Sequence Local Alignment: The Smith-Waterman Algorithm

Oliver Bonham-Carter

December 17, 2013

1 Introduction

One of the fundamental tasks in bioinformatics research is to determine common regions of genetic code between samples of genetic sequence data. The discovery of code which is very similar suggests that both organisms may share a common ancestry. It may also suggest that the code is highly conserved may contain genes which code for proteins which both organisms require for survival. These common genes suggest that the proteins have similar functions (or else they might be quite different after all this evolutionary time) and can be used to determine the relatedness (taxonomy) of the organisms.

To determine similar regions between two strings or nucleotide or protein sequences, tools and algorithms from computer science are required due to the enormity of the task. To find which regions are common between sequences (known as a *local sequence alignment*), the SmithWaterman algorithm [1, 2] is often employed. Instead of comparing one entire sequence to another for global similarity, the algorithm works by comparing smaller segments of all possible lengths inside the sequences to all those of the other sequence. The algorithm optimizes the similarity measure of these smaller regions by the use of a matrix structure where scores are listed in each entry relating the base-base similarity in terms of the scores of previous bases similarities.

2 The Algorithm

We will follow the example given `http://en.wikipedia.org/wiki/Smith-Waterman_algorithm`. The algorithm creates a matrix H in which all the scoring work is performed. We first define the matrix cells;

$$H(i,0) = 0, \text{ for } 0 \leq i \leq m \text{ and } H(0,j) = 0, \text{ for } 0 \leq j \leq n.$$
$$\text{If } a_i = b_j \text{ then } w(a_i, b_j) = w(\text{match}) \text{ or if } a_i \neq b_j \text{ then } w(a_i, b_j) = w(\text{mismatch}).$$

$$\begin{cases} 0 & \\ H(i-1, j-1) + w(a_i, b_j) & \text{Match/Mismatch} \\ H(i-1, j) + w(a_i, -) & \text{Deletion} \\ H(i, j-1) + w(-, b_j) & \text{Insertion} \end{cases}$$

- $a, b =$ strings over alphabet Σ.
- $m = length(a)$
- $n = length(b)$
- $H(i, j)$ is the maximum Similarity Score between a suffix of $a[1...i]$ and a suffix of $b[1...j]$
- $w(c, d), c, d \in \Sigma \cup \{'-'\}$ is the gap-scoring scheme.

In this lab we will implement the Smith-Waterman local alignment algorithm in a spreadsheet where we can interact with the comparison of two sequences.

3 Example

We we will show the algorithm in action. We wish to compare two sequences, $S_1 = ACACACTA$ and $S_2 = AGCACACA$ with,

- the match score: $w(\text{match}) = +2$
- the mismatch score: $w(a, -) = w(-, b) = w(\text{mismatch}) = -1$

$$H = \left(\begin{array}{cccccccccc}
 & - & A & C & A & C & A & C & T & A \\
- & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
A & 0 & 2 & 1 & 2 & 1 & 2 & 1 & 0 & 2 \\
G & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 \\
C & 0 & 0 & 3 & 2 & 3 & 2 & 3 & 2 & 1 \\
A & 0 & 2 & 2 & 5 & 4 & 5 & 4 & 3 & 4 \\
C & 0 & 1 & 4 & 4 & 7 & 6 & 7 & 6 & 5 \\
A & 0 & 2 & 3 & 6 & 6 & 9 & 8 & 7 & 8 \\
C & 0 & 1 & 4 & 5 & 8 & 8 & 11 & 10 & 9 \\
A & 0 & 2 & 3 & 6 & 7 & 10 & 10 & 10 & 12
\end{array}\right)$$

Figure 1: Here we see the scores. We follow the highest score in the bottom right to the upper left. At each point, we note the bases in the sequences to give us the most optimal alignment having the highest score.

$$T = \left(\begin{array}{cccccccccc}
 & - & A & C & A & C & A & C & T & A \\
- & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
A & 0 & \nwarrow & \leftarrow & \nwarrow & \leftarrow & \nwarrow & \leftarrow & \leftarrow & \nwarrow \\
G & 0 & \uparrow & \nwarrow & \uparrow & \nwarrow & \uparrow & \nwarrow & \nwarrow & \uparrow \\
C & 0 & \uparrow & \nwarrow & \nwarrow & \nwarrow & \leftarrow & \nwarrow & \leftarrow & \leftarrow \\
A & 0 & \nwarrow & \uparrow & \nwarrow & \leftarrow & \nwarrow & \leftarrow & \leftarrow & \nwarrow \\
C & 0 & \uparrow & \nwarrow & \uparrow & \nwarrow & \leftarrow & \nwarrow & \leftarrow & \leftarrow \\
A & 0 & \nwarrow & \uparrow & \nwarrow & \uparrow & \nwarrow & \leftarrow & \leftarrow & \nwarrow \\
C & 0 & \uparrow & \nwarrow & \uparrow & \nwarrow & \uparrow & \nwarrow & \leftarrow & \leftarrow \\
A & 0 & \nwarrow & \uparrow & \nwarrow & \uparrow & \nwarrow & \uparrow & \nwarrow & \nwarrow
\end{array}\right)$$

Figure 2: Here we note the arrows have replaced the scores. The arrows help to show the direction of the most optimal alignment. Each arrow indicates the best paring of two bases in the final alignment.

To obtain the optimum local alignment, we start with the highest value in the matrix (i, j). Then, we go backwards to one of positions $(i1, j)$, $(i, j1)$, and $(i1, j1)$ depending on the direction of movement used to construct the matrix. We keep the process until we reach a matrix cell with zero value as in Figure 1.

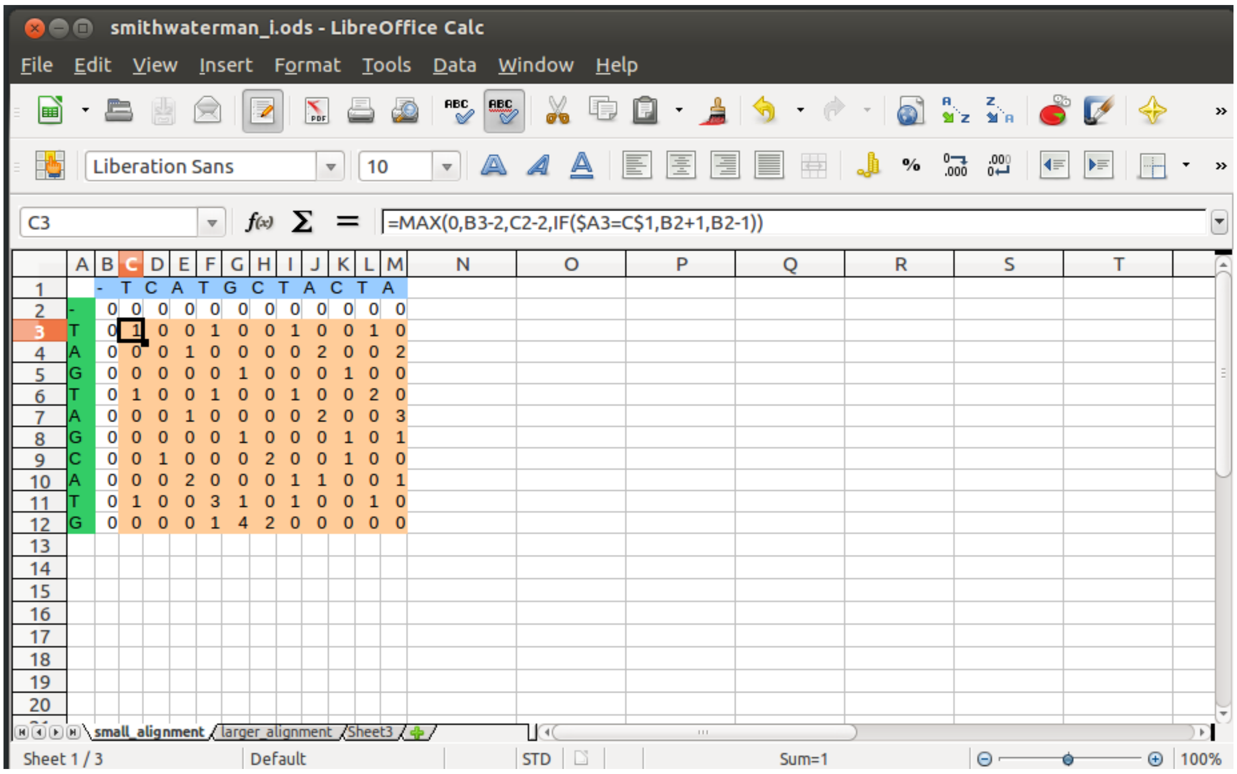
In the example, the highest value corresponds to the cell in position $(8, 8)$. The walk back corresponds to $(8, 8)$, $(7, 7)$, $(7, 6)$, $(6, 5)$, $(5, 4)$, $(4, 3)$, $(3, 2)$, $(2, 1)$, $(1, 1)$, and $(0, 0)$. We use arrows to describe this path in the matrix shown in Figure 2. Once we've finished, we reconstruct the alignment as follows: Starting with the last value, we reach (i, j) using the previously calculated path. A diagonal jump implies there is an alignment (either a match or a mismatch). A top-down jump implies there is a deletion. A left-right jump implies there is an insertion. Our final output from the Smith-Waterman algorithm is the following local alignment:

$$S_1 = \texttt{A-CACACTA}$$
$$S_2 = \texttt{AGCACAC-A}$$

4 Lab Work

For this experiment, we will use OpenOffice.org's Spreadsheet, although Excel should also work. Add the equation below to the cell `C3` and then drag and drop the equation to all the cells as in Figures 3 and 4. Do not program the equations to any other cells.

$$= MAX(0, B3 - 2, C2 - 2, IF(\$A3 = C\$1, B2 + 1, B2 - 1))$$


smithwaterman_i.ods - LibreOffice Calc
File Edit View Insert Format Tools Data Window Help
Liberation Sans
10
C3
small_alignment
larger_alignment
Sheet3
Sheet 1 / 3
Default
STD
Sum=1
100%

```
=MAX(0,B3-2,C2-2,IF($A3=C$1,B2+1,B2-1))
```

	A	B	C	D	E	F	G	H	I	J	K	L	M
1		-	T	C	A	T	G	C	T	A	C	T	A
2	-	0	0	0	0	0	0	0	0	0	0	0	0
3	T	0	1	0	0	1	0	0	1	0	0	1	0
4	A	0	0	0	1	0	0	0	0	2	0	0	2
5	G	0	0	0	0	0	1	0	0	0	1	0	0
6	T	0	1	0	0	1	0	0	1	0	0	2	0
7	A	0	0	0	1	0	0	0	0	2	0	0	3
8	G	0	0	0	0	0	1	0	0	0	1	0	1
9	C	0	0	1	0	0	0	2	0	0	1	0	0
10	A	0	0	0	2	0	0	0	1	1	0	0	1
11	T	0	1	0	0	3	1	0	1	0	0	1	0
12	G	0	0	0	0	1	4	2	0	0	0	0	0

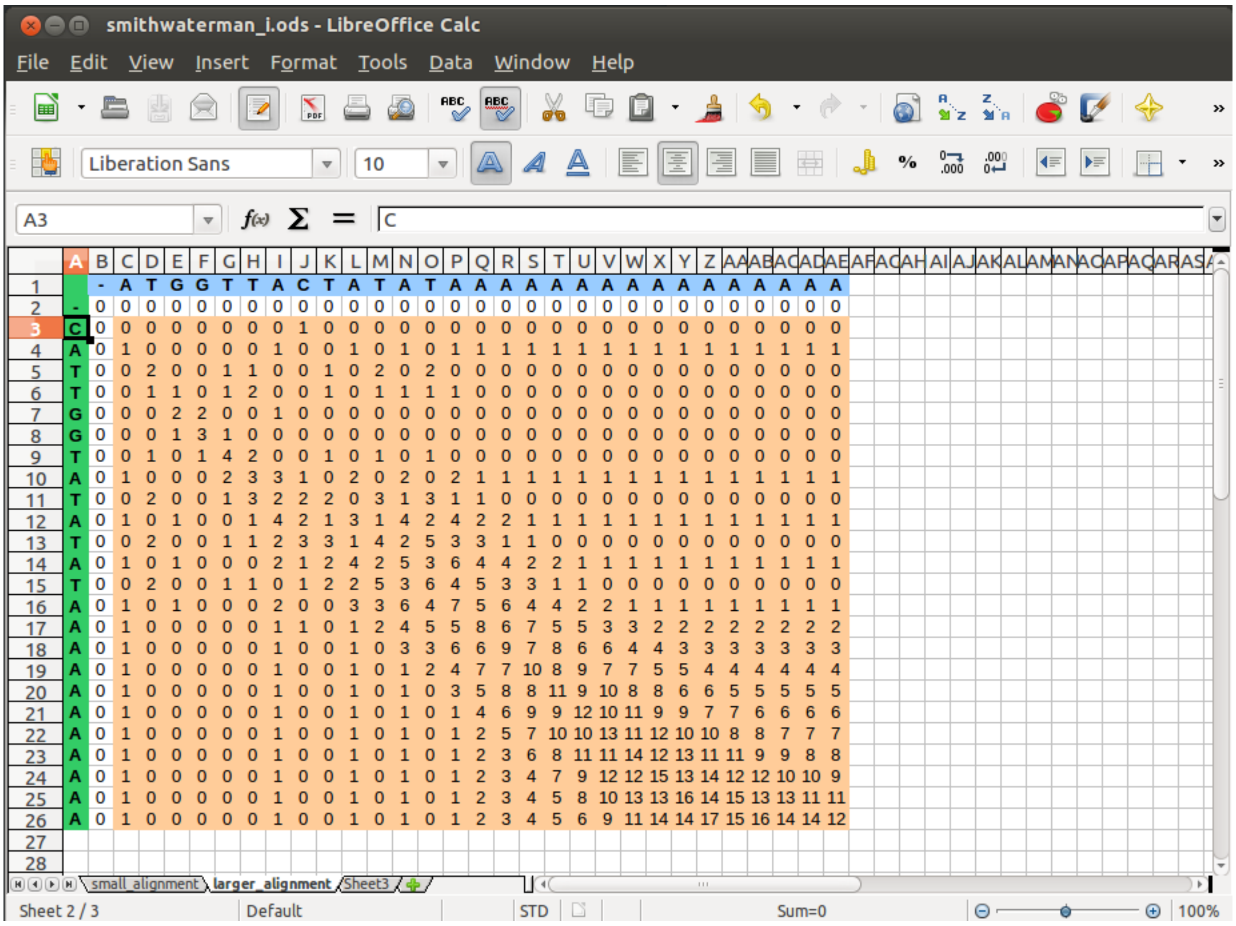
Figure 3: The OpenOffice spreadsheet. Excel ought to work as well. The equation at cell C3 is: $= MAX(0, B3 - 2, C2 - 2, IF(\$A3 = C\$1, B2 + 1, B2 - 1))$.

The actual alignment is the following. Can you see why this is the case?

```
Seq3 ATGGTTACTATATAAAAAAAAAAA
     ||.|.|| ||||||||||||||||
Seq4 ATTGGTA-TATATAAAAAAAAAAAA
```

4.1 Experiment!

Try adding your own sequences to perform an alignment using the Smith-Waterman local alignment algorithm. Can you determine which is the best alignment for your sequences?


smithwaterman_i.ods - LibreOffice Calc
File Edit View Insert Format Tools Data Window Help
Liberation Sans
10
A3
C
small_alignment
larger_alignment
Sheet3
Sheet 2 / 3
Default
STD
Sum=0
100%

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	AA	AB	AC	AD	AE
1		-	A	T	G	G	T	T	A	C	T	A	T	A	T	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A
2	-	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	C	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	A	0	1	0	0	0	0	0	1	0	0	1	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
5	T	0	0	2	0	0	1	1	0	0	1	0	2	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	T	0	0	1	1	0	1	2	0	0	1	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	G	0	0	0	2	2	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	G	0	0	0	1	3	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	T	0	0	1	0	1	4	2	0	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	A	0	1	0	0	0	2	3	3	1	0	2	0	2	0	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
11	T	0	0	2	0	0	1	3	2	2	2	0	3	1	3	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	A	0	1	0	1	0	0	1	4	2	1	3	1	4	2	4	2	2	1	1	1	1	1	1	1	1	1	1	1	1	1
13	T	0	0	2	0	0	1	1	2	3	3	1	4	2	5	3	3	1	1	0	0	0	0	0	0	0	0	0	0	0	0
14	A	0	1	0	1	0	0	0	2	1	2	4	2	5	3	6	4	4	2	2	1	1	1	1	1	1	1	1	1	1	1
15	T	0	0	2	0	0	1	1	0	1	2	2	5	3	6	4	5	3	3	1	1	0	0	0	0	0	0	0	0	0	0
16	A	0	1	0	1	0	0	0	2	0	0	3	3	6	4	7	5	6	4	4	2	2	1	1	1	1	1	1	1	1	1
17	A	0	1	0	0	0	0	0	1	1	0	1	2	4	5	5	8	6	7	5	5	3	3	2	2	2	2	2	2	2	2
18	A	0	1	0	0	0	0	0	1	0	0	1	0	3	3	6	6	9	7	8	6	6	4	4	3	3	3	3	3	3	3
19	A	0	1	0	0	0	0	0	1	0	0	1	0	1	2	4	7	7	10	8	9	7	7	5	5	4	4	4	4	4	4
20	A	0	1	0	0	0	0	0	1	0	0	1	0	1	0	3	5	8	8	11	9	10	8	8	6	6	5	5	5	5	5
21	A	0	1	0	0	0	0	0	1	0	0	1	0	1	0	1	4	6	9	9	12	10	11	9	9	7	7	6	6	6	6
22	A	0	1	0	0	0	0	0	1	0	0	1	0	1	0	1	2	5	7	10	10	13	11	12	10	10	8	8	7	7	7
23	A	0	1	0	0	0	0	0	1	0	0	1	0	1	0	1	2	3	6	8	11	11	14	12	13	11	11	9	9	8	8
24	A	0	1	0	0	0	0	0	1	0	0	1	0	1	0	1	2	3	4	7	9	12	12	15	13	14	12	12	10	10	9
25	A	0	1	0	0	0	0	0	1	0	0	1	0	1	0	1	2	3	4	5	8	10	13	13	16	14	15	13	13	11	11
26	A	0	1	0	0	0	0	0	1	0	0	1	0	1	0	1	2	3	4	5	6	9	11	14	14	17	15	16	14	14	12

Figure 4: The OpenOffice spreadsheet. Here we have a larger comparison task to perform. More calculation is now necessary for these longer sequences. $S_3 = ATGGTTACTATATAAAAAAAAAAAAAAAA$ and $S_4 = CATTGGTATATATAAAAAAAAAAA$.

References

[1] T. F. Smith and M. S. Waterman, “Identification of common molecular subsequences,” *Journal of molecular biology*, vol. 147, no. 1, pp. 195–197, 1981.

[2] S. B. Needleman and C. D. Wunsch, “A general method applicable to the search for similarities in the amino acid sequence of two proteins,” *Journal of molecular biology*, vol. 48, no. 3, pp. 443–453, 1970.