignment of gaps to positions $0,...m$ in S and $0,...n$ in T so as to line up each character in one sequence with either a character or a gap in the other sequence and this will result in two sequences of equal length.

An alignment between two input sequences, S and T over the alphabet Σ = {A,C,G,T}, expresses an equivalence relationship between the pair of sequences by generating two sequences S' and T' of equal length by inserting the gaps into S and T.

An optimal alignment is one that minimizes the number of the gaps are inserted while simultaneously minimizing the number of replacements operation occur (i.e. a C aligned with a T).

Gaps are an important concept in biological applications because a stream of gaps in a DNA sequence my present a significant biological characteristic, gaps usually incur a **penalty** to the potential alignment score between two sequences. If we remove the gaps from S' and T' then we restored S and T.

To illustrate the idea of an alignment, consider the sequences given by S=TAAGAAC and T= TGAC. They could have an optimal alignment consisting of S' and T' below:

S'= T A A G A A C

T'= T — — G — A C

Given the scoring functions of one for a matching cost and zero for gaps and replacement cost, the above alignment would have a score of four. Note that other alignments can share the maximal score:

S'= T A A G A A C

T'= T — — G A — C

And other sub-optimal alignments can have lower scores (3 in the following instance):

S'= T A A G A A C

T'= — — T G — A C

See ([1] & [13]).

4.2. Dot Plot Problem:

Dot Plot Problem is one of the earliest methods of comparing two DNA Sequences Alignment which plots the regions of the similarity between them by hand.

Dot Plot Problem is to create a table by setting one DNA Sequence on a vertical axis and the other on a horizontal axis and the dots mark a match between nucleotides in the sequences. [5]

Table (4.1)

	A	G	C	T	A	G	A	G	A
A	\				·		·		·
G		\				·		·	
C			\						
A	·				·		·		·
T				\					
A	·				\		·		·
G		·				\		·	
G		·				·		\	
A	·				·		·		\

In the table (3.1), the sequence AGCATAGGA is matched against the sequence AGCTAGAGA, the regions of similarity occur where it is clear that there is a **string of diagonal dots** in the dot plot, so we can easily compute the similarity by setting the value (2) in each dot in the three diagonal strings and the value (-1) in the two gaps (i.e. (A,T) and (G,A)) and obtain:

Optimal Similarity : A G C + (A,T) + T A G + (G,A) + G A
2+2+2 + (-1) + 2+2+2 + (-1) + 2+2 = 14

4.3. DNA Sequence Alignment Formula:

DNA Sequence Alignment Problem is an alignment present in optimal path between the point $E(0,0)$ and the point $E(m,n)$ and Dynamic Programming used to solve it in the following formula:

Given two string S and T with length m and n respectively, our goal is to compute the optimal sequence alignment of S and T.

Let $V(i, j)$ defined as the value of the alignment of the strings S= $(S_1,...,S_m)$ and T= $(T_1,...,T_n)$.

$V(m,n)$ is the optimal sequence alignment of S and T, in order to compute $V(i, j)$ we must have an initial conditions: See ([7], [9] & [13])

$$V(0,0) = 0$$

$$V(i,0) = V(i-1,0) + \sigma(S(i),-), \qquad for 1 \le i \le m$$

$$V(0, j) = V(0, j-1) + \sigma(-,T(j)), \qquad for 1 \le j \le n$$

And recurrence relation used for $1 \le i \le m, 1 \le j \le n$ defined as:

$$V(i,j) = max \begin{cases} V(i-1,j) + \sigma(S(i),-) \\ V(i,j-1) + \sigma(-,T(j)) \\ V(i-1,j-1) + \sigma(S(i),T(j)) \end{cases} \qquad \ldots(8)$$

Where

$$\left.\begin{array}{l}\left.\begin{array}{ll}\sigma(S(i),-)=-1 & (S(i)\text{ up to a gap}),\\ \sigma(-,T(j))=-1 & (T(j)\text{ up to a gap}),\end{array}\right\}\text{ Gap penalty}\\ \sigma(S(i),T(j))=\begin{cases}2 \text{ if } S(i)=T(j)\ (S(i)\text{ match }T(j)\text{ in same character})\\ -1 \text{ if } S(i)\neq T(j)\ (S(i)\text{ match }T(j)\text{ in different character})\end{cases}\end{array}\right\}\ldots(9)$$

4.4. Example:

The table (4.2) gives the transformation of the DNA sequence alignment AACTGGTACC to TTCACGGCA using Dynamic Programming Problem:

Table (4.2)

$V(i,j)$		0	1 T	2 T	3 C	4 A	5 C	6 G	7 G	8 C	9 A
0		0	-1	-2	-3	-4	-5	-6	-7	-8	-9
1	A	-1	-1	-2	-3	-1	-2	-3	-4	-5	-6
2	A	-2	-2	-2	-3	-1	-2	-3	-4	-5	-3
3	C	-3	-3	-3	0	-1	1	0	-1	-2	-3
4	T	-4	-1	-1	-1	-1	0	0	-1	-2	-3
5	G	-5	-2	-2	-2	-2	-1	2	2	1	0
6	G	-6	-3	-3	-3	-3	-2	1	4	3	2
7	T	-7	-4	-1	-2	-3	-3	0	3	3	2
8	A	-8	-5	-2	-2	0	-1	-1	2	2	5
9	C	-9	-6	-3	0	-1	2	1	1	4	4
10	C	-10	-7	-4	-1	-1	1	1	0	3	3

We chose an arbitrary edit distance calculations from table (2.1) such that:

$$V(2,4)=\max\begin{cases}V(1,4)-1=-1-1=-2\\ V(2,3)-1=-3-1=-4\\ V(1,3)-1=-3+2=-1\end{cases}\quad=-1$$

$$V(7,7)=\max\begin{cases}V(6,7)-1=4-1=3\\ V(7,6)-1=0-1=-1\\ V(6,6)-1=1-1=0\end{cases}\quad=3$$

From the table (4.2) we got the terminal value $V(10,9)=3$ which gives the optimal sequence alignment of the two strings, such that:

$$V(10,9)=\max\begin{cases} V(9,9)-1=4-1=3 \\ V(10,8)-1=3-1=2 \\ V(9,8)-1=4-1=3 \end{cases}$$

A MATLAB code presented in section 6 and has been used to solve the DNA Sequence Alignment Problem with function (`SeqAlign`).

The recovering of the paths from $opt[m][n]$ into $opt[0][0]$ according to the Dynamic Programming formula for DNA Sequence Alignment in (eqs. (8) & (9)) with depending on the following three possibilities:

I. If a character $S(i)$ up to $T(j)$ for $1 \le i \le m,\ 1 \le j \le n$, then:

$$opt[i-1][j-1]=opt[i][j]-2 \quad \text{if} \quad S(i)=T(j) \qquad ...(10)$$

$$opt[i-1][j-1]=opt[i][j]+1 \quad \text{if} \quad S(i)\neq T(j) \qquad ...(11)$$

II. If a character $S(i)$ up with a gap for $1 \le i \le m,\ 0 \le j \le n$, then:

$$opt[i-1][j]=opt[i][j]+1 \qquad \ldots(12)$$

III. If a character $T(j)$ up with a gap for $1 \le j \le n,\ 0 \le i \le m$, then:

$$opt[i][j-1]=opt[i][j]+1 \qquad \ldots(13)$$

And by the (eqs. (10), (11), (12) and (13)) the recovering of the optimal paths from $opt[10][9]$ into $opt[0][0]$ and according to the inversion of the arrows in the shaded squares we get:

	OR
$opt[10][9]+1=4=opt[9][8]$	$opt[10][9]+1=4=opt[9][9]$
$opt[9][8]-2=2=opt[8][7]$	$opt[9][9]+1=5=opt[8][9]$
$opt[8][7]+1=3=opt[7][7]$	$opt[8][9]-2=3=opt[7][8]$
$opt[7][7]+1=4=opt[6][7]$	$opt[7][8]+1=4=opt[6][7]$
$opt[6][7]-2=2=opt[5][6]$	$opt[6][7]-2=2=opt[5][6]$
$opt[5][6]-2=0=opt[4][5]$	$opt[5][6]-2=0=opt[4][5]$
$opt[4][5]+1=1=opt[3][5]$	$opt[4][5]+1=1=opt[3][5]$
$opt[3][5]-2=-1=opt[2][4]$	$opt[3][5]-2=-1=opt[2][4]$
$opt[2][4]-2=-3=opt[1][3]$	$opt[2][4]-2=-3=opt[1][3]$
– – – – – – – – – – – – – – – – –	– – – – – – – – – – – – – – – – –
$opt[1][3]+1=-2=opt[1][2]$	$opt[1][3]+1=-2=opt[1][2]$
$opt[1][2]+1=-1=opt[1][1]$	$opt[1][2]+1=-1=opt[1][1]$
$opt[1][1]+1=0=opt[0][0]$	$opt[1][1]+1=0=opt[0][0]$
OR – – – – – – – – – – – – – – – –	*OR* – – – – – – – – – – – – – – – –
$opt[1][3]+1=-2=opt[0][2]$	$opt[1][3]+1=-2=opt[0][2]$
$opt[0][2]+1=-1=opt[0][1]$	$opt[0][2]+1=-1=opt[0][1]$
$opt[0][1]+1=0=opt[0][0]$	$opt[0][1]+1=0=opt[0][0]$
OR – – – – – – – – – – – – – – – –	*OR* – – – – – – – – – – – – – – – –
$opt[1][3]+1=-2=opt[1][2]$	$opt[1][3]+1=-2=opt[1][2]$
$opt[1][2]+1=-1=opt[0][1]$	$opt[1][2]+1=-1=opt[0][1]$
$opt[0][1]+1=0=opt[0][0]$	$opt[0][1]+1=0=opt[0][0]$

Thus, according to the recovering of the optimal paths, the similarity of the two sequence will be in form:

S'= A — — A C T G G T A C C

T'= T T C A C — G G — — C A

OR,

S'= — — A A C T G G T A C C

T'= T T C A C — G G — — C A

OR,

S'= — A — A C T G G T A C C

T'= T T C A C — G G — — C A

OR,

S'= A — — A C T G G T A C C

T'= T T C A C — G G C A — —

OR,

S'= — — A A C T G G T A C C

T'= T T C A C — G G C A — —

OR,

S'= — A — A C T G G T A C C

T'= T T C A C — G G C A — —

Respectively.

All the paths of transformation the DNA sequences alignment AACTGGTACC to TTCACGGCA are optimal because they have the same number of gaps (five gaps) and the same length (12). See ([1] & [11]).

4.5. Solving Example 4.4 according to BIPM:

Before going to compute the optimal sequence alignment of the DNA sequences alignment AACTGGTACC and TTCACGGCA, we need to re-formulate the binary integer programming model in equations (7.1) and (7.4) to be suitable to find that sequence, as follows:

$$\text{Max} \sum_i \sum_j d_{S_i T_j} x_{S_i T_j} \qquad \ldots(14.1)$$

and

$$d_{S_i T_j} = \begin{cases} 2 & \text{if} \quad S_i = T_j \\ -1 & \text{otherwise} \end{cases} \qquad \ldots(14.2)$$

Now, we have two DNA sequences strings S and T, such that:

S:

S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9	S_{10}
A	A	C	T	G	G	T	A	C	C

T:

T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9
T	T	C	A	C	G	G	C	A

with length $|S| = m = 10$ and $|T| = n = 9$

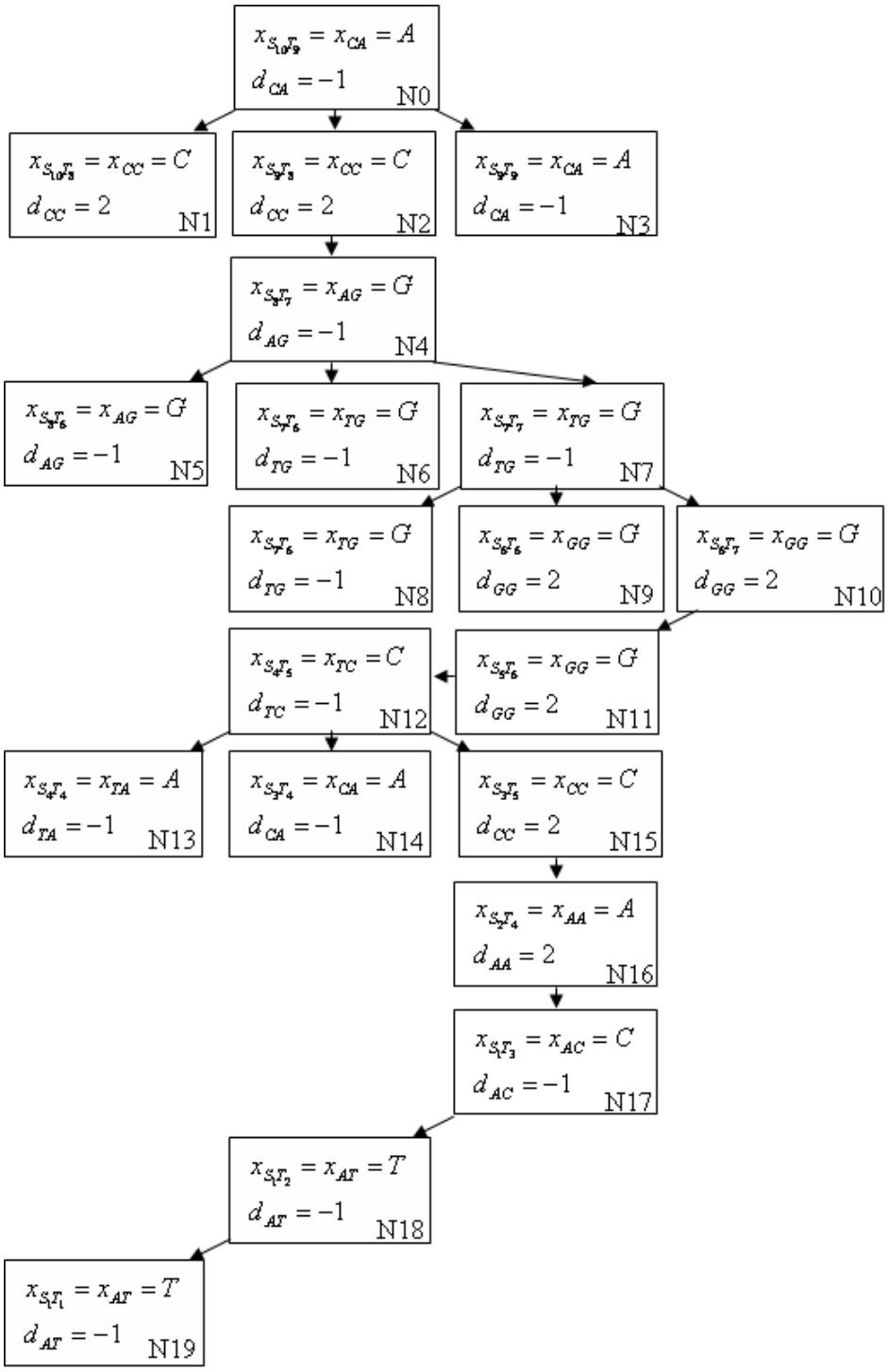

$x_{S_{10}T_9} = x_{CA} = A$
$d_{CA} = -1$
N0
$x_{S_{10}T_3} = x_{CC} = C$
$d_{CC} = 2$
N1
$x_{S_9T_3} = x_{CC} = C$
$d_{CC} = 2$
N2
$x_{S_9T_9} = x_{CA} = A$
$d_{CA} = -1$
N3
$x_{S_9T_7} = x_{AG} = G$
$d_{AG} = -1$
N4
$x_{S_9T_6} = x_{AG} = G$
$d_{AG} = -1$
N5
$x_{S_8T_6} = x_{TG} = G$
$d_{TG} = -1$
N6
$x_{S_8T_7} = x_{TG} = G$
$d_{TG} = -1$
N7
$x_{S_8T_6} = x_{TG} = G$
$d_{TG} = -1$
N8
$x_{S_6T_6} = x_{GG} = G$
$d_{GG} = 2$
N9
$x_{S_6T_7} = x_{GG} = G$
$d_{GG} = 2$
N10
$x_{S_4T_5} = x_{TC} = C$
$d_{TC} = -1$
N12
$x_{S_5T_6} = x_{GG} = G$
$d_{GG} = 2$
N11
$x_{S_4T_4} = x_{TA} = A$
$d_{TA} = -1$
N13
$x_{S_3T_4} = x_{CA} = A$
$d_{CA} = -1$
N14
$x_{S_3T_5} = x_{CC} = C$
$d_{CC} = 2$
N15
$x_{S_2T_4} = x_{AA} = A$
$d_{AA} = 2$
N16
$x_{S_1T_3} = x_{AC} = C$
$d_{AC} = -1$
N17
$x_{S_1T_2} = x_{AT} = T$
$d_{AT} = -1$
N18
$x_{S_1T_1} = x_{AT} = T$
$d_{AT} = -1$
N19

Figure (4.1)

By following the path of the nodes: N19-N18-N17-N16-N15-N12-N11-N10-N7-N4-N2-N0 we have the value of the objective function:

$$d_{AT}x_{AT} + d_{AT}x_{AT} + d_{AC}x_{AC} + d_{AA}x_{AA} + d_{CC}x_{CC} + d_{TC}x_{TC} + d_{GG}x_{GG} + d_{GG}x_{GG} + d_{TG}x_{TG} + d_{AG}x_{AG} + d_{CC}x_{CC} + d_{CA}x_{CA}$$

$$=-1\ (1)\ + (-1)\ (1)\ + (-1)\ (1)\ + 2\ (1)\ + 2\ (1)\ + (-1)\ (1)\ + 2\ (1)\ + 2\ (1)\ + (-1)\ (1) + (-1)\ (1) + 2\ (1) + (-1)\ (1) = 3$$

and we have three transformation of AACTGGTACC to TTCACGGCA (by fixing first the similar characters in the nodes: N2, N10, N11, N15 and N16) such that:

S'= A — — A C T G G T A C C

T'= T T C A C — G G — — C A

OR,

S'= — — A A C T G G T A C C

T'= T T C A C — G G — — C A

OR,

S'= — A — A C T G G T A C C

T'= T T C A C — G G — — C A

5. Conclusions:

With the binary integer programming model proposed for computing DNA Sequence Alignment some important conclusions were reached:

I. The model presented allows to obtain good and effective results for the Edit Distance and DNA Sequence Alignment Problems as the results of applying the Dynamic Programming Problem.

II. The binary model solved by exact algorithm for computing the problems and obtaining most of the optimal paths with least number of gaps and shortest length and within a very reasonable computational time in handly solution than the Dynamic Programming which needs $O(mn)$ time. See ([4] & [7])

III. The binary model proved it's efficient to solve a wide of real-life problems and gives a very good solutions and one of these problems is computing the string sequence and it's applications in molecular biology.

6. MATLAB codes of Dynamic Programming Problem: See [3]

6.1. For Edit Distance Problem:

```
function e = EditDist(s1,s2)
%EditDist Finds the Edit Distance between strings s1 and s2.
Uses an
% efficient dynamic programming algorithm.
DelCost = 1;
InsCost = 1;
ReplCost = 1;
[m1,n1] = size(s1);
[m2,n2] = size(s2);
%Initialize dynamic matrix E with appropriate size:
E = zeros(n1+1,n2+1);
%This is dynamic programming algorithm:
for i = 1:n1
   E(i+1,1) = E(i,1) + DelCost;
end;
for j = 1:n2
   E(1,j+1) = E(1,j) + InsCost;
end;
for i = 1:n1
   for j = 1:n2
      if s1(i) == s2(j)
         Repl = 0;
      else
         Repl = ReplCost;
      end;
      E(i+1,j+1)    =    min([E(i,j)+Repl    E(i+1,j)+DelCost
E(i,j+1)+InsCost]);
   end;
end;
E(n1+1,n2+1)    =    min([E(n1,n2)+Repl    E(n1+1,n2)+DelCost
E(n1,n2+1)+InsCost])
e = E(n1+1,n2+1);
```

The Result:

```
>> EditDist('VINTNER','WRITERS')

E =
     0     1     2     3     4     5     6     7
     1     1     2     3     4     5     6     7
     2     2     2     2     3     4     5     6
     3     3     3     3     3     4     5     6
     4     4     4     4     3     4     5     6
     5     5     5     5     4     4     5     6
     6     6     6     6     5     4     5     6
     7     7     6     7     6     5     4     5

ans =
     5
```

6.2. For DNA Sequence Alignment Problem:

```
function v = SeqAlign(dna1,dna2)
%SeqAlign Finds the Sequence Alignment between two DNA
strings dna1
%              dna2. Uses an efficient dynamic programming
algorithm.
DelCost = -1;
InsCost = -1;
ReplCost = -1;
[m1,n1] = size(dna1);
[m2,n2] = size(dna2);
%Initialize dynamic matrix V with appropriate size:
V = zeros(n1+1,n2+1);
%This is dynamic programming algorithm:
for i = 1:n1
   V(i+1,1) = V(i,1) + DelCost;
end;
for j = 1:n2
   V(1,j+1) = V(1,j) + InsCost;
end;
for i = 1:n1
   for j = 1:n2
      if dna1(i) == dna2(j)
         Repl = 2;
      else
         Repl = ReplCost;
      end;
      V(i+1,j+1)    =    max([V(i,j)+Repl    V(i+1,j)+DelCost
V(i,j+1)+InsCost]);
   end;
end;
V(n1+1,n2+1)    =    max([V(n1,n2)+Repl    V(n1+1,n2)+DelCost
V(n1,n2+1)+InsCost])
v = V(n1+1,n2+1);
```

The Result:

```
>> SeqAlign('AACTGGTACC','TTCACGGCA')
V =
     0    -1    -2    -3    -4    -5    -6    -7    -8    -9
    -1    -1    -2    -3    -1    -2    -3    -4    -5    -6
    -2    -2    -2    -3    -1    -2    -3    -4    -5    -3
    -3    -3    -3     0    -1     1     0    -1    -2    -3
    -4    -1    -1    -1    -1     0     0    -1    -2    -3
    -5    -2    -2    -2    -2    -1     2     2     1     0
    -6    -3    -3    -3    -3    -2     1     4     3     2
    -7    -4    -1    -2    -3    -3     0     3     3     2
    -8    -5    -2    -2     0    -1    -1     2     2     5
    -9    -6    -3     0    -1     2     1     1     4     4
   -10    -7    -4    -1    -1     1     1     0     3     3
ans =
     3
```

REFERENCES

[1] Batzoglou, S., "*Sequence Alignment*", lecture #2, CS262, Stanford Computer Science, USA, 2006. URL: http://ai.stanford.edu/~serafim/CS262_2005/LectureNotes/Lecture2.pdf

[2] Bruce, K., "*Dynamic Programming and Minimum String Edit Distance*", lecture #4, Natural Language Processing CS 181, Pomona College, USA, 2008.URL: http://www.cs.pomona.edu/classes/cs181NLP/lectures/Lec4/Lec4.pdf

[3] Castro, M., " *MATLAB Code for Edit Distance*", official website of MATLAB, 2000. URL: http://www.mathworks.com/matlabcentral/files/213/EditDist.m

[4] Chan, T., "*Practical Linear Space Algorithms for Computing String-Edit Distances*", Springer Berlin / Heidelberg Publishing, Germany, pp. 504-513, 2006.URL: http://www.springerlink.com/content/b7h5840706130846/fulltext.pdf

[5] Church, K. and Helfman, J., "*Dotplot: a Program for Exploring Self-Similarity in Millions of Lines of Text and Code*". Journal of Computational and Graphical Statistics, 2(2):153-174, 1993.URL: http://www.imagebeat.com/dotplot/rp.jcgs.pdf

[6] Gilleland, M., "*Levenshtein Distance , in Three Flavors*", Merriam Park Software,USA, 2007. URL: http://www.merriampark.com/ld.htm

[7] Gusfield, D., "*Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*", Cambridge University Press, 1st edition, UK, 1997.

[8] Hillier, F. and Lieberman, G., "*Introduction to Operation Research*", McGraw-Hill Press, 7th edition, Holden Day, New York, USA, 2001.

[9] Jones, N. and Pevzner, P., "*Introduction to Bioinformatics Algorithms*", MIT Press, 1st edition, Cambridge, Massachusetts, USA, 2004.

[10] Levenshtein, V., "*Binary codes capable of correcting deletions, insertions and reversals*", Soviet Physics Doklady 10(8) p707-710, 1966.

[11] Rouchka, E., "*Aligning DNA Sequences Using Dynamic Programming*",Crossroads, the ACM student magazine, Xrds13-1, 2006.URL: http://www.acm.org/crossroads/xrds13-1/dna.html

[12] Royce, T. and Necaise, R., "*A Parallel Algorithm for DNA Alignment*",Crossroads, the ACM student magazine, Xrds9-3, 2003. URL: http://www.acm.org/crossroads/xrds9-3/alignment.html

[13] Tompa, M., "*Lecture Notes on Biological Sequence Analysis*", Technical Report #2000-06-01, University of Massachusetts Lowell, USA, 2000.URL: http://www.cs.uml.edu/bioinformatics/resources/lectures/tompa00lecture.pdf