On the use of don't care regions for protein sequence alignment

Nalvo Franco de Almeida Jr.
nalvo@dct.ufms.br

João Carlos Setubal
setubal@dcc.unicamp.br

Martin Tompa
tompa@cs.washington.edu

Relatório Técnico IC–99-07

Fevereiro de 1999

On the use of don't care regions for protein sequence alignment[*]

Nalvo F. de Almeida Jr. [†‡]
nalvo@dct.ufms.br

João Carlos Setubal [‡]
setubal@dcc.unicamp.br

Martin Tompa [§]
tompa@cs.washington.edu

February 25, 1999

Abstract

We present a general algorithm to align two protein sequences that uses the concept of *don't care* regions. It is based on the classical Needleman-Wunsch dynamic programming method using affine penalty functions. We tested the algorithm on two problems, comparing it against the standard dynamic programming algorithm. Results were not consistently or significantly better, suggesting that refinements are needed.

1 Introduction

We present a new general algorithm to align two character sequences, based on the classical Needleman-Wunsch dynamic programming method [8], using affine gap-penalty functions. The novelty resides in the possibility of designating columns in the alignment as "don't care" columns. Intuitively, the "don't care" columns would form regions where we do not care about the quality of the alignment, in terms of substitutions, insertions or deletions. The algorithm is a generalization of the local dynamic programming alignment algorithm. The main application for such an algorithm is the alignment of distantly related protein sequences, and we start by motivating this application.

An important step in determining the relationship between two proteins is the alignment of their amino acid sequences. Such an alignment can easily be obtained by using the classical Needleman-Wunsch [8] and Smith-Waterman [11] dynamic programming algorithms, and their variants. When the proteins are evolutionarily close such alignments are "good", in the sense that the score obtained is significant and the conserved and nonconserved regions are clearly shown. However, the problem is much harder when the proteins are distantly

[*]Part of this work was done while the first author was visiting the Department of Computer Science & Engineering, University of Washington, Seattle, USA, with funding from CAPES/Fulbright(0627/96). The first two authors are members of Projeto PRONEX 107/97 (MCT/FINEP).
[†]On leave from DCT, UFMS, CP 549, Campo Grande, MS, 79070-900, Brazil.
[‡]Instituto de Computação, UNICAMP, CP 6176, Campinas, SP, 13083-970, Brazil.
[§]Department of Computer Science & Engineering, University of Washington, Seattle, WA 98195, USA.

related, that is, when they are homologous in an evolutionary sense but the amino acid sequence similarity is low. For these cases alignments based only on sequence similarity usually result in low scores, and the alignments themselves are usually quite different from those obtained using structural information, which can be used as a "standard of truth".

The alignments based on structural information show that conserved regions (also known as "core segments") contain relatively few insertions and deletions, whereas nonconserved regions present relatively many insertions and deletions. A better alignment algorithm should thus use this knowledge and apply the usual set of matching penalties only to what is deemed to be a conserved region; nonconserved regions, once detected, should be penalized using a different and less stringent set of penalties. This is what our algorithm does, designating characters in what are considered nonconserved regions as "don't care" symbols in both sequences. We have tested our algorithm using a well-known benchmark from the literature [3]. Results were only marginally better than those obtained with a standard dynamic programming algorithm, suggesting that further work is needed.

The paper is structured as follows. In Section 2 we describe the algorithm. In Section 3 we present results of a simple experiment using a sequence alignment problem. In Section 4 we show the experimental results using the benchmark mentioned above. In Section 5 we briefly comment on other applications that could benefit from our algorithm. The conclusion is in Section 6.

2 The Algorithm

We start by defining a *gap* as a contiguous segment of *spaces*. A space is denoted by the symbol '-'. We use the concept of similarity rather than distance.

As mentioned above, our algorithm is a standard dynamic programming algorithm for comparing two character sequences using affine penalty functions that can operate in two modes. In one mode it applies the usual penalties for gap opening (h) and gap extension (g). Thus a gap with k spaces has score $h+kg$. In the other mode the algorithm uses a don't care region initiation parameter (h'), a don't care region extension parameter, for matches and mismatches (g'), and a don't care gap extension parameter, inside a don't care region ($\bar{g}$). Thus a don't care region with k_1 matches/mismatches and k_2 spaces has total score $h' + k_1 g' + k_2 \bar{g}$. The switch from one mode to the other is simply given by the maximization requirement in the dynamic programming recurrences, to be given below. The use of the two parameters g' and $\bar{g}$ for don't care regions is a compromise between using only one (don't care extension) and three (extension with matches/mismatches, gap initiation, and gap extension).

The input for our algorithm consists of the protein sequences s and t, $|s| = m$ and $|t| = n$, a residue comparison matrix M, giving scores for aligning each pair of residues, and values for the penalty parameters h, g, h', g', and $\bar{g}$. The output is a score and an alignment. Figure 1 shows an example of alignment output.

```
GA-----PVPVDENDEGLQRALQFATAEYNRASNDKYSSRVVQ-VISANRQLVSGIKY-------IL
| *******|   || |     ****  | |    |  *******|  ||    |  |
GEWEIIDIGPFTQNDGGFAVDEENKTGQYGRLTFNKVIRPCMQKTIYENRREIKGYEYQLYASDKLF
```

Figure 1: An alignment with three don't care regions. Each column inside a don't care region is denoted by the symbol `*`, and the symbol `|` denotes an exact match. Note that there can be a match inside a don't care region.

The score and alignment computation is done by standard dynamic programming. We use six arrays A, B, C, D, E, and F. Entry (i, j) in each array contains the maximum score of an alignment of the prefix of s up to s_i with the prefix of t up to t_j such that:

A:	s_i is matched with t_j
B:	a space in s is matched with t_j
C:	s_i is matched with a space in t
D:	s_i is matched with t_j inside a don't care region
E:	a space in s is matched with t_j inside a don't care region
F:	s_i is matched with a space in t inside a don't care region

Noting that $0 \leq i \leq m$, $0 \leq j \leq n$, and that $M(s_i, t_j)$ is the matching score between s_i and t_j given by the residue comparison matrix, the dynamic programming recurrence formulas are as follows:

$$A_{i,j} = M(s_i, t_j) + \max \left\{ \begin{array}{ll} A_{i-1,j-1}, & B_{i-1,j-1}, \\ C_{i-1,j-1}, & D_{i-1,j-1}, \\ E_{i-1,j-1}, & F_{i-1,j-1} \end{array} \right\}, \quad i, j > 0$$

$$B_{i,j} = \max \left\{ \begin{array}{ll} A_{i,j-1} - (h+g), & B_{i,j-1} - g, \\ C_{i,j-1} - (h+g), & D_{i,j-1} - (h+g), \\ E_{i,j-1} - (h+g), & F_{i,j-1} - (h+g) \end{array} \right\}, \quad \begin{array}{c} j > 0 \\ (i,j) \neq (0,1) \end{array}$$

$$C_{i,j} = \max \left\{ \begin{array}{ll} A_{i-1,j} - (h+g), & B_{i-1,j} - (h+g), \\ C_{i-1,j} - g, & D_{i-1,j} - (h+g), \\ E_{i-1,j} - (h+g), & F_{i-1,j} - (h+g) \end{array} \right\}, \quad i, j > 0$$

$$D_{i,j} = \max \left\{ \begin{array}{ll} A_{i-1,j-1} - (h'+g'), & B_{i-1,j-1} - (h'+g'), \\ C_{i-1,j-1} - (h'+g'), & D_{i-1,j-1} - g', \\ E_{i-1,j-1} - g', & F_{i-1,j-1} - g' \end{array} \right\}, \quad \begin{array}{c} i, j > 0 \\ (i,j) \neq (1,1) \end{array}$$

$$E_{i,j} = \max \left\{ \begin{array}{ll} A_{i,j-1} - (h' + \bar{g}), & B_{i,j-1} - (h' + \bar{g}), \\ C_{i,j-1} - (h' + \bar{g}), & D_{i,j-1} - \bar{g}, \\ E_{i,j-1} - \bar{g}, & F_{i,j-1} - \bar{g} \end{array} \right\}, \quad \begin{array}{c} j > 0 \\ (i,j) \neq (0,1) \end{array}$$

$$F_{i,j} = \max \left\{ \begin{array}{ll} A_{i-1,j} - (h' + \bar{g}), & B_{i-1,j} - (h' + \bar{g}), \\ C_{i-1,j} - (h' + \bar{g}), & D_{i-1,j} - \bar{g}, \\ E_{i-1,j} - \bar{g}, & F_{i-1,j} - \bar{g} \end{array} \right\}, \quad i, j > 0.$$

These recurrences are straightforward extensions of the usual dynamic programming recurrences, as found in [10] or [6]. As an example, we explain the recurrences for matrix F. In this case the final alignment column has s_i and a don't-care space, that is, a space inside a don't care region. Previous entries to be considered are those that contain the best alignment of $s_1 \ldots s_{i-1}$ and $t_1 \ldots t_j$. We must check whether the space represents a gap extension or a gap initiation inside the don't care region; if it is a gap extension, there's no need to count the number of spaces, since we do not distinguish between isolated spaces and contiguous spaces in a don't-care region. Therefore, if it is a gap initiation we charge $h' + \bar{g}$; otherwise we charge only $\bar{g}$.

The basis of the recurrence depends on how we want to charge the initial and final gaps. Below we present the initialization for the global-local variant, where the characters in s are not penalized when matched with end spaces in sequence t.

$$\begin{array}{l} A_{0,0} = B_{0,0} = C_{0,0} = D_{0,0} = E_{0,0} = F_{0,0} = 0, \\ A_{i,0} = B_{i,0} = D_{i,0} = E_{i,0} = F_{i,0} = -\infty \ \text{ and } \ C_{i,0} = 0, \quad \text{for } \ i = 1, \ldots, m, \\ A_{0,j} = C_{0,j} = D_{0,j} = F_{0,j} = -\infty, \quad \text{for } \ j = 1, \ldots, n, \\ B_{0,1} = -(h + g), \\ E_{0,1} = -(h' + \bar{g}), \\ D_{1,1} = -(h' + g'). \end{array}$$

When all matrix entries have been computed, the entry that contains the final score depends on the kind of alignment chosen (global, local, or global-local). In the global-local case the desired entry is the maximum value in the last columns of all arrays.

The time complexity of this algorithm is clearly $\Theta(mn)$. The space is also $\Theta(mn)$, since we need a constant number of $m \times n$-arrays.

3 Application: Local Alignment

We present a simple application of our algorithm: obtaining the alignment of two sequences s and t, which we know contain a number of similar subsequences separated by sequences that are not similar. The expected result of a good algorithm for this problem is an alignment in which the similar subsequences are in fact aligned. This is a problem in which we can compare our algorithm to the standard dynamic programming algorithm to see if the don't care feature results in better alignments.

We performed this comparison based on pairs of sequences generated by a computer program with the following parameters: 1) the number of similar subsequences (between 1 and 5; the same number in s and in t); 2) the minimum and maximum sizes of these sequences; 3) the percent of mutations (insertions, deletions and substitutions) in them; and 4) the minimum and maximum sizes of the substrings between the similar subsequences (the total number of these sequences is the number of subsequences to be aligned plus one).

```
               1111111111111 1111111                  22222222222222
HRYDNKH-GDRI---RHDSKNPMVEEYN-AWQLCAN-----MHPEKTDDVGDILAIKYEGEPYMHSRI
***************||| ||||||||| |||||||******************||| ||||||||||
--GHWMYGSMKRGLARHDHKNPMVEEYNMAWQLCANWKDQIEPTGVAFGIFNFHAIKAEGEPYMHSRI
               111111111111111111111                  22222222222222

 222222
-VTKECIEAWMEKACIDYQISRCS
 |||||| ||    |*********
FVTKECI-AWDVREC-----KMLL
2222222
```

```
                    1111111111111 1111111                   22222222
HRYDNKHGDRI---------RHDSKNPMVEEYN-AWQLCANMHPEKTDDVG------DILAIKYEGEP
       |            ||| ||||||||| |||||||         |         ||| ||||
-------GHWMYGSMKRGLARHDHKNPMVEEYNMAWQLCANWK-DQIEPTGVAFGIFNFHAIKAEGEP
                    111111111111111111111                   22222222

222222222222
YMHSRIVTKECIEAWMEKACIDYQISRCS---
||||||           | ||      |
YMHSRI--------FVTKECIAWDVRECKMLL
22222        2222222
```

Figure 2: Two alignments of the same generated pair of sequences. The second alignment was the output from the standard dynamic programming algorithm with $h = 3.3$ and $g = 0.26$ (according [3] these are the best choices for values).

Figure 2 shows the result for a particular instance of this problem. There are 2 similar subsequences to be aligned, and each pair differs in 2 residues (10% mutation). Shown are the alignments obtained with our don't care algorithm and with the standard dynamic programming algorithm. The subsequences marked with numbers are the ones to be aligned.

```
              11111111111111111111              22222222222222222222
KTPHRMSQWQNTKAPILCEAIFALDYSKRISYQS---IDQFTRYGNVVIWNFYASLLFMGDYTNKPQM
**************|||*****||||||||||||**************|||||| |||||||******
GVCPTYQEDEVEAYPILMCEAIALDYSKRISYQSWLMVNFAPLTMFQNIWNFYAQLLFMGDY---NTN
              11111111111111111111              22222222222222   222


MFFCSVCVNGHVSQ
**************
KPQMPEEGPSHYVN
2222


                 11111111111111111111                22222222222222
KTPHRMSQWQNTKA---PILCEAIFALDYSKRISYQS-----IDQFTRYGNVVIWNFYASLLFMGDY-
         |       |||     ||||||||||||         |   |  |||||| |||||||
---GVCPTYQEDEVEAYPILMCEAIALDYSKRISYQSWLMVNFAPLTMFQN--IWNFYAQLLFMGDYN
                 11111111111111111111                222222222222222


222222
TNKPQMMFFCSVCVNGHVSQ
||||||          |
TNKPQM----PEEGPSHYVN
222222
```

Figure 3: Two alignments of another instance of the local alignment problem, with the same parameter values as in the above figure.

Our algorithm obtained a good alignment for both subsequences, whereas the standard dynamic programming algorithm did not align well the second subsequence. However, Figure 3 shows another instance of the problem in which we obtained the opposite result.

We have run experiments on several instances of this problem, experimenting with different sets of values for the parameters, and results were unfortunately not consistently in favor of the don't care algorithm. This shows that refinements are needed in the algorithm, and this is work in progress.

4 Application: Fold Recognition

In this section we describe what should be the main application for our algorithm, the *Fold Recognition Problem*. In this problem the input is a set S of protein sequences with unknown three-dimensional structure and a database T of protein sequences whose structure is known. The desired output is a pairing (s, t) for all $s \in S$ such that $t \in T$, and such that, among all sequences in T, t is the one most structurally similar to s. The entries in T are called *folds*, hence the problem name. The entries in S are called *probes*.

For such a problem to be interesting, the sequence similarity between a probe and its corresponding fold should be low. Algorithms for this problem that rely solely on sequence comparison must thus be able to distinguish between two kinds of low similarity: those that result from the correct alignment between two conserved regions and those that result from unrelated regions. The don't care feature of our algorithm aims precisely at being able to make this distinction.

4.1 A Benchmark

A well-known benchmark for this problem is given by Fischer *et al.* [3], which we now briefly describe. In this benchmark, $|S| = 68$ and $|T| = 301$. The solutions are provided in the form of a set $\mathcal{P}$ of pairs (probe, fold). The set $\mathcal{P}$ was obtained through actual structural comparisons. It provides, according to the authors, the "standard of truth" necessary to assess results. The sets S and T were carefully constructed according to several criteria, one of them being the requirement that all the pairs in $\mathcal{P}$ have approximately 30% of sequence identity or less.

An algorithm applied to this benchmark should output, for each probe, a ranking of all folds in order of decreasing structural similarity, as judged by the algorithm. The performance of an algorithm is measured by an *overall score* R, given by

$$R = \frac{\sum_{i=1}^{|S|} \frac{1}{r_i}}{|S|},$$

where r_i denotes the placement of the correct fold in the ranking output for probe i, as assigned by the algorithm. For a perfect algorithm, $R = 1$, while in general the value will be between 0 and 1.

Fischer *et al.* [3] have tested several algorithms using this benchmark. Comprehensive results can be found in

http://www.doe-mbi.ucla.edu/people/fischer/BENCH/benchmark1.html

4.2 Experimental Set-up

Our algorithm, as described in Section 2 is quite general. In order to test it on the above benchmark several aspects must be specified.

rank	# probes	%
1	41	60.3
2–5	11	16.2
>5	16	23.5

Table 1: Ranks achieved by our algorithm.

We chose to use the global-local version, because in general folds are shorter than probes, and we do not want to penalize end spaces in the probe. The way the ranking is computed is another important issue. Since raw scores of the alignments using local or global algorithms are in general dependent on the length of the sequences [3], methods using these algorithms require some kind of normalization for ranking the scores. On the other hand, in the global-local case, the raw scores do not have such a dependence. So, we used raw scores in this experiment.

The residue comparison matrix used was that of Gonnet *et al.* [4], but with each entry divided by 4. (This division was done by Fischer *et al.* [3], and we did the same to compare our results to theirs.)

Values for gap and don't care penalty parameters were obtained by a brute-force searching method over a reduced set $\mathcal{P}'$ of pairs (probe, fold). The search was over an interval of possible values for each parameter, and testing for each possible combination. The best values obtained were $h = 2.4$, $g = 0.15$, $h' = 2.6$, $g' = 0.18$ and $\bar{g} = 0.2$.

4.3 Results

We obtained an overall score R of 0.68, with 60.3% of correct hits (probe assigned to correct fold).

The benchmark has a difficulty index for each one of the 68 probes, based on tests with available algorithms carried out by the benchmark's authors. That index is the average rank achieved by these algorithms. That is, a probe with difficulty index equal to 1.0 means that this probe is very easy, because all algorithms were able to determine its correct fold. Among the 68 probes, there are 12 such "easy" probes. Our algorithm obtained the correct fold for all these sequences plus 29 other probes with difficulty index greater than one.

Table 1 summarizes our results, giving ranks and respective number and percentage of probes. Table 2 shows the results obtained with the standard dynamic programming algorithm. Its score was 0.66.

These results show that the don't care algorithm performed marginally better than the standard dynamic programming algorithm, a result which is not surprising given the re-

rank	# probes	%
1	40	58.8
2–5	10	14.7
>5	18	26.5

Table 2: Ranks achieved by the standard dynamic programming algorithm.

sults described in the previous section. Experiments with many other parameter values yielded similar results, adding more evidence that the don't care algorithm requires more refinements than simple parameter fine-tuning.

5 Other applications

In this section we outline other potentially interesting applications for our algorithm.

Given two related protein sequences, the problem of determining the most accurate structural alignment between them is different from the fold recognition problem [3, 9, 5, 2]. Lesk *et al.* [7] proposed variable gap penalties plus the use of secondary structure for solving this problem. By using 3 distinct penalties for positions inside a structure, outside a structure, or at the ends of a structure, they were able to obtain good alignments between a new protein, with no structural information, and another with known secondary structure. Our algorithm can be easily modified to apply the same idea, but in a more general form. We can do it by applying weights to specific positions of the known secondary structure. Don't care regions would be determined using these weights plus the usual set of penalties.

Such an approach, if successful, could be used as a clue to predict secondary structures (α-helices and β-strands) from the amino acid sequence of a protein, given another related protein with known structural information.

6 Conclusion

We have proposed a new general algorithm to align two character sequences, based on the classical Needleman-Wunsch dynamic programming method [8], using affine gap-penalty functions. Our contribution resides in the possibility of designating columns in the alignment as "don't care" columns.

As described in the paper, it is clear that refinements are needed before the algorithm performs consistently and significantly better than the standard dynamic programming algorithm (in those applications in which both algorithms can be applied). We have outlined additional applications where our algorithm can be potentially useful. Other possibilities are:

- Using the don't care idea in multiple sequence alignments.
- T. Akutsu [1] studied a variant of the string matching problem in which the pattern string may contain a variable number of don't care characters, though they must be in fixed positions. We would like to adapt our algorithm to solve this problem.

References

[1] T. Akutsu. Approximate string matching with variable length don't care characters. *IEICE Trans. Inf. & Syst.*, E79-D(9):1353–1354, September 1996.

[2] D. Fischer and D. Eisenberg. Protein fold recognition using sequence-derived predictions. *Protein Science*, 5:947–955, 1996.

[3] D. Fischer, A. Elofsson, D. Rice, and D. Eisenberg. Assessing the performance of fold recognition methods by means of a comprehensive benchmark. In *Proc. Pacific Symposium on Biocomputing*, pages 300–318, January 1996.

[4] G. Gonnet, M. Cohen, and S. Benner. Exhaustive matching of the entire potein sequence database. *Science*, 256:1443–1445, June 1992.

[5] M. Gribskov, A. D. McLachlan, and D. Eisenberg. Profile analysis: detection of distantly related proteins. In *PNAS*, volume 84, pages 4355–4358, 1987.

[6] D. Gusfield. *Algorithms on strings, tress and sequences: computer science and computational biology.* Cambridge University Press, Cambridge, 1997.

[7] A. Lesk, M. Levitt, and C. Chothia. Alignment of the amino acid sequences of distantly related proteins using variable gap penalties. *Protein Engineering*, 1(1):77–78, 1986.

[8] S. Needleman and C. Wunsch. A general method applicable to the search for similarity in the amino acid sequence of two proteins. *Journal of Moleular Biology*, 48:443–453, 1970.

[9] W. R. Pearson. Comparison of methods for searching protein sequence databases. *Protein Science*, 4:1145–1160, 1995.

[10] J. C. Setubal and J. Meidanis. *Introduction to computational molecular biology.* PWS Publishing Co., 1997.

[11] T. Smith and M. Waterman. Identification of common molecular subsequences. *Journal of Moleular Biology*, 147:195–197, 1981.