Toms Global Alignment Algorithm

The currently used major algorithms for aligning biological sequences (protein and nucleic acid sequences) stem from the pioneering work of Needleman and Wunsch (1970). This algorithm performs optimal matching alignments with a time complexity of **O(m*n)** and a space complexity of **O(m*n)**, where **m** – length of shortest sequence to be aligned and **n** – length of longest sequence to be aligned. Toms algorithm is modification of Needleman-Wunsch algorithm with a time complexity of **O(m)** and a space complexity of **O(n)**. It is not only fast for finding optimal alignments but it is also super fast for finding close to optimal alignments.

Sequence alignment or sequence comparison lies at heart of the bioinformatics, which describes the way of arrangement of DNA/RNA or protein sequences, in order to identify the regions of similarity among them. It is used to infer structural, functional and evolutionary relationship between the sequences. Alignment finds similarity level between query sequence and different database sequences. The algorithm works by dynamic programming approach, which divides the problem into smaller independent sub problems. It finds the alignment more quantitatively by assigning scores.

Before going into Toms algorithm specifics, we will discuss in details the Needleman-Wunsch algorithm, which is appropriate for finding the best alignment of two sequences that are similar in length and similar across their entire lengths.

Then we will introduce three very simple modifications, which will allow accelerate algorithm, reduce memory usage and allow align files with length up to **1,000,000,000**.

Needleman-Wunsch algorithm

In possible alignment we can see that letters may match, mismatch, be deleted or inserted:

- Match: The two letters are the same
- Mismatch: The two letters are differential
- Insertion or Deletion (indel): One letter aligns to a gap in the other string.

There are various ways to score these three scenarios

The simple basic scoring schema can be assumed as, if two residues (nucleotide or amino acid) at i^{th} and j^{th} position are same, matching score is **2** (**S(i,j) = 2**) or if the two residues at i^{th} and j^{th} position are not same, mismatch score is assumed as **-3** (**S(i,j) = -3**). The linear gap score (**g**) or linear gap penalty is assumed as **-2**.

Gap score is defined as penalty given to alignment, when we have insertion or deletion.

The dynamic programming matrix is defined with three different steps.
1. Initialization of the matrixes with initial scores and back pointers.
2. Matrix filling with maximum scores and back pointers.
3. Using trace back matrix trace back the residues for appropriate alignment.

To study the algorithms, consider the two given DNA sequences: ACGCATCA (sequence **a**), ACTGATTCA (sequence **b**).

The length (count of the nucleotides or amino acids) of the sequence **a** and sequence **b** are: **m = 8** and **n = 9** respectively. Two initial matrixes (first – scoring matrix **M** and second – trace back matrix **N**) are created with **m + 1** row's and **n + 1** column's (where **m** and **n** corresponds to length of the sequences). Extra row and column is given, so as to align with gap at the starting of the sequences as shown in Figure 1.

Scoring Matrix

	-	A	C	T	G	A	T	T	C	A
-										
A										
C										
G										
C										
A										

T										
C										
A										

Trace back matrix

	-	A	C	T	G	A	T	T	C	A
-										
A										
C										
G										
C										
A										
T										
C										
A										

Figure 1. Initial matrixes

1. Initialization of score and trace back matrixes

Scoring matrix initialization:

M(0, 0) = 0
M(i, 0) = (i – 1) * g, for **i = 1, … , m**
M(0, j) = (j – 1) * g, for **j = 1, … , n**

Trace back matrix initialization:

N(0, 0) = "done"
N(i, 0) = "j", for **i = 1, … , m**
N(0, j) = "i", for **j = 1, … , n**

Given there is no 'top' or 'top-left' cells for the first row we can only add from the existing cell to the left. Hence we add **-2** for each shift to the right as this represents an insertion or deletion from the previous score. This results in the score matrix first row being **0, -2, -4, -6, -8, -10, -12, -14, -16, -18**. The first row in trace back matrix will be: **"end", "i", "i", "i", "i", "i", "i", "i", "i", "i".** Here we use character **"i"** for direction code **"left"**. The same applies to the first column as we only have existing scores above. In first column of trace back matrix we use character **"j"** for direction **"up"**.

Thus we have (Figure 2):

Scoring Matrix

	-	A	C	T	G	A	T	T	C	A
-	0	-2	-4	-6	-8	-10	-12	-14	-16	-18
A	-2									
C	-4									
G	-6									
C	-8									
A	-10									
T	-12									
C	-14									
A	-16									

Trace back matrix

	-	A	C	T	G	A	T	T	C	A
-	done	i	i	i	i	i	i	i	i	i
A	j									
C	j									
G	j									

C	j									
A	j									
T	j									
C	j									
A	j									

Figure 2. Initialization of matrixes

2. Calculation of scores and filling the trace back matrix

The second and crucial step of the algorithm is calculation of scores and filling the trace back matrix. To find the maximum score $M_{(i,j)}$ of each cell, it is required to know the neighboring scores (northwest diagonal, left and up) of the current position. From the assumed values, add the match or mismatch (assumed) score to the diagonal value. Similarly add the gap score to the other neighboring values. Thus, we can obtain three different values, from that take the maximum among them and fill the $\mathbf{i}^{th}$ and $\mathbf{j}^{th}$ position with the score obtained. Simultaneously we get the score matrix we compute the trace back matrix which helps us to process deduction of the best alignment.

Overall the equations can be showed in the following manner:

$$\mathbf{M(i, j) = Maximum[M(i\text{-}1, j\text{-}1) + S(i, j), M(i, j\text{-}1) + g, M(i\text{-}1, j) + g]}$$

$$\mathbf{N(i, j) = \text{“d” if } M(i, j) = M(i\text{-}1, j\text{-}1) + S(i, j);}$$
$$\mathbf{\text{“i” if } M(i, j) = M(i, j\text{-}1) + g}$$
$$\mathbf{\text{“j” if } M(i, j) = M(i\text{-}1, j) + g;}$$

for $\mathbf{i = 1, \ldots, m}$ and for $\mathbf{j = 1, \ldots, n}$

Formula 1.

Here **M(i, j)** is the score located in row **i** and column **j** of scoring matrix **M** and **N(i, j)** is the direction located in row **i** and column **j** of trace back matrix **N**.

To score the matrix of the current position (the first position **M(1, 1)**) the above stated formula can be used. The first residue (nucleotides) in the two sequences are **‘A’** and **‘A’**. Since they are matching residues, the score would (**S(1, 1)= 2**) be **2**.

$$\mathbf{M(1, 1) = Maximum[M(0, 0) + S(1, 1), M(1, 0) + g, M(0, 1) + g] =}$$

$$\mathbf{Maximum[0 + (2), (-2) + (-2), (-2) + (-2)] =}$$

$$\mathbf{Maximum[2, -4, -4] = 2}$$

N(1, 1) = “d” (because **M(1, 1) = M(0, 0) + S(1, 1)**)

The obtained score **2** is placed in position **i,j** (**1,1**) of the scoring matrix. Because we get maximum from element **M(0, 0)** we place value “**d**” (code for direction “diagonal”) to appropriate position **i,j** (**1,1**) of the trace back matrix.

Similarly using the above equations, we fill all the remaining rows and columns of scoring matrix. Simultaneously we place the back pointers to the cells of trace back matrix from where the maximum score is obtained, which are predecessors of the current cell (Figure 3).

Scoring Matrix

	-	A	C	T	G	A	T	T	C	A
-	0	-2	-4	-6	-8	-10	-12	-14	-16	-18
A	-2	2	0	-2	-4	-6	-8	-10	-12	-14

C	-4	0	4	2	0	-2	-4	-6	-8	-10
G	-6	-2	2	1	4	2	0	-2	-4	-6
C	-8	-4	0	-1	2	1	-1	-3	0	-2
A	-10	-6	-2	-3	0	4	2	0	-2	2
T	-12	-8	-4	0	-2	2	6	4	2	0
C	-14	-10	-6	-2	-3	0	4	3	6	4
A	-16	-12	-8	-4	-5	-1	2	1	4	8

Trace back matrix

	-	A	C	T	G	A	T	T	C	A
-	done	i	i	i	i	i	i	i	i	i
A	j	d	i	i	i	d	i	i	i	d
C	j	j	d	i	i	i	i	i	d	i
G	j	j	j	d	d	i	i	i	i	i
C	j	j	d	d	j	d	d	d	d	i
A	j	d	j	d	j	d	i	i	i	d
T	j	j	j	d	i	j	d	d	i	i
C	j	j	d	j	d	j	j	d	d	i
A	j	d	j	j	d	d	j	d	j	d

Figure 3. Matrixes filling with scores and back pointers

3. Deducing the alignment from the trace back matrix

The final step in the algorithm is the trace back for the best alignment:

1. Trace back is the process of deduction of the best alignment from the trace back matrix.
2. Trace back always begins with the last cell, which is the bottom right cell. In the above-mentioned example, one can see the bottom right cell code **d**.
3. It moves according to the trace back value written in the cell.
4. There are 3 possible traversals, **d** - diagonal, **i** - left or **j** - up.
5. Trace back completion happens when the top-left cell becomes “end”.

Best alignment:

1. Alignment is deduced from the values of cells, which are through the trace back path, by taking into account the values of the cell that are in trace back matrix.
2. In trace back matrix, If the value is **d** (“diagonal”), the letters from the two sequences are aligned. If it is **i** (“left”), a Gap is introduced in the left sequence, and if it is **j** (“up”), a gap is introduced in the top sequence, and always obtained sequences are aligned backwards. (Figure 4).

Trace back matrix

	-	A	C	T	G	A	T	T	C	A
-	done	i	i	i	i	i	i	i	i	i
A	j	d	i	i	i	d	i	i	i	d
C	j	j	d	i	i	i	i	i	d	i
G	j	j	j	d	d	i	i	i	i	i
C	j	j	d	d	j	d	d	d	d	i
A	j	d	j	d	j	d	i	i	i	d
T	j	j	j	d	i	j	d	d	i	i
C	j	j	d	j	d	j	j	d	d	i
A	j	d	j	j	d	d	j	d	j	d

Best alignment

```
AC-GCA-TCA
||  |  |  |||
ACTG-ATTCA
```

Figure 4. The best alignment with trace backing

Alignment with affine gap scores

The alternative to using linear gap score is to use an affine gap score

$$g = o + l * e,$$

with negative **o** the gap-open score and negative **e** the gap-extension score.

Instead of using one score matrix **M(i, j)** to represent the best score **S(i, j)** attainable up to **a(i)** and **b(j)**, we will now use three matrices **M**, **X** and **Y**:

1. **M(i, j)** is the best score up to **(i, j),** given that **a(i)** is aligned to **b(j)**,
2. **X(i, j)** is the best score up to **(i, j)**, given that **a(i)** is aligned to a gap, and
3. **Y(i, j)** is the best score up to **(i, j)**, given that **b(j)** is aligned to a gap.

Scoring matrixes initialization:

$$M(0, 0) = 0, X(0, 0) = Y(0, 0) = -\infty$$
$$X(0, j) = o + (j - 1) * e, M(0, j) = X(0, j) = -\infty, \text{ for } j = 1, \ldots, n$$
$$Y(i, 0) = o + (i - 1) * e, M(i, 0) = Y(i, 0) = -\infty, \text{ for } i = 1, \ldots, m$$

Trace back matrix initialization:

$$N(0, 0) = \text{“done”}$$
$$N(i, 0) = \text{“j”}, \quad \text{for } i = 1, \ldots, m$$
$$N(0, j) = \text{“i”}, \quad \text{for } j = 1, \ldots, n$$

Scoring matrixes and trace back matrix recursion:

$X(i, j) = \text{Maximum}[M(i, j - 1) + o + e, X(i, j - 1) + e]$ (Maximum[begin gap in **a**, continue gap in **a**])
$Y(i, j) = \text{Maximum}[M(i - 1, j) + o + e, Y(i - 1, j) + e]$ (Maximum[begin gap in **b**, continue gap in **b**])
$M(i, j) = S(i , j) + \text{Maximum}[M(i - 1, j - 1), X(i - 1, j - 1), Y(i - 1, j - 1)]$ (Maximum[match or mismatch, end gaps in **a**, end gaps in **b**])

$$N(i, j) = \text{“d” if } M(i, j) = \text{Maximum}[M(i, j), X(i, j), Y(i, j)];$$
$$\text{“i” if } X(i, j) = \text{Maximum}[M(i, j), X(i, j), Y(i, j)];$$
$$\text{“j” if } Y(i, j) = \text{Maximum}[M(i, j), X(i, j), Y(i, j)]$$

for $i = 1, \ldots, m$ and for $j = 1, \ldots, n$

Toms algorithm

First improvement

As you can see on the Figure 4 the sells of optimal solution are located on diagonal of trace back matrix. For getting this solution we do not need to calculate whole matrix, it is enough to calculate the cells, located in tunnel **t** on **2** highlighted diagonals of Scoring Matrix shown on Figure 5. For our example two diagonals is minimum size of tunnel **t**, which can hold the cells for best alignment. We can easily see that if m = n the minimum tunnel width will be **1**. In case when **m != n** the minimum tunnel width will be **n – m + 1**. Here and further we expect that **n > m** (sequence **b** is longer than sequence **a**) otherwise we can exchange them.

In many cases (usually when sequences **a** and **b** have low similarity and the best alignment contains additional gaps) to hold the best alignment the tunnel should have additional quantity **tb** of diagonals (tunnel borders) located to the left and to the right of highlighted minimum tunnel diagonals. For our example (just to demonstrate the algorithm details) we will use tunnel width **tw = 4** - two highlighted minimum tunnel diagonals and two tunnel borders – one for each side of tunnel (**tb = 1**) .

Scoring Matrix

	-	A	C	T	G	A	T	T	C	A
-	0	-2	-4							
A	-2	2	0	-2						
C		0	4	2	0					
G			2	1	4	2				
C				-1	2	1	-1			
A					0	4	2	0		
T						2	6	4	2	
C							4	3	6	4
A								1	4	8

Trace back matrix

	-	A	C	T	G	A	T	T	C	A
-	done	i	i							
A	j	d	i	i						
C		j	d	i	i					
G			j	d	d	i				
C				d	j	d	d			
A					j	d	i	i		
T						j	d	d	i	
C							j	d	d	i
A								d	j	d

Figure 5. Diagonally located cells are sufficient for calculation of optimal solution

Now we can pay attention that:

- Optimal alignment solution for two FASTA files occupying diagonally located cells in tunnel between upper left corner and down right corner.
- From score matrix we can see that cells with maximum scores located in the tunnel or not so far from it.

Now for our calculation we can reduce the quantity of columns in score and trace back matrixes from **10** to **4**. Here **4** is tunnel width **tw**. New (tunnel) matrixes are shown on Figure 6.

Please pay attention that:

1. Northwest diagonals of Needleman-Wunsh matrixes become columns in Toms tunnel matrixes
2. Columns of Needleman-Wunsh matrixes become northeast diagonals in Toms tunnel matrixes
3. Column headers of Needleman-Wunsh matrixes (nucleotides of sequence **b**) become headers of northeast diagonals in Toms tunnel matrixes

Toms Tunnel Scoring Matrix

			-	A	C
-		0	-2	-4	T
A	-2	2	0	-2	G
C	0	4	2	0	A
G	2	1	4	2	T
C	-1	2	1	-1	T
A	0	4	2	0	C
T	2	6	4	2	A
C	4	3	6	4	
A	1	4	8		

Toms Tunnel Trace back matrix

			-	A	C
-		done	i	i	T
A	j	d	i	i	G
C	j	d	i	i	A
G	j	d	d	i	T
C	d	j	d	d	T
A	j	d	i	i	C
T	j	d	d	i	A
C	j	d	d	i	
A	d	j	d		

Figure 6. Toms score and trace back matrixes

Now direction for symbol **"i"** in tunnel trace back matrix is the same **"left"**, but **"d"** now means **"up"** and **"j"** means **"up and right"**. As we can see the process of calculation tunnel scoring matrix and tunnel trace back matrix is the same, difference only in location of cells relatively each other. Also trace back process should be started from position **k = tw - tb** in last row of tunnel trace back matrix. Here p is quantity of columns in tunnel trace back matrix (tunnel width **tw**) and **tb** os tunnel border width.

Second improvement

If we will look closer to the final step of algorithm we can see that trace back for the best alignment don't required the scoring matrix. This matrix we need only for calculation direction codes in the trace back matrix. This is why we don't need store in memory whole scoring matrix. We can store in memory only one row – current row. Cells of next row we can calculate from current row cells and place again to the current row.

Lets look to this process in details.

First of all we place to the current tunnel row **R** the row **#1** of scoring matrix:

#1	#2	#3	#4
	0	-2	-4

Please pay attention that this row **R** in column **#1** has no data because both score matrix and trace back matrix have no cell at this position in tunnel.
Score **R(k)** in position **k (k = 1, …, tw)** of current tunnel row **i** can be calculated using formula:

$$\mathbf{R(k) = Maximum[R(k) + S(i, j), M(k - 1) + g, M(k + 1) + g]}$$

Formula 2.

Here **j** is calculated using formula: **j = k + i – tb**, where **tb** is tunnel border size.
As we can see from Formula 2 when **k = 0** we do not include in Maximum[…] component **M(k – 1)+ g** and when **k = tw** we do not include in Maximum[…] component **M(k + 1) + g**, because score cells for these components are outside of tunnel.

Remark: in case with affine gap scores we have exactly the same algorithm for calculation three scoring matrixes M, X and Y. For every of these matrixex we can use only one row with length **tw** to store all data required for back trace matrix creation.

Third improvement

After implementing first two improvements we have time complexity proportional to **n*tw** for calculation of scores and filling the trace back matrix, where **n** is length of longest sequence and **tw** is the tunnel width. We also have time complexity proportional to **n** for deducing the alignment from the trace back matrix.
Space complexity is **m*2*tw** bytes to store trace back matrix, where **m** is length of shortest sequence and **tw** is tunnel width. (JavaScript, language that we used for implementing algorithms, store characters in 2 bytes as 16 bit binary numbers.)

We can reduce space complexity extra 8 times if we will present directional codes **“i”, “j”** and **“d”** in trace back matrix as two digit binary numbers, because as we said earlier, in JavaScript one character is represented by 2 bytes with 16 digit binary number. We can use following binary numbers for direction codes:

i	01
j	10
d	00

Figure 7. Binary numbers instead of direction code 16 bit characters

Using these binary codes we can store 8 direction binary codes (for example: **j, d, i, d, j, d, i, j**) in two byte character space as 16 bit binary number: **1000010010000110**.

In case of nucleotide alignment we also can reduce RAM memory usage 8 times by representing original sequences nucleotides instead of characters by two-digit binary number (Figure 8):

A	00
C	01
G	10
T	11

Figure 8. Binary numbers instead of nucleotide characters

In case of nucleotide alignment we can get additional reducing of RAM memory usage by using in alignmented sequences instead of characters three-digit binary numbers (Figure 9):

A	000
C	001
G	010
T	011
-	100

Figure 9. Binary numbers instead of nucleotide characters and gap signs

Forth improvement

Hirschberg’s ideas for linear space alignment can be applicable to our situation and reduce the algorithm space requirements to minimum.

In many cases when all results we need is only alignment score (sequence match quality) we don’t need to create the back track matrix and we can skip the trace back step. In these cases Toms algorithm has microscopic space requirements – **O(m)** and it can be effectively used in real time applications, embedding systems, etc.

Results

For testing we wrote JavaScript programs for both algorithms.
We used Mac Book Pro with 16 GB of RAM and 2.5 GHz processor.
For both algorithms and all tests we used **6** as a gap penalty **g**, **3** as a mismatch penalty, and **0** as the score for a match.
Browser for testing – Safari Version 10.0.1 (10602.2.14.0.7).

Test 1. Alignment of sequences with 40% - 50% similarity (the worst case scenario for Toms algorithm)

For comparison we found the same best alignment between two nucleotide sequences of equal lengths: 100, 200, … , 102,400 using both Needleman-Wunsch algorithm (**NW**) and Toms algorithm (**T**).

Results represented by Table 1.

Here is information about columns of Table 1:
Length contains the length of nucleotide sequences.
NW-time contains time of Needleman-Wunsch algorithm.

NW-ram contains RAM usage of Needleman-Wunsch algorithm.
T-time contains time of Toms algorithm.
T-ram contains RAM usage of Toms algorithm.
Tunnel holds the tunnel width of Toms algorithm, required to get the same alignment as Needleman-Wunsch algorithm.
NW-time/T-time shows haw many times Toms algorithm faster than Needleman-Wunsch algorithm.
NW-ram/T-ram shows haw many times Needleman-Wunsch algorithm consume RAM in compare with Toms Algorithm.

Length	NW-time	NW-ram	T-time	T-ram	Tunnel	NW-time/T-time	NW-ram/T-ram
100	0.009	3,528	0.001	1,228	16	9.00	2.87
200	0.016	12,028	0.002	2,428	16	8.00	4.95
400	0.055	44,028	0.003	4,828	16	18.33	9.12
800	0.187	168,028	0.006	9,628	16	31.17	17.45
1,600	0.718	656,028	0.013	22,428	24	55.23	29.25
3,200	2.791	2,592,028	0.032	51,228	32	87.22	50.60
6,400	11.190	10,304,028	0.151	179,228	80	74.11	57.49
12,800	49.634	41,088,028	0.528	512,028	128	94.00	80.25
25,600	195.584	164,096,028	3.011	2,867,228	416	64.96	57.23
51,200	756.726	655,872,028	10.006	9,420,828	704	75.63	69.62
102,400	3,004.203	2,623,488,112	29.442	27,721,056	1,032	102.04	94.64

Table 1. Toms vs Needleman-Wunsch time and space performance (40% - 50% similarity)

Maximum length of sequences we could align with Needleman-Wunsch algorithm was 102,400. Time was around 50 minutes.
As you can see Toms algorithm with the same quality align sequences with 102,400 nucleotide approximately 100 times faster and use 95 times less memory.

Test 2. Alignment of sequences with 92% similarity (8% mismatch) - best scenario

For comparison we found the same best alignment between two nucleotide sequences of equal length 100, 200, …, 102,400 using both Needleman-Wunsch algorithm (**NW**) and Toms Algorithm (**T**).

Results represented by Table 2.

Here is information about new columns of Table 2:
Mismatches specifies the quantity of randomly placed different nucleotide in sequences.

Length	Mismatches	NW-time	NW-RAM	T-time	T-ram	Tunnel	NW-time/T-time	NW-ram/T-ram
100	8	0.002	3,300	0.001	1,000	8	2.00	3.30
200	16	0.011	11,600	0.001	2,000	8	11.00	5.80
400	32	0.044	43,200	0.001	4,000	8	44.00	10.80
800	64	0.185	166,400	0.002	8,000	8	92.50	20.80
1,600	128	0.719	652,800	0.004	16,000	8	179.75	40.80
3,200	256	2.849	2,585,600	0.007	32,000	8	407.00	80.80
6,400	512	11.251	10,291,200	0.020	64,000	8	562.55	160.80
12,800	1,024	48.586	41,062,400	0.029	128,000	8	1,675.38	320.80
25,600	2,048	201.355	164,044,800	0.050	256,000	8	4,027.10	640.80
51,200	4,096	827.686	655,769,600	0.138	512,000	8	5,997.72	1,280.80
102,400	8,192	3,287.611	2,622,259,200	0.230	1,024,000	8	14,293.96	2,560.80

Table 2. Toms vs Needleman-Wunsch time and space performance (92% similarity)

As you can see Toms algorithm with the same quality align sequences with 102,400 nucleotides approximately 14,294 times faster and use 2,560 times less memory.

Test 3. Alignment of BIG sequences with 40% - 50% similarity

For comparison we found the approximate alignment between two nucleotide sequences of equal length 227,215,351 (Chromosome #1) using Toms Algorithm (**T**) with different tunnel size. This test gives us information how Toms Algorihm alignment quality depends from tunnel width.

Results represented by Table 2.

Here is information about new columns of Table 2:
Score specifies the alignment penalty score: smaller score – better alignment quality.
Matches specify the quantity of equal nucleotide in sequences.
Mismatches specify the quantity of different nucleotide in sequences.
Gaps specify the quantity of gaps added during alignment process.

Length	T-time	T-ram	Tunnel	Score	Matches	Mismatches	Gaps
227,215,351	997.887	2,726,584,240	16	437,852,397	95,977,786	126,333,157	9,808,821
227,215,351	1,630.276	3,181,014,942	24	434,939,886	97,728,677	124,322,248	10,328,857
227,215,351	2,166.913	3,635,445,644	32	433,325,934	98,685,443	123,225,888	10,608,045
227,215,351	2,680.877	4,089,876,346	40	432,331,602	99,284,883	122,537,116	10,786,709
227,215,351	3,183.363	4,544,307,048	48	431,618,751	99,708,500	122,051,499	10,910,709
227,215,351	3,710.798	4,998,737,750	56	431,096,403	100,020,157	121,693,995	11,002,403
227,215,351	4,222.587	5,453,168,452	64	430,686,033	100,262,946	121,415,873	11,073,069
227,215,351	4,752.078	5,907,599,154	72	430,338,033	100,464,188	121,186,217	11,129,897
227,215,351	5,230.103	6,362,029,856	80	430,044,945	100,630,764	120,996,681	11,175,817
227,215,351	5,695.261	6,816,460,558	88	429,805,587	100,766,551	120,842,227	11,213,151
227,215,351	6,118.027	7,270,891,260	96	429,596,463	100,884,094	120,708,739	11,245,041
227,215,351	6,765.925	7,725,321,962	104	429,405,135	100,989,465	120,589,503	11,272,771
227,215,351	7,363.305	8,179,752,664	112	429,235,173	101,089,457	120,475,065	11,301,663
227,215,351	7,992.569	8,634,183,366	120	429,037,440	101,185,506	120,368,970	11,321,755
227,215,351	8,651.942	9,088,614,068	128	428,901,582	101,263,576	120,279,972	11,343,611

Table 3. Toms algorithm quality (40% - 50% similarity)

Results of this test show us that:

1. Time growing in linear proportion from Tunnel width **tw**: **T-time = O(tw)**
2. Quality of alignment growing much slower than tunnel width **tw**. If we increase tunnel width from 16 to 128 (8 times or 700%) the alignment penalty score is reduced only from **437,852,397** to **428,901,582** (approximately 2%). This mean we can get much faster very good quality of alignment (even with 40% - 50% similarity) with relatively narrow tunnel.

Programing tip: Allocate and free Ram ASAP.

Summary

Because of huge performance (time and space) and outstanding quality of alignment Toms Algorithm can be used (with or without minor modifications) also in many applications outside of bioinformatics:

1. Computer stereo vision and stereo matching
2. Image rectification by camera resectioning or calibration
3. Removing lens unexpected distortions
4. Inexact text matching (e.g. spell checking; web page search)
5. Speech recognition
6. Real time applications
7. Embedding systems
8. Measuring navigation disorientation in web based systems
9. Natural language processing
10. Optimal matching in social science

References:

1. Needleman, Saul B. & Wunsch, Christian D. (1970). "A general method applicable to the search for similarities in the amino acid sequence of two proteins". Journal of Molecular Biology. 48 (3): 443–53. doi:10.1016/0022-2836(70)90057-4. PMID 5420325.
2. Hirschberg, D. S. (1975). "A linear space algorithm for computing maximal common subsequences". Communications of the ACM. 18 (6): 341–343. doi:10.1145/360825.360861. MR 0375829.
3. Gotoh, O. (1982) An improved algorithm for matching biological sequences. J. Mol. Biol. 162, 705-708.
4. Hirschberg, D. S. (1975) A linear space algorithm for computing maximal common subsequences. Comm. ACM, 18, 341-343.
5. Huang, X., R. Hardison and W. Miller (1990) A space-efficient algorithm for local similarities. CABIOS 6, 373-381.
6. Huang, X. and W. Miller (1991) A time-efficient, linear-space local similarity algorithm. Advances in Applied Mathematics 12, 337-357.
7. Myers, E. and W. Miller (1988) Optimal alignments in linear space. CABIOS 4, 11-17.
8. Myers, E. and W. Miller (1989) Approximate matching of regular expressions. Bull. Math. Biol. 51, 5-37.
9. Smith, T. F. and M. S. Waterman (1981) Identification of common molecular sequences. J. Mol. Biol. 197, 723-728.
10. Waterman, M. S., T. F. Smith and W. A. Beyer (1976) Some biological sequence metrics. Adv. Math. 20, 367-387.
11. Waterman, M. S. and M. Eggert (1987) A new algorithm for best subsequence alignments with application to tRNA-rRNA comparisons. J. Mol. Biol. 197, 723-728
12. Dieny R., Thevenon J., Martinez-del-Rincon J., Nebel J.-C. (2011) "Bioinformatics inspired algorithm for stereo correspondence". International Conference on Computer Vision Theory and Applications, March 5–7, Vilamoura - Algarve, Portugal.
13. Madeo S., Pelliccia R., Salvadori C., Martinez-del-Rincon J., Nebel J.-C. (2014) "An optimized stereo vision implementation for embedded systems: application to RGB and Infra-Red images". Journal of Real-Time Image Processing.
14. Martinez-del-Rincon J., Thevenon J., Dieny R., Nebel J.-C. (2012) "Dense Pixel Matching Between Unrectified and Distorted Images Using Dynamic Programming". International Conference on Computer Vision Theory and Applications, 24–26 February, Rome, Italy.
15. Tolga Güyer, Bilal Atasoy, and Sibel Somyürek Gazi University, Turkey. Measuring Disorientation Based on the NeedlemanWunsch Algorithm. International Review of Research in Open and Distributed Learning Volume 16, Number 2 http://files.eric.ed.gov/fulltext/EJ1061099.pdf
16. Abbott A.; Tsay A. (2000). "Sequence Analysis and Optimal Matching Methods in Sociology, Review and Prospect". Sociological Methods and Research. 29 (1): 3–33. doi:10.1177/0049124100029001001.