Building a sequence aligner capable of affine gap penalties.

Isak Johansson-Åkhe, Linköping University

October 2018

Introduction

Protein-protein sequence alignment is the basis and first step of many bioinformatical methods. In 1970, the first draft of the Needleman-Wunsch algorithm for global sequence alignment was first introduced [1]. This was a simple algorithm, with no gap penalties (only rewards or penalties for matches or mismatches), and cubic complexity. Through the years, many great additions have come to this first basis for sequence alignment, including gap penalties, *affine gap penalties*, a simplification to quadratic complexity, and the option for finding the best local alignments with the Smith-Waterman algorithm [2].

In this short project, an aligner capable of finding both global and local alignments while using the affine gap penalty is implemented and briefly analyzed.

Theory

Presented below are the respective algorithms for global and local alignment using affine gap penalties.

Needleman-Wunsch with Affine Gap Penalty

Input:

Sequences A and B to be aligned.

Variables:

$i \in \{1, 2, ..., n\}$
$j \in \{1, 2, ..., m\}$
$M(i, j)$
$G_x(i, j)$
$G_y(i, j)$
where n and m are the lengths of sequences A and B, respectively.

Initialization:

$M(0,0) = 0$
$G_x(i,0) = h + g \times i$
$G_y(0,j) = h + g \times j$
all other cells: $-\infty$

where h is the penalty for opening a new gap and g is the penalty for extending a gap.

Calculations:

For all i and j, do:

$$M(i,j) = \max \begin{cases} M(i-1,j-1) + s(x_i, y_j), \\ Gx(i-1,j-1) + s(x_i, y_j), \\ G_y(i-1,j-1) + s(x_i, y_j) \end{cases} \tag{1}$$

$$G_x(i,j) = \max \begin{cases} M(i-1,j) + h + g, \\ G_x(i-1,j) + g \end{cases} \tag{2}$$

$$G_y(i,j) = \max \begin{cases} M(i,j-1) + h + g, \\ G_y(i,j-1) + g \end{cases} \tag{3}$$

where s is the scoring matrix used, x_i is the amino acid at position i in sequence A, and y_j is the amino acid at position j in sequence B.

Traceback

Start at $\max(M(n,m), G_x(n,m), G_y n, m)$ and follow pointers back to $M(0,0)$, $G_x(0,0)$, or $G_y 0, 0$, introducing gaps in A and B when moving in G_x or G_y respectively, and matches when moving in M.

Smith-Waterman with Affine Gap Penalty

Input:

Sequences A and B to be aligned.

Variables:

$i \in \{1, 2, ..., n\}$
$j \in \{1, 2, ..., m\}$
$M(i,j)$
$G_x(i,j)$
$G_y(i,j)$
where n and m are the lengths of sequences A and B, respectively.

Initialization:

$M(0,0) = 0$
$G_x(i,0) = \max(h + g \times i, 0)$
$G_y(0,j) = \max(h + g \times j, 0)$
all other cells: $-\infty$

where h is the penalty for opening a new gap and g is the penalty for extending a gap.

Calculations:

For all i and j, do:

$$M(i,j) = \max \begin{cases} M(i-1,j-1) + s(x_i, y_j), \\ G_x(i-1,j-1) + s(x_i, y_j), \\ G_y(i-1,j-1) + s(x_i, y_j), \\ 0 \end{cases} \tag{4}$$

$$G_x(i,j) = \max \begin{cases} M(i-1,j) + h + g, \\ G_x(i-1,j) + g \end{cases} \tag{5}$$

$$G_y(i,j) = \max \begin{cases} M(i,j-1) + h + g, \\ G_y(i,j-1) + g \end{cases} \tag{6}$$

where s is the scoring matrix used, x_i is the amino acid at position i in sequence A, and y_j is the amino acid at position j in sequence B.

Traceback

Start at largest $M(i,j)$ and follow pointers back to any of $M(0,0) = 0$, introducing gaps in A and B when moving in G_x or G_y respectively, and matches when moving in M.

EMBOSS Implementation

Commonly available implementations of these algorithms are available via the EMBOSS software package [3]. The EMBOSS implementation follow slightly different rules however;

- When opening a gap, there is no gap extension penalty added to the first gap.
- In the cases that a sequence starts with or ends in a gap, no gap opening penalty is added to the starting and/or ending gap.

Method

Both Needleman-Wunsch and Smith-Waterman with affine gap penalties were included in the final script created. Additionally, for ease of comparison, an option was included to run the script with the EMBOSS scoring method. This script was called *aligner.py*.

Some light analysis of time efficiency and method complexity was performed, optimizing the script further. Details can be found in the *Project_supplementary.ipynb* jupyter notebook. The optimized script is called *aligner_optimized.py*. Both instances of the script was kept separately for ease of comparison.

Results

The resulting aligner is complexity $O(nm)$, as derived from tests in *Project_supplementary.ipynb*, and can manage many different kinds of alignments, as described by its help statement:

```
usage: aligner_optimized.py [-h] [-g GAP] [-o OPENGAP] [-e] [-l]
[-m MATRIX] [-c CHARBREAK]
A B

Aligns sequence A and sequence B using affine gap penalties. If
you want to align with regular gap penalties, set the --opengap
flag to 0.

positional arguments:
A                     Sequence A
B                     Sequence B

optional arguments:
-h, --help            show this help message and exit
-g GAP, --gap GAP     Gap extension penalty (should most often
be a negative number).
-o OPENGAP, --opengap OPENGAP
Gap opening penalty (should most often be a negative
number).
-e, --emboss          Use EMBOSS rules for alignment, meaning you
don't receive gap extension when opening a gap, and having
gaps at the end or beginning of a sequence does not
count as opening a gap.
-l, --local           Make a local alignment rather than a global.
-m MATRIX, --matrix MATRIX
Which scoring matrix to use. The matrix should be
saved as a python dictionary in plain-text. Default
behaviour is using the inbuilt PAM250.
-c CHARBREAK, --charbreak CHARBREAK
```

```
How many characters per row you want the output to
have. Default is no linebreaks.
```

A quick analysis of the runtime over aligning randomized sequences of varying length support the complexity prediction, as seen in figure 1.

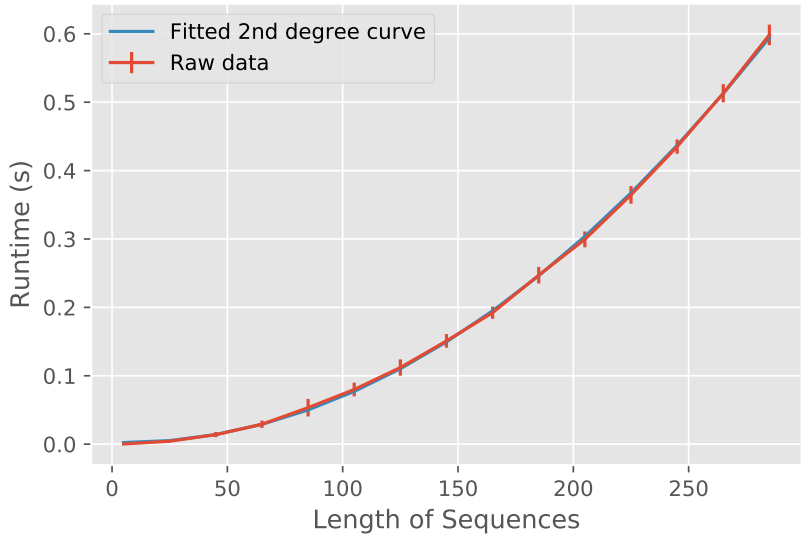

Figure 1: The time for the aligner to finish plotted against the length of the sequences it's tasked to align. All measurements were made with PAM250 matrix, global alignment, open gap penalty of -5, extend gap penalty of -1, and EMBOSS mode turned off. The times are means over 100 random sequences, with the bars showing standard deviation.

Test Cases

Some examples comparing to the official EMBOSS NEEDLE[3]:

Example 1

EMBOSS:

```
#=======================================
#
# Aligned_sequences: 2
# 1: HBA_HUMAN
# 2: HBA_MOUSE
# Matrix: EPAM250
```

```
# Gap_penalty: 5.0
# Extend_penalty: 1.0
#
# Length: 142
# Identity:     122/142 (85.9%)
# Similarity:   136/142 (95.8%)
# Gaps:           0/142 ( 0.0%)
# Score: 609.0
#
#
#=======================================

HBA_HUMAN          1 MVLSPADKTNVKAAWGKVGAHAGEYGAEALERMFLSFPTTKTYFPHFDLS     50
                     ||||..||:|:||||||:|:|::||||||||||||.|||||||||||||:|
HBA_MOUSE          1 MVLSGEDKSNIKAAWGKIGGHGAEYGAEALERMFASFPTTKTYFPHFDVS     50

HBA_HUMAN         51 HGSAQVKGHGKKVADALTNAVAHVDDMPNALSALSDLHAHKLRVDPVNFK    100
                     ||||||||||||||||::|.:|:||:|.||||||||||||||||||||||
HBA_MOUSE         51 HGSAQVKGHGKKVADALASAAGHLDDLPGALSALSDLHAHKLRVDPVNFK    100

HBA_HUMAN        101 LLSHCLLVTLAAHLPAEFTPAVHASLDKFLASVSTVLTSKYR    142
                     ||||||||||:|.||:||||||||||||||||||||||||||
HBA_MOUSE        101 LLSHCLLVTLASHHPADFTPAVHASLDKFLASVSTVLTSKYR    142
```

Output from my own aligner, run with the same penalties (and the EMBOSS flag):

```
SCORE: 609

MVLSPADKTNVKAAWGKVGAHAGEYGAEALERMFLSFPTTKTYFPHFDLS
:::::.::.:.:::::::.:.:..:::::::::::.::::::::::::.:
MVLSGEDKSNIKAAWGKIGGHGAEYGAEALERMFASFPTTKTYFPHFDVS

HGSAQVKGHGKKVADALTNAVAHVDDMPNALSALSDLHAHKLRVDPVNFK
::::::::::::::::::..:.:.::.:.::::::::::::::::::::
HGSAQVKGHGKKVADALASAAGHLDDLPGALSALSDLHAHKLRVDPVNFK

LLSHCLLVTLAAHLPAEFTPAVHASLDKFLASVSTVLTSKYR
:::::::::::.:.::.:::::::::::::::::::::::::
LLSHCLLVTLASHHPADFTPAVHASLDKFLASVSTVLTSKYR
```

Example 2

EMBOSS:

```
########################################
# Program: needle
```

```
# Rundate: Fri 19 Oct 2018 10:41:02
# Commandline: needle
#    -auto
#    -stdout
#    -asequence emboss_needle-I20181019-104058-0584-90315017-p2m.asequence
#    -bsequence emboss_needle-I20181019-104058-0584-90315017-p2m.bsequence
#    -datafile EPAM250
#    -gapopen 5.0
#    -gapextend 1.0
#    -endopen 10.0
#    -endextend 0.5
#    -aformat3 pair
#    -sprotein1
#    -sprotein2
# Align_format: pair
# Report_file: stdout
########################################

#=======================================
#
# Aligned_sequences: 2
# 1: FABPH_HUMAN
# 2: FABP5_MOUSE
# Matrix: EPAM250
# Gap_penalty: 5.0
# Extend_penalty: 1.0
#
# Length: 137
# Identity:      64/137 (46.7%)
# Similarity:    99/137 (72.3%)
# Gaps:           6/137 ( 4.4%)
# Score: 352.0
#
#
#=======================================

FABPH_HUMAN         1 M--VDAFLGTWKLVDSKNFDDYMKSLGVGFATRQVASMTKPTTIIEKNGD     48
                      |  :..:.|.|:|::|..|::|||.||||:|.|::|:|:||..||..:|:
FABP5_MOUSE         1 MASLKDLEGKWRLMESHGFEEYMKELGVGLALRKMAAMAKPDCIITCDGN     50

FABPH_HUMAN        49 ILTLKTHSTFKNTEISFKLGVEFDETTADDRKVKSIVTLDGGKLVHLQKW     98
                      .:|:||:||.|.|.|.:|.:||..|||||||:||..::.|:::|.||:.|:|
FABP5_MOUSE        51 NITVKTESTVKTTVFSCNLGEKFDETTADGRKTETVCTFQDGALVQHQQW    100

FABPH_HUMAN        99 DGQETTLVRELIDGKLIL--TLTHGTAVCTRTYEKEA    133
                      ||:|:|:.|.|.|||:|:  .:.::|  |||.|||..
```

```
FABP5_MOUSE      101 DGKESTITRKLKDGKMIVECVMNNAT--CTRVYEKVQ    135
```

Output from my own aligner, run with the same penalties (and the EMBOSS flag):

```
SCORE: 352.0

M--VDAFLGTWKLVDSKNFDDYMKSLGVGFATRQVASMTKPTTIIEKNGD
:  .....:.:.:..:..:::.:::.:.:..:.:.::..:::...:.
MASLKDLEGKWRLMESHGFEEYMKELGVGLALRKMAAMAKPDCIITCDGN

ILTLKTHSTFKNTEISF--KLGVEFDETTADDRKVKSIVTLDGGKLVHLQ
..:.::.::.:: . :  .::.:::::::.::.....:...:.::..:
NITVKTESTVKTT-V-FSCNLGEKFDETTADGRKTETVCTFQDGALVQHQ

KWDGQETTLVRELIDGKLILT--LTHGTAVCTRTYEKEA
.:::.:.:..:.:.::::..  ....:  :::.::::..
QWDGKESTITRKLKDGKMIVECVMNNAT--CTRVYEKVQ
```

As you can see, these alignments are not identical, but do have identical scores, indicating either a fault within the aligner, or a non-unique solution to the alignment. Since the scores are identical, the latter seems more probable.

Conclusions

The aligner is a relatively efficient implementation of the affine gap modified Needleman-Wunsch and/or Waterman-Smith algorithms. Theoretically, the algorithm has a complexity of $O(nm)$, which is supported by numerical experimentation (see the results section).

Future Work

The current implementation is in Python, and while this makes it easy to modify and read, it also makes it slow. If this implementation should have any real value, it would have to be reimplemented in a compiled language, such as C.

References

[1] Saul B Needleman and Christian D Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of molecular biology*, 48(3):443–453, 1970.

[2] Temple F Smith and Michael S Waterman. Comparison of biosequences. *Advances in applied mathematics*, 2(4):482–489, 1981.

[3] I. Rice, P. Longden and A. Bleasby. Emboss: The european molecular biology open software suite. *Trends in Genetics*, 16(6):276–277, 2000.