# Global Alignment

## Algorithms in BioInformatics
## Mandatory Project 1

**Magnus Erik Hvass Pedersen (971055)**
**Daimi, University of Aarhus**
**September 2004**

## 1 Introduction

The purpose of this report is to verify attendance of the author to the *Algorithms in BioInformatics* course at the department of computer science, University of Aarhus.

The reader is assumed to be familiar with the problem description as well as the course litterature, and [1] in particular.

## 2 Affine Gap Weights

Calculating the affine gap weights can be done using the recurrences in [1, p. 244]. First we have:

$$V(0,0) = E(0,0) = F(0,0) = 0$$

And when the end gaps are not free, the general base case for $i, j > 0$, is as follows:

$$\begin{aligned} V(i,0) = E(i,0) &= -W_g - iW_s \\ V(0,j) = E(0,j) &= -W_g - jW_s \end{aligned}$$

where $W_g > 0$ is the gap-weight and $W_s \geq 0$ is the space-weight. The overall recurrence is defined as:

$$V(i,j) = \max\left[E(i,j),\, F(i,j),\, G(i,j)\right]$$

with the sub-recurrences being:

$$\begin{aligned} E(i,j) &= \max\left[E(i,j-1),\, V(i,j-1) - W_g\right] - W_s \\ F(i,j) &= \max\left[F(i-1,j),\, V(i-1,j) - W_g\right] - W_s \\ G(i,j) &= V(i-1,j-1) + s(S_1(i), S_2(j)) \end{aligned}$$

Note that $G(i,j)$ is not a recurrence, but the other two, $E$ and $F$, are both recurrences that must have their own tables in the dynamic programming method.

The values or scores of comparing different characters are defined in a score matrix, so that $s(S_1(i), S_2(j))$ is this number for comparing characters $S_1(i)$ and $S_2(j)$. We could similarly have that the weight for each space $W_s$, was a comparison such as $s(S_1(i), \_)$, but this is not suggested by the score matrix given in the project formulation.

# 3 Constant Gap Weights

If we remove the influence of the space-weight by setting $W_s = 0$ in the above, so that gaps are weighted only with the constant $W_g$, we then have the base case:

$$V(i,0) = E(i,0) = -W_g$$
$$V(0,j) = E(0,j) = -W_g$$

And now, we only have one recurrence:

$$V(i,j) = \max\left[E(i,j),\, F(i,j),\, G(i,j)\right]$$

Where $E$ and $F$ are no longer recurrences themselves:

$$\begin{aligned} E(i,j) &= V(i,j-1) - W_g \\ F(i,j) &= V(i-1,j) - W_g \\ G(i,j) &= V(i-1,j-1) + s(S_1(i), S_2(j)) \end{aligned}$$

Which for $E$'s case, can be seen from the following:

$$\begin{aligned} E(i,j) &= \max\left[E(i,j-1),\, V(i,j-1) - W_g\right] \\ &= \max\left[\max\left[E(i,j-2),\, V(i,j-2) - W_g\right],\, V(i,j-1) - W_g\right] \\ &= \max\left[E(i,j-2),\, V(i,j-2) - W_g,\, V(i,j-1) - W_g\right] \\ &= \cdots \\ &= \max\left[E(i,0),\, V(i,0) - W_g,\, V(i,1) - W_g,\, \cdots,\, V(i,j-1) - W_g\right] \\ &= \max\left[V(i,0),\, V(i,0) - W_g,\, V(i,1) - W_g,\, \cdots,\, V(i,j-1) - W_g\right] \\ &= \max\left[V(i,0),\, V(i,1) - W_g,\, \cdots,\, V(i,j-1) - W_g\right] \\ &= \max\left[V(i,0),\, V(i,1) - W_g,\, \cdots,\, V(i,j-1) - W_g\right] \end{aligned}$$

And similarly for $F$, so when we are ultimately only interested in $V(i,j)$, and $V(n,m)$ in particular, we find that $E(i,j-1)$ is included in $V(i,j-1)$ when $W_s = 0$. Similarly we have that $F(i-1,j)$ is included in $V(i-1,j)$. It will therefore suffice with only one recurrence table for the dynamic programming method, namely that of $V(i,j)$.

# 4 Trace-Back

Computing the trace-back pointers is done as usual, in that a corresponding pointer is added for each of $E$, $F$, and $G$ that equals $V$ for that cell. If this is the case for $E$, then a left-pointer is added, for $F$ an up-pointer is added, and for $G$ a diagonal pointer is added.

After the entire table has been calculated, the pointers are then traversed in a backwards manner, outputting the character from $S_2$ and a space instead of the character from $S_1$ if a cell points left, and if pointing up, a space is printed

instead of the character from $S_2$, and the character from $S_1$ is then printed. Finally, if the pointer is diagonal, then both characters are printed, regardless of whether they match or not.

# 5 Implementation

Implemented in *Microsoft Visual C++ .NET*, the base-class for an alignment algorithm is `LAlign`. The sub-classes for quadratic- and linear-space algorithms are then `LAlignQuadraticSpace` and `LAlignLinearSpace`, which implement the algorithm for computing string similarity with a space-weight of 1, similar to [1, p. 227].

The linear-space algorithm does not work correctly, and is therefore not extended to the cases of constant and affine gap-weights. The quadratic-space counterparts are implemented however, and can be found in `LAlignQSConstantGap` which is further extended in `LAlignQSAffineGap`.

The main-program that communicates with the user and iterates the input sequences, writes the output and so forth, is found in the file `bioinf_align.cpp` and the project does indeed compile to a *Microsoft Windows* executable named `bioinf_align.exe`.

## 5.1 Virtual Functions

The most interesting usage of virtual functions for specializing the behaviour of sub-classes, is found in the `LAlignQuadraticSpace`-class with the functions for calculating the up-, left-, and diagonal scores of a cell $V(i, j)$, and for initializing the first row and column of the dynamic programming table. These functions are then specialized in the `LAlignQSConstantGap` and `LAlignQSAffineGap` classes.

## 5.2 Trace-Back Table

The cells in the trace-back table are `char`-typed, as we only need 3 bits to store whether the pointer exists at all, is pointing up, left, or diagonally. The proper bits are then added with the binary-or operand, and retrieved with binary-and.

When traversing the track-back table in reverse order, starting at cell $(n, m)$, the different kinds of pointers are prioritized so that diagonal pointers are favoured over up-pointers, which again precede the left-pointer – again, this is necessary as there may be several optimal alignment paths in the table.

## 5.3 File Formats

The class for the score-matrix is `LScoreMatrix`, and is read from a file in socalled *Phylip* format, from which the alphabet is also implicitly defined. If unknown characters are encountered from the input sequences, they are replaced with the first character defined in the score-matrix. Internally to the program, all characters are in lower-case, so an alphabet can not be case-sensitive.

The user can choose between file-formats for the input- and output-sequence files, as the strings that the user enters are merely passed to the SEQIO functions, which are capable of handling GenBank, FASTA, etc.

Output files measuring time usage, scores, gap-lengths, and number of gaps, are created in a format compatible with *gnuplot*'s command *plot*, which is used for creating the figures below.

# 6 Testing

There is no explicit error- or exception-handling, but the `assert()`-function is used in the debug-build of the program, so that erroneous conditions are caught during development.

During development, small hand-written sequences were tested with the intention of uncovering implementation errors. For the quadratic-space algorithms, no further errors have been encountered.

For some unknown reason, running the release-build in a command-shell, prints the wrong scores, whereas running the same build from the MS Visual Studio environment, prints the correct scores.

# 7 Experimental Results

Using the given score-matrix (`score01.txt`) and sequence-files (`human.seq` and `mouse.seq`, both in GenBank format), with gap-weight $W_g = 10$ and space-weight $W_s = 2$, and not accepting sequence-pairs where $n \cdot m > 2.5e7$, yields 80 processed sequence pairs out of 117 in total.

The algorithm with affine gap-weight scores a total of 1294508, with a total of 20850 gaps, totalling a gap-length of 92032. This makes for an average of 260.6 gaps per sequence with an average gap-length of 4.414. The time-usage is illustrated in figure 1, and underlines the analysis in [1], that the algorithm is indeed $O(n \cdot m)$. The somewhat *jagged* tendency of the graph, is probably due to disk-cache usage, imposed by the large additional memory requirements for the $E$ and $F$ tables.

The algorithm with constant gap-weight scores a total of 1251561, with a total of 12986 gaps and a total gap-length of 87118. The average is therefore 162.3 gaps per sequence, and 6.71 in average gap-length. So having $W_s = 0$ and $W_g = 10$ means that there are substantially fewer, but somewhat longer gaps, than with the affine algorithm having a space penalty of $W_s = 2$. Note however, that the overall similarity score is higher when using affine gap-weights, even though there are more gaps.

The time-usage for the constant gap-weight algorithm, is shown in figure 2, and again supports the $O(nm)$ analysis.

The memory usage has not been explicitly tested, but it appears obvious from the source-code, that the allocated arrays are quadratic in their space-usage. (Naturally intuition on these matters may fail.)

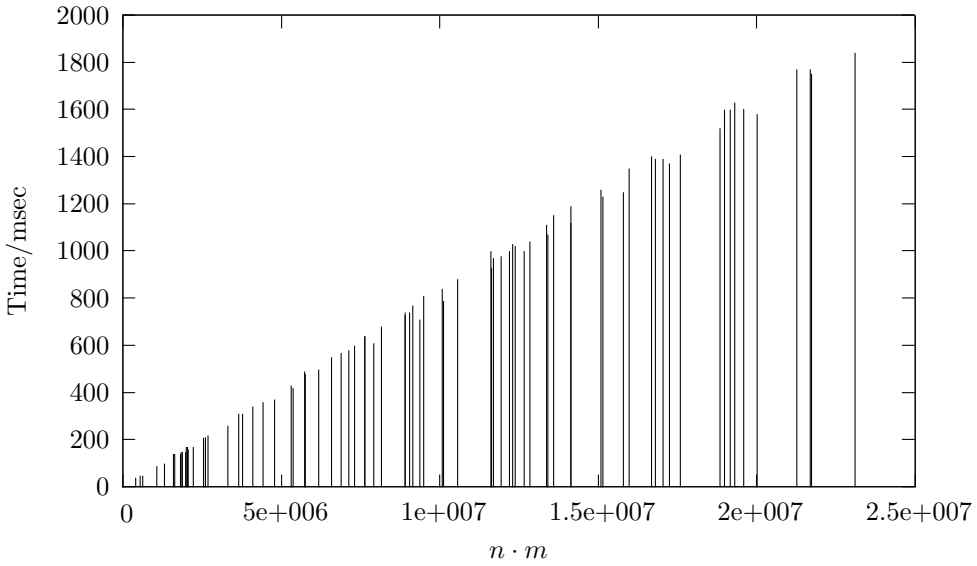


Figure 1: Affine gap-weight, relationship between length of input sequences ($n \cdot m$) and time usage, documenting the $O(nm)$ time-complexity.

The similarity score with affine and convex gap-weights are shown in figures 3 and 4, respectively.

Watching the output of the program, a substantial amount of time is spent on memory allocation and de-allocation. A solution would be to only allocate new aligning-objects when the lengths of the strings have grown beyond what is allocated, instead of allocating (and then deleting) a new object for each sequence-pair. The `LAlign`-object should then keep track of its allocated size as well as the size required by the current strings. See appendix A for an example of two aligned sequences.

# References

[1] Dan Gusfield: Algorithms on Strings, Trees, and Sequences, Cambridge University Press, 1999, ISBN 0-521-58519-8

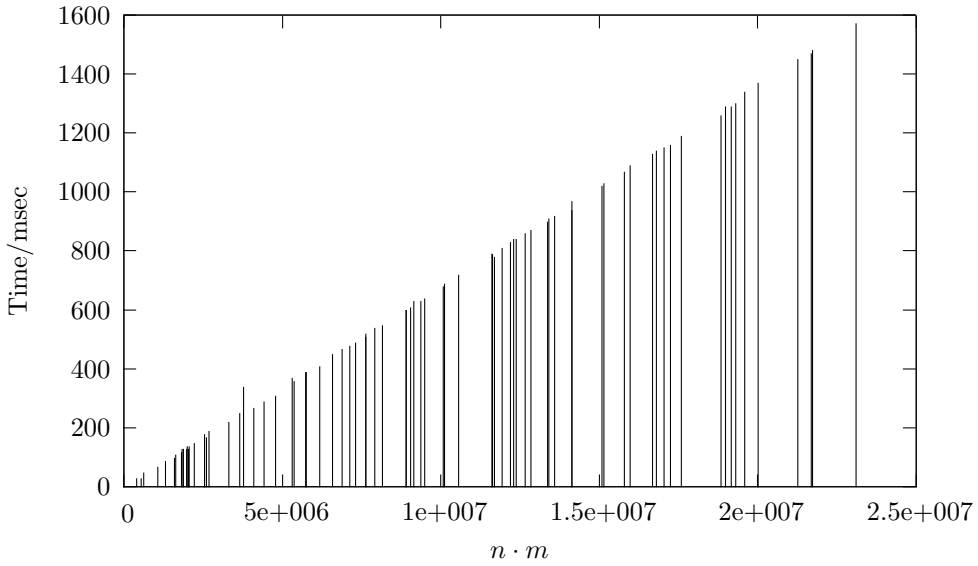


Figure 2: Constant gap-weight, relationship between length of input sequences ($n \cdot m$) and time usage, documenting the $O(nm)$ time-complexity.

# A Sequence Alignment

The *HUMOTNPI* sequence from `human.seq`, and *MUSOXYNEUI* from `mouse.seq`, aligned with affine gap-weight ($W_g = 10$ and $W_s = 2$).

```
------------------------------------------------------------
ggatccagcacctcttctggtcgccaaggaaacctgcgtgcacaaatgtacacacacaaa

------------------------------------------------------------
attaaaattgaaatgcaaaaactgttcccaaaatgtactgctaccatcatgcgggggact

------------------------------------------------------------
tgccccacccaacgtcgctcacacactaggcaagtactctgctactggggctgcatgtaa

------------------------------------------------------------
caccttcccatgcagacctatgcagacctgcagcccaaacctgaaatgtacccagagcct

------------------------------------------------------------
gcccaacctgattgcaccaaaatgggcgaaccatcatatgtggcccacctgagaagggta

------------------------------------------------------------
tgaccttggcacaaatggccttgcctgtagcctgaggccacctgttggccacactccagc
```

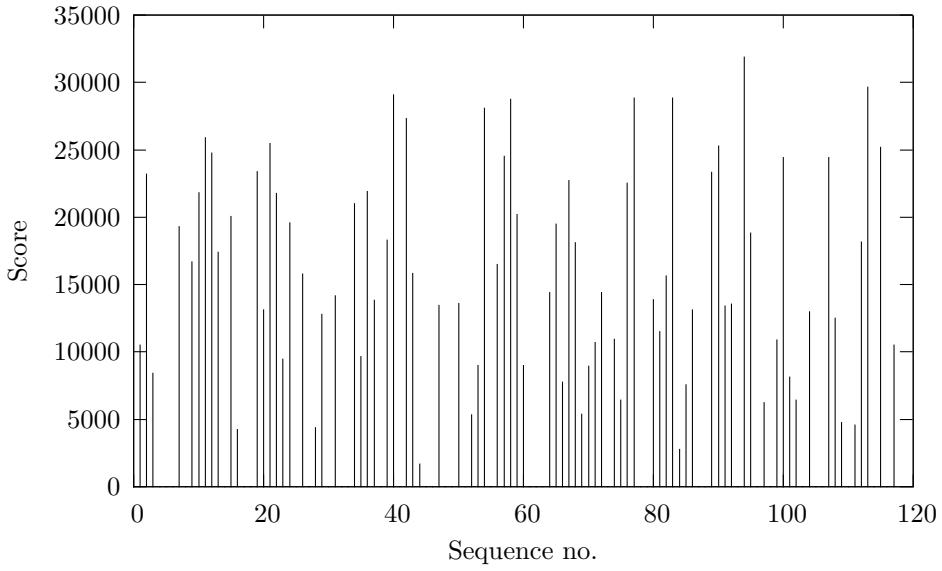


Figure 3: Affine gap-weight, similarity score for each sequence-pair.

```
-----------------------------gga---t------c-----------------
agtctgatggcccactgtcctctcaaacaggagtctaggcacctagtgtggtagtggata

-ctg----c--c-----agag-c-----c---t------cc------tc----cca----
actaaactcagcatttgggagacagaagcagatggagcgctgtgagttcaaggccagcct

--cct--------gg----ag----ggg---t---c-cc------a-------------g
ggtctacacagcaggttctaatacagagttttaaatactagtgtaattttccttttgctg

c-g----tc---c-----------ac-----c--ttc-c-ctgcccca------g---c-
taatttttctttctttttatttaatttggtctgttaactctgttttggtttgatctct

-cc----ccctcc--t-----c------gaggtactggga-g-g--c------t------
actgagactttgttttacgggctagttagaagagctgaggtgcattcagagattgaacaa

g--g--------------ata-aagtct----tcggc-t--gggcca-cac-ccca----
gacgccatctttttccccatctaagtttcaggtccatgtaagggctccctcactcactgg

ccccaaatt--c---tccc-----------tgtcc--------ca---cc---cta----
ccctatcctgccttattctgagatattggatatctgtgaaaaacagctcctggctagggc

gtg-------------ccca--------ggccaccc-cggcct-gctcccttccgcaagg
gcacctccaacccctcccaagtctctctagcctcttgtagcctaggccacccccttccagg
```

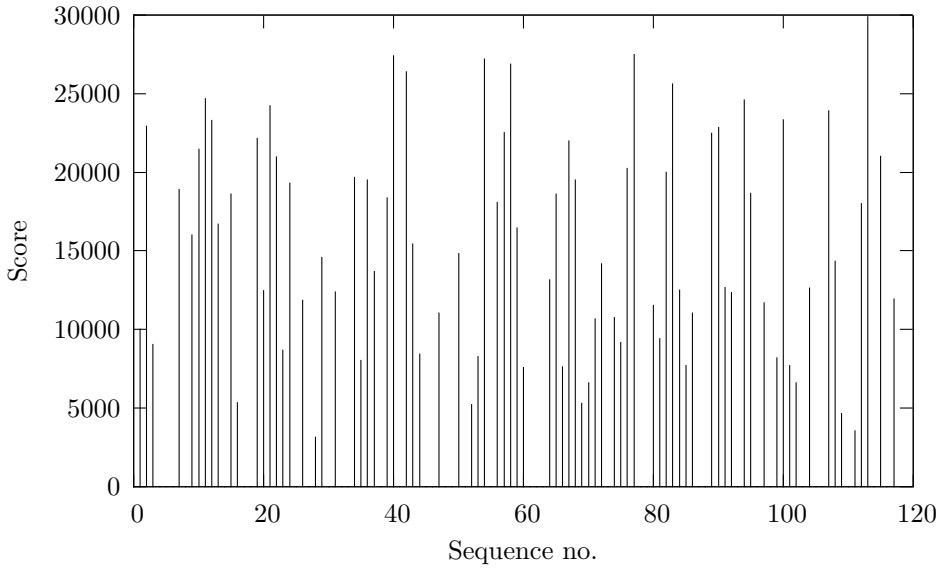


Figure 4: Constant gap-weight, similarity score for each sequence-pair.

```
cacctcaccttctgtg-cccagaccattagccaacgcggtgaccttgaccccggcccagg
ctgcttctcttttgagttccaggtcattagcagagacgatgaccttgaccctagcccaga

ccctgctaatgaagaggaaagc-cc--gta-cgcact-cggcctgacccacggcgaccct
ccctgcaaatgaagggcctgcctctaaacagcgtggaacaatttcaccca-agagacctt

ctgtgaccaatcatactaccaacctc-ttaaacagagctccaccgacgcaatgcccaggc
ctgtgaccagtcatgctgtcaccctctttagacagtgctccaccatggcagtg-ccagac

ataaaaa----gg-ccaggcc-gagagaccgccaccagtcacggaccctggacccagcgc
ataaaaaggtcggtctgggccggagaaaccatcacctacagcggatctcaga-ct-gagc

acccgcaccatggccggccccagcctcgcttgctgtctgctcggcctcctggcgctgacc
accatcgccatggcctgccccagtctcgcttgctgcctgcttggcttactggctctgacc

tccgcctgctacatccagaactgccccctgggaggcaagagggccgcgccggacctcgac
tcggcctgctacatccagaactgccccctgggcggcaagagggctgtgctggacctggat

gtgcgcaaggtgagtc-cccagccctggtcccgcggcgctccggggagggagggacccgc
atgcgcaaggttagtctccccgaccctgtccc-tt-cccttcccgttctg-gcgatgcta

agccacaggggcgcgccccgctccggcctcgcctgagaactccaggagctgagcggattt
```

```
aggacca-gagaa-g---ctctcccacct-aca-gagagcattc----c--cgcaca-ct

tgacgccccgcccttgaccgcggtcgaggcccccacggcgccccagcgtctcagccccgc
tgccagccctaccaaggcctcgcgtg-ggaacccagggctttgggaagtgtta-----g-

tgtccccgcccgaactccgaaccccggaccccagcatccttgcccggcgcaccccggccg
-gctccctcttg-a---cg--ccgtgaagg-taacgacaatg-ccggagcacccactgcc

gcctcgcagggtcctccgagcgagtccccagcgccgccccgcgtcccgctcaccccgccc
-cctcgctctg-ccacagtccggattcggattg-tgcacggcg--cc-cac-ccgcatcc

gtcccccgagtgcctcccctgcggccccgggggcaaaggccgctgcttcgggcccaatat
ttccccacagtgtctcccctgcggcccgggcggcaaaggacgctgcttcggaccaagcat

ctgctgcgcggaagagctgggctgcttcgtgggcaccgccgaagcgctgcgctgccagga
ctgctgcgcggacgagctgggctgcttcgtgggcaccgccgaggcgctgcgctgccagga

ggagaactacctgccgtcgccctgccagtccggccagaaggcgtgcgggagcgggggccg
ggagaactacctgccttcgccctgccagtctggccagaagccctgcgggagcggaggccg

ct--g-cgccttgggcctctgctgcagcccgggtgagcggggcaaggcgc-tccgg-g-g
ctgcgccgccacaggcatctgctgcagcccgggtgagcaggaggggggcccagcaggtgac

ccagggggaggcgggcgggggtgcggccgggattcccctgactccacctcttcctccaga
ccggcaaggagccgtcgggtttgcagctcaga--acactgac-ccatttc-tcttgcaga

cggctgccacgccgaccctgcctgcgacgcggaagccaccttctcccagcgctgaaactt
tggctgccgcacagaccccgcctgcgaccctgagtctgccttctcggagcgctg-agccc

gatggctccgaacaccctcgaagcgcgccactcgcttccc----ccatagccacccc---
actttctgggaataccttagcgcgcttccttcgttccccatggccactgccagaaaaaa

----------ag-aaa-tggtgaaaataaaataaag---ca--ggtttttctcctctacct
aaaaaaaaaagaaaagaaaagaaaagaaaagaaaaataaagtagatttcctcttcaaact

tgactcgtgtctaagtgccagaaatggga---cgg---gg-a--gggggcattgtgggac
tgactggtgtctaattgtcggaaacgggagggaggaaaggcaccgggaacgccgtgctct

t-ggaa--gatc
tggcatcttgta
```