Normalisation of Affine Gap Costs used in Optimal Sequence Alignment.

Lloyd Allison,
Department of Computer Science,
Monash University,
Australia 3168.
email: lloyd@cs.monash.edu.au
Partially supported by an Australian Research Council grant.
Feb 1992
revised Sep 1992

Abstract. It is shown how to normalise the costs of an alignment algorithm that employs affine or linear gap costs. The normalised costs are interpreted as the -log probabilities of the instructions of a finite-state edit-machine. This gives an explicit model relating sequences that can be linked to processes of mutation and evolution.

Keywords: affine, alignment, edit distance, gap, indel, linear, model, normalisation.

1. Introduction.

Alignment algorithms are in common use in Molecular Biology and in other disciplines[$Need, $Sank, $Myer] to calculate an edit-distance between a pair of sequences, such as two DNA or protein sequences. A cost is assigned to an alignment by a cost function, often having one or more parameters or weights. An alignment algorithm finds an optimal alignment having minimum cost under its cost function. (Equivalently a similarity score can be assigned and the algorithm finds an alignment having a maximum score.) Functions vary considerably, particularly in the costs assigned to insertions and deletions (indels). Many algorithms use *affine* or linear cost functions, of the form w(L)=a+b*L, for gaps or runs of insertions or deletions of length L where ‘a’ and ‘b’ are constants. It is shown how to *normalise* cost functions of this kind so that they can be interpreted as the -log probability of edit operations carried out by finite-state edit-machines.

Normalisation shows that an explicit, probabilistic, edit-machine lies behind an affine cost function. It extracts probabilities that can be related to processes of mutation and evolution in the problem-domain. It implies that a particular set of parameters is optimal for a given pair or family of sequences thus removing any arbitrariness inherent in the choice of weights. Previously, "it has not been possible to specify appropriate null or gap costs *apriori*" - Altschul[$Alts1]. In principle the approach also implies a rational way of combining a number of machines in an evolutionary tree[$Alli1].

There are at least three approaches to the alignment problem in the literature: **1.** A cost function is chosen and its parameters are adjusted until the alignments are judged to be satisfactory by some means. **2.** A probabilistic model of the combined effects of mutation and evolution is placed on pairs of sequence[$Alli2]. **3.** A probabilistic model of the rates and kinds of mutation is designed and the implied model on pairs of sequences is calculated[$Bish, $Tava, $Thor1, $Thor2]. In the second and third approaches, which are closely related, any parameters of a model can either be chosen apriori or inferred from the sequences. This paper primarily shows how to relate the first approach to the second. The third approach is perhaps the most desirable but there are some mathematical difficulties with the more realistic models. It should also be noted that a recent empirical study by Gonnet et al[$Gonn] suggests that, for proteins, the probability of a gap being of length L is approximately proportional to $L^{-1.5}$ which implies a logarithmic cost function on gap lengths.

2. Simple Costs.

Given two sequences, $\alpha=\alpha_1...\alpha_m$ and $\beta=\beta_1...\beta_n$, an *alignment* is formed by padding out α and β with special *null* characters '-' to $\alpha'=\alpha'_1...\alpha'_p$ and $\beta'=\beta'_1...\beta'_p$ having the same length. Corresponding characters can thus be matched up in pairs - $<\alpha'_1,\beta'_1>...<\alpha'_p,\beta'_p>$. Pairs are usually written vertically:

```
eg. α=ACTACGTACCAGT, β=ACGTAGTCAACGT

An Alignment:
    α′ = AC-TACGT-ACCAGT
         || || ||  | | ||
    β′ = ACGTA-GTCAAC-GT
```

The pair <–,–> is forbidden. The pairs in an alignment can be thought of as representing editing instructions, for a simple *edit-machine*, to change α into β; below we often talk about the instructions in an alignment. The machine reads α and executes the instructions to produce β. A pair <x,x> represents a match or copy instruction of character x from α into β. A pair <x,y> represents an instruction to change character x to y. A pair <x,–> represents a deletion of character x from α and <–,y> represents an insertion of character y into β. Insertions and deletions are called *indels*. Indels and changes are called mutations. A run of insertions or of deletions is called a *gap* because that is how it appears in an alignment. If a cost function is defined on pairs, or equivalently on edit instructions, the cost of the alignment is found by summing over its pairs.

We define *simple* cost functions to be those where the cost of an edit instruction is independent of the context of neighbouring instructions in an alignment. These are special cases of affine costs w(L)=a+b*L where a=0. The well known dynamic programming algorithm[$Wagn, $Sank] finds an optimal alignment in $O(|\alpha|*|\beta|)$ time although there are faster algorithms for special cases, such as for similar strings.

It is almost universal for a match, or a pair <x,x>, to have cost zero. In Sellers' basic edit-distance[$Sell], all mutations have cost one. The corresponding optimal alignments thus demonstrate the minimum number of mutations needed to change α into β. In general the cost of a mutation can depend on the particular characters involved; notable examples are the 'PAM'-matrix[$Dayh] and its recent variations[$Gonn, $Jone] used to cost changes of amino-acids in proteins where the cost of <x,y> depends on the -log probability of y being substituted for x.

The rank-order of alignments for any two sequences, under a given simple cost function, is invariant under certain transformations of the costs. If all costs are multiplied by a positive constant 'k', the cost of each alignment is also multiplied by k and so the rank-order does not change. If a constant 'j' per character is added to each instruction (including matches) then $j*(|\alpha|+|\beta|)$ is added to every alignment's cost and so does not change their rank-order. This means that there are two degrees of freedom in transforming given simple costs while not changing the rank-order of alignments.

It is proposed that the costs be treated as the -log probabilities of the edit instructions in which case the probabilities of the allowed instructions, at a given point, must sum to one. (High probability gives low cost, low probability gives high cost.) This constraint leaves one degree of freedom in transforming given simple costs. This remaining choice can be resolved either from prior expectations on the distribution of pairs of sequences, or adaptively for a particular pair of sequences so as to maximise their likelihood[$Bish, $Thor1, $Alli2].

The independence of instruction costs naturally implies a one-state edit-machine. Having but one state, the probability and thus the cost of each machine instruction is constant. The machine can execute match, change, insert and delete instructions. Match, insert and delete instructions each have one character parameter. A change has two character parameters - which must differ. If for simplicity, all characters are considered to be equally likely and all changes equally likely, the probability of any one character is 1/|Alphabet| and that of any two differing characters is 1/(|Alphabet|*(|Alphabet|–1)). This allows us to normalise Sellers' edit-distance for |Alphabet|=4 say:

```
          original  transformed     normalised
          --------  -----------  ------------------
m match       0          2j      -log(p(m))+2
c change      1          k+2j    -log(p(c))+log(12)
i insert      1          k+j     -log(p(i))+2
d delete      1          k+j     -log(p(d))+2
```

eg. $p(m) = 2\hat{}(-2j+2)$
$k>0$
$p(d)<1 \Rightarrow -\log(p(m))>0 \Rightarrow j>1$

$p(m)+p(c)+p(d)+p(i) = 1$
$\Rightarrow k = -\log((1-2^{-2j+2}) \ / \ (2^{-j+3} + 2^{-2j+\log(12)}))$

Normalising Sellers' Edit-Distance Costs, |Alphabet|=4.

A program was written to run through various values of j and to solve for k. It prints out j and k, the corresponding normalised costs and the instruction probabilities:

		costs			probabilities		
j	k	Cmatch	Cchnge	Ci=Cd	Pmatch	Pchnge	Pi=Pd
1.1	5.61	2.20	7.81	6.71	0.871	0.053	0.038
1.2	4.57	2.40	6.97	5.77	0.758	0.096	0.073
1.4	3.48	2.80	6.28	4.88	0.574	0.154	0.136
2.0	1.87	4.00	5.87	3.87	0.250	0.205	0.273
3.0	0.34	6.00	6.34	3.34	0.062	0.148	0.395

Normalised Sellers' Edit-Distance.

Costs corresponding to small values of j seem more natural perhaps but all costs displayed give the same rank-order of alignments for a given pair of sequences. (There may be a small problem with numerical accuracy when real-valued costs are used where alignments are equally optimal under integer costs.)

3. Affine Costs.

Simple indel costs are thought to be unrealistic for macro-molecules. For example, they give L>1 isolated, one-character indels the same total cost as one gap of L inserts or L deletes. The former are considered to be L separate mutation events and together to be far less probable than the latter single (large) mutation event. To correct this problem, affine gap costs w(L)=a+b*L are used where L is the gap length. The constant 'a' is a gap start-up penalty. Typical values are a=3, b=1. Almost invariably a match still costs zero and a change costs one. Gotoh[$Goto] gave a modified dynamic programming algorithm to find optimal alignments for such costs. Interestingly, affine change costs do not seem to have been considered.

Affine gap costs naturally imply a three-state edit-machine. This machine can execute match, change, insert and delete instructions in each state, but with probabilities depending on the state.

```
S1 :match/change state
S2 :insert state
S3 :delete state

State-Transitions:
   S1--match--->S1   S2--match--->S1   S3--match--->S1
   S1--change-->S1   S2--change-->S1   S3--change-->S1
   S1--insert-->S2   S2--insert-->S2   S3--insert-->S2
   S1--delete-->S3   S2--delete-->S3   S3--delete-->S3
```

NB. $P(i^L|S1) = P(i|S1) \;.\; P(i^{L-1}|S2)$

$$-\log(P(i^L|S1)) = -\log(P(i|S1)) - \log(P(i|S2))*(L-1) \;\;\tilde{}\;\; a+b*L$$

Three-State Machine.

The previous transformations on costs, ie. multiplication by k>0 and addition of j per character, still leave the rank order of alignments unchanged in this case. However they are not sufficient to permit normalisation. The reason is that an insert gap cannot immediately follow an insert gap and a delete gap cannot immediately follow a delete gap. Therefore the probabilities of copy, change and delete (insert) must be higher after an insert (delete) than after a copy or a change. (Note, a delete is allowed to follow an insert and v.v. - see later.) Normalisation is possible with the following observation. The cost of an insert gap of L characters is made up of a+b for the first character and b for each other character. The total cost remains the same if a constant ‘x’ is subtracted from the cost of the first character and added either to that of the last character, or to the cost of the immediately following operation. For long strings, where the effect on the cost of a terminal gap can be ignored, this does not change the rank-order of alignments. Forcing the instruction probabilities in each state to sum to one leaves one degree of freedom and this can be resolved as before.

	original S1	transformed S1	normalised S	transformed S2
m match	0	2j	-log(p(m|S))+2	2j+x
c change	1	k+2j	-log(p(c|S))+log(12)	k+2j+x
i insert	a+b	(a+b)k+j-x	-log(p(i|S))+2	bk+j
d delete	a+b	(a+b)k+j-x	-log(p(d|S))+2	(a+b)k+j

for S = S1, S2 or S3

NB. S3 similar to S2, p(d|S2)=p(i|S3) etc.

p(m|S)+P(c|S)+p(i|S)+p(d|S) = 1, for S=S1, S2, S3
k>0, j>1

Normalising Affine Indel Costs.

A program was written to run through various values of j, given a and b, and to solve for k and x. Results for a=3 and b=1 are given below:

j	k	x	Cmatch	Cchnge	Cinsrt	Cdelet	Pmatch	Pchnge	Pinsrt	Pdelet	
1.1	4.33	0.14	2.20	6.53	18.30	18.30	0.871	0.129	0.000	0.000	-S1
			2.34	6.68	5.43	18.44	0.790	0.117	0.092	0.000	-S2
1.2	3.23	0.29	2.40	5.63	13.84	13.84	0.758	0.242	0.000	0.000	-S1
			2.69	5.93	4.43	14.14	0.618	0.197	0.185	0.000	-S2
1.4	2.07	0.64	2.80	4.87	9.04	9.04	0.574	0.411	0.008	0.008	-S1
			3.44	5.50	3.47	9.68	0.370	0.264	0.361	0.005	-S2
2.0	0.86	0.88	4.00	4.86	4.57	4.57	0.250	0.412	0.168	0.168	-S1
			4.88	5.75	2.86	5.45	0.136	0.224	0.550	0.091	-S2
3.0	0.14	0.17	6.00	6.14	3.39	3.39	0.062	0.170	0.383	0.383	-S1
			6.17	6.31	3.14	3.56	0.055	0.151	0.454	0.340	-S2

Normalised Affine Indel Costs, w(L) = 3.0 + 1.0 * L

Altschul and Erickson[$Alts2] suggest using a=2.5 and b=0.5. It can be seen below that reducing b to much less than 1 markedly increases P(i|S2) (and also P(d|S3)), thus increasing the expected length of gaps. Constant ‘a’ principally governs P(i|S1) and P(d|S1).

j	k	x	Cmatch	Cchnge	Cinsrt	Cdelet	Pmatch	Pchnge	Pinsrt	Pdelet	
1.1	4.34	0.77	2.20	6.54	13.36	13.36	0.871	0.129	0.000	0.000	-S1
			2.97	7.31	3.27	14.13	0.510	0.075	0.414	0.000	-S2
1.2	3.28	1.17	2.40	5.68	9.88	9.88	0.758	0.234	0.004	0.004	-S1
			3.57	6.85	2.84	11.05	0.336	0.104	0.558	0.002	-S2
1.4	2.30	1.60	2.80	5.10	6.71	6.71	0.574	0.349	0.038	0.038	-S1
			4.40	6.70	2.55	8.31	0.190	0.115	0.682	0.013	-S2
2.0	1.17	1.25	4.00	5.17	4.26	4.26	0.250	0.333	0.209	0.209	-S1
			5.25	6.42	2.58	5.51	0.105	0.140	0.667	0.088	-S2
3.0	0.19	0.20	6.00	6.19	3.37	3.37	0.062	0.164	0.386	0.386	-S1
			6.20	6.39	3.10	3.58	0.054	0.143	0.468	0.335	-S2

Normalised Affine Indel Costs, w(L) = 2.5 + 0.5 * L

When a=0 and b>0 the probability of an instruction is the same in each state and the machine is equivalent to a one-state machine. Normalisation in this way does not seem to be possible if b=0.

It was noted that the model permits a delete to immediately follow an insert and v.v. even though this is never seen in an optimal alignment. The reason is that the original costs are not adjusted immediately after an indel to account for this - at least in any algorithm that the author knows of. The search process generally does not propose such alignments, as they cannot be optimal, but they could be costed quite naturally and heavily, if it did. This is not to say that such algorithms and models are right or wrong on this matter - it is just common practice. It has also been suggested that *all* possible alignments should be averaged in a weighted fashion[$Bish, $Thor1, $Thor2, $Alli1, $Alli2, $Yee]. In this case an insert can indeed follow a delete - but with very low probability. This is a different matter from the manipulations involving x which do not change the rank-order of alignments. The possibility of allowing an insert gap to immediately follow another insert gap is yet another matter as it would change the form of the prior on gap lengths and its -log would no longer be affine.

4. Discussion.

It has been shown how to normalise alignment costs based on simple costs and on affine gap costs. They imply one and three-state edit-machines respectively. It is evident that piecewise-linear gap costs[$Goto, $Mill] imply finite-state edit-machines and can be normalised in the same way. Normalisation makes explicit an edit-model that is implicit in common alignment methods. Explicit probabilities are calculated and can be related to processes of mutation and evolution. The validity of the models can be scrutinised and questioned. This is a good thing and can provide the basis of a programme for finding better models. "The vast amounts of data available on DNA sequences suggests that useful models can be developed"[$Tava].

It might be thought that normalisation would require alignment algorithms to use real arithmetic in their inner loop. However, if this is a problem the process can be reversed and integer weights found that give a close approximation to arbitrary instruction probabilities. Alternatively, some favourite costs can be used and the corresponding probabilities calculated.

5. References.

[$Alli1] Allison L., C. S. Wallace & C. N. Yee. Minimum message length encoding, evolutionary trees and multiple alignment.
25^{th} Hawaii Int. Conf. on Sys. Sci. **1** 663-674 1992.

[$Alli2] Allison L., C. S. Wallace & C. N. Yee. Finite-state models in the alignment of macro-molecules.
J. Mol. Evol. **35**(1) 77-89 1992.

[$Alts1] Altschul S. F. Gap costs for multiple sequence alignment.
J. Theor. Biol. **138** 297-309 1989.

[$Alts2] Altschul S. F. & B. W. Erickson. Optimal sequence alignment using affine gap costs.
Bull. Math. Biol. **48**(5/6) 603-616 1986.

[$Bish] Bishop M. J. & E. A. Thompson. Maximum likelihood alignment of DNA sequences.
J. Mol. Biol. **190** 159-165 1986.

[$Dayh] Dayhoff M. O., R. M. Schwartz & B. C. Orcutt. A model of evolutionary change in proteins.
in *Atlas of Protein Sequence and Structure* Vol. 5, Suppl. 3, M. O. Dayhoff ed. 345-352 1978.

[$Gonn] Gonnet G. H., M. A. Cohen & S. A. Benner. Exhaustive matching of the entire protein sequence database.
Science **256** 1443-1445 1992.

[$Goto] Gotoh O. An improved algorithm for matching biological sequences.
J. Mol. Biol. **162** 705-708 1982.

[$Jone] Jones D. T., W. R. Taylor & J. M. Thornton. The rapid generation of mutation data matrices from protein sequences.
Comp. Appl. BioSci. **8**(3) 275-282 1992.

[$Mill] Miller W. & E. W. Myers. Sequence comparison with concave weighting functions.
Bull. Math. Biol. **50**(2) 97-120 1988.

[$Myer] Myers E. W. & W. Miller. Row replacement algorithms for screen editors.
A.C.M. Trans. Prog. Lang. & Sys. (TOPLAS) **11**(1) 33-56 Jan 1989.

[$Need] Needleman S. B. & C. D. Wunsch. A general method applicable to the search for similarities in the amino acid sequences of two proteins.
J. Mol. Biol. **48** 443-453 1970.

[$Sank] Sankoff D. & J. B. Kruskal (eds). *Time Warps, String Edits and Macromolecules: The Theory and Practice of Sequence Comparison.*
Addison Wesley 1983.

[$Sell] Sellers P. H. On the theory and computation of evolutionary distances.
SIAM J. Appl. Math. **26**(4) 787-793 1974.

[$Tava] Tavare S. Some probabilistic and statistical problems in the analysis of DNA sequences. Lect. Math. Life Sci. **17** Am. Math. Soc. 57-86 1986.

[$Thor1] Thorne J. L., H. Kishino & J. Felsenstein. An evolutionary model for maximum likelihood alignment of DNA.
J. Mol. Evol. **33** 114-124 1991.

[$Thor2] Thorne J. L., H. Kishino & J. Felsenstein. Inching towards reality: an improved likelihood model of sequence evolution.
J. Mol. Evol. **34** 3-16 1992.

[$Wagn] Wagner R. A. & M. J. Fischer. The string to string correction problem.
JACM **21**(1) 168-173 1974.

[$Yee] Yee C. N. & L. Allison. Reconstruction of strings past.
Comp. Appl. BioSci. **9**(1) 1-7 Feb 1993.