Use-Case Specific Extensions to Global Sequence Alignment with Affine Gap Penalties

Chengyuan Sha
Queen's university
Kingston, Ontario

Tong Liu
Queen's university
Kingston, Ontario

Tristan Samis
Queen's university
Kingston, Ontario

ABSTRACT

Sequence alignment is often the first step in lots of biology and bioinformatics research, such as genome analysis, phylogenetics, and homology between species closely related in evolution. Central to sequence alignment algorithms is the introduction of gaps, which correspond to insertion or deletion mutations. The introduction of gaps must be carefully balanced against the quality of the alignment, a metric that is usually calculated using an affine gap penalty function. This report presents a summary of current sequence alignment algorithms as well as the variety of penalty functions in use. As well, we attempt to make several extensions to existing affine gap alignment algorithms. These extensions make use of additional bioinformatics data about the nature of sequences to be aligned, such as knowledge of existing conserved protein domains within the amino acid sequence, or knowledge of post-transcriptional regions which exist within the nucleotide sequence. We conclude that our extensions resulted in increased performance over basic affine gap penalty alignment when applied to sequences known to have these properties.

ACM Reference Format:
Chengyuan Sha, Tong Liu, and Tristan Samis. 2021. Use-Case Specific Extensions to Global Sequence Alignment with Affine Gap Penalties. *ACM Trans. Graph.* 37, 4, Article 111 (August 2021), 8 pages.

1 INTRODUCTION

The advent of high-throughput sequencing technologies and the availability of entire genomics databases has created a wealth of data at the disposal of bioinformatics researchers. One task of great interest to researchers is the comparison of two nucleotide or polypeptide sequences. Identifying sequences bearing large regions of similarities can be of importance in numerous different contexts and applications, ranging from predictions of protein function to analysis of whole genomes. When trying to understand the molecular structure of an unknown protein, sequence alignment is often one of the first steps performed by researchers. An optimal alignment is achieved when a maximal number of amino acids are matched between two sequences. While direct character-by-character comparisons of sequences may be appropriate for extremely short and highly conserved protein domains, a more robust and generalized approach is required when dealing with whole protein sequences. One approach is to allow the insertion of gaps, which results in greater flexibility in the alignment of two sequences at the cost of increasing the difference (in terms of poly-peptides matched) between those sequences. From an evolutionary perspective, a gap indicates the possibility of an insertion or deletion mutation, with longer gaps corresponding to larger indel mutations. The difference or "penalty" for gapped sequence alignments has traditionally been modelled as a linear function of the length of the gap with some constant intercept representing the penalty for opening the gap. That is, the penalty p of some gap of length n poly-peptides would be modelled by the linear relation $p(n) = a + bn$, where a represents the penalty for opening a gap and b represents the penalty for extending the gap.

An implementation which laid down the groundwork for pairwise alignment of sequences is the Needleman-Wunsch algorithm [Needleman and Wunsch 1970], a dynamic programming algorithm which produces an optimal gapped alignment of two sequences by minimizing the penalty based on the function described above. In this implementation, the optimal alignment of two peptide sequences A and B of length M and N respectively may be represented using a 2D array ARR. Each cell in this array, $ARR_{i,j}$ represents the score associated with the pairwise combination of the i^{th} amino acid in A with the j^{th} amino acid in B. A possible alignment between A and B may be constructed starting from $ARR_{0,0}$ (conventionally beginning at the top left) following some path denoted by ARR_{i_1,j_1} ... ARR_{i_x,j_y} such that $x \leq M, y \leq N$ and each step in the path has a non-decreasing column and row index compared to the previous step. Two maximally aligned (*i.e.*, identical) sequences would have a strictly diagonal path from $ARR_{0,0}$ down to $ARR_{M,N}$. Gaps inserted into the alignment would be represented by increases to either the column or row index, but not both, between two steps. This basic implementation of the Needleman-Wunsch algorithm requires $M \times N$ look-ups in the worst case scenario, resulting in a time complexity in $O(mn)$. Since it fills an array of size $M \times N$, the space complexity is thus also in $O(mn)$.

The simplest method of scoring the alignment is described below. The score $M(i, j)$ is taken as the maximum of either a match or some constant penalty associated with a gap in either of the input sequences denoted by g.

$$M(i,j) = max \begin{cases} M(i-1, j-1) + score(s[i-1], s[j-1]) \\ M(i-1, j) - g \\ M(i, j-1) - g \end{cases}$$

For affine gaps, the scoring of the alignment is modified by the introduction of two scoring matrices denoted by U and L below.

Authors' addresses: Chengyuan Sha, Queen's university, Kingston, Ontario; Tong Liu, Queen's university, Kingston, Ontario; Tristan Samis, Queen's university, Kingston, Ontario.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
© 2021 Copyright held by the owner/author(s).
0730-0301/2021/8-ART111
https://doi.org/

Instead of a constant penalty for gaps, different penalties are assessed for the opening and extension of gaps, denoted by g and e respectively.

$$U(i,j) = max \begin{cases} M(i,j-1) - g \\ U(i,j-1) - e \end{cases}$$

$$L(i,j) = max \begin{cases} M(i-1,j) - g \\ L(i-1,j) - e \end{cases}$$

$$M(i,j) = max \begin{cases} M(i-1,j-1) + score(s[i-1], s[j-1]) \\ L(i,j) \\ U(i,j) \end{cases}$$

The alignment produced by the algorithm is largely dependent on the penalties associated with the opening and extension of gaps in the input sequences. As an amendment to the linear gap penalty function with constant opening and extension values, more robust implementations of Needleman-Wunsch calculate penalties using custom match and extension weights. These weights are derived from two factors. Firstly, the biochemical nature of amino acids must be considered, as substitutions of one amino acid by another with similar properties may not disrupt protein function. Namely, a mutation resulting in the replacement of lysine for arginine (two charged, hydrophilic amino acids) may not disrupt protein folding and the overall function of the peptide sub-unit. However, a mutation resulting in the replacement of lysine (a charged, hydrophilic amino acid) for leucine (a hydrophobic amino acid) will be more disruptive to the folded structure of the poly-peptide. This impacts the calculation of amino acid sequence alignments from an evolutionary perspective, in that certain pairwise substitutions (such as lysine for arginine) in a given alignment may still be scored as a partial match, whereas others (such as lysine for leucine) will be heavily penalized. Secondly, the redundant nature of the genetic code dictates that certain amino acids appear more frequently as a result of being represented by a larger set of distinct codons. For instance, serine has an observed frequency of 8.1% whereas tryptophan has an observed frequency of 1.3% among vertebrates. In the context of gap penalties, this means that a higher weight should be given to the alignment of tryptophan than serine between two sequences. Researchers have been able to combine these two observations to produce a table of substitution frequencies known as BLOck SUbstitution Matrix (BLOSUM), which contains the log-odds score for all 210 possible substitutions between the 20 amino acids.

However, improvements can still be made to the base implementation of the Needleman-Wunsch algorithm. The quadratic time complexity of the algorithm is a barrier when calculating alignments on long sequences of genomic DNA. Improvements can be made to the algorithm both in generalizable applications as well as in specific applications. Researchers should be able to make use of known properties of input sequences, and using this information, tailor the algorithm so that the run-time efficiency of the algorithm is better than polynomial.

2 RELATED WORK

2.1 Local Alignment vs Global Alignment

Numerous amendments have been made to the Needleman-Wunsch algorithm [Needleman and Wunsch 1970] in a variety of different capacities. The Waterman-Smith algorithm [Smith and Waterman 1981] is a modified version which is used when local alignment is desired over global alignment of two sequences. The two algorithms are similar in that gaps are scored using some affine gap penalty function and both are able to make use of dynamic programming principles to improve run-time efficiency. The key distinction lies in that scores in the Waterman-Smith alignment matrix cannot be negative. Instead, the algorithm begins at the highest-scoring match and terminates once a cell with a score of 0 is encountered. In practice, this produces an optimal local alignment (as opposed to the global alignment from Needleman-Wunsch).

2.2 Non-Linear Gap Penalty Functions

The implementation of an algorithm based on local alignment has been expanded upon by Gu [Gu and Li 1995] and Mott [Mott 1999], by changing the scoring function. Gu and Li found that a logarithmic affine gap penalty, $p(n) = a + b \ln n$ was more appropriate when finding local that local alignments over long sequences of genomic DNA, because of the existence of long regions of introns. They stated that a purely linear gap penalty does not represent the evolutionary incidence of insertion/deletion mutations in that it overly punishing the extension of very long gaps. As well, they suggest that the gap opening penalty for insertions should be larger than that of deletions, as the proportion of deletion mutations is larger than that of insertion mutations in nature. Overall, Gu and Li improved upon the original affine gap penalty function by making it more suitable for use with long sequences of genomic DNA. Mott expanded upon the idea of custom affine gap penalties by implementing a monotonic, non-linear penalty function. The main idea of this implementation was to maintain a candidate list of values which maximize the score of gaps inserted. The performance of this improvement is most noticeable when long gaps are expected. Both of these approaches dealt with improving the accuracy of the alignment of local exonic regions in the context of large genomics sequences.

2.3 Variable Gap Penalty Function

Building further upon the idea of customized gap penalties, Madhusudhan *et al.* introduced a function in order to compute a completely variable gap penalty [Madhusudhan et al. 2006]. The function varies with respect to whether the mutation is an insertion or deletion, and penalizes gaps which are introduced in conserved domains. These include peptide sub-units which regulate the secondary structure of the protein, buried regions, and straight segments. The parameters of the function were trained on a set of 238 paired sequences with known peptide structures and conserved regions. Using this variable gap penalty function, an improvement in alignment accuracy from 81% to 84.5% over the standard affine gap penalty function was noted. This improved represents an approach to calculating gap penalties which is heavily based on bioinformatics knowledge available to researchers. The parameters of a variable

gap penalty function could be further optimized by re-training over larger sets of proteomics data using more powerful hardware.

Further work on generalized affine gap costs has been done by Altschul [Altschul 1998]. The improvements to the gap penalty function were based on the premises that conserved protein domains tend to fall into ungapped blocks which are then divided by regions of introns or untranslated sequences. A simple generalization was applied to the gap penalty function which allows intron regions to be gapped entirely at little cost to the alignment score. The soundness of this approach is supported by the fact that the distribution of scores from local alignments is extreme, an observation which coincides with the fact that conserved domains tend to fall in between long regions of introns. This generalization has been shown to improve both the alignment accuracy and the alignment score for local alignments of long genomic sequences.

The idea of a generalizable affine gap penalty has been further expanded on by Zachariah *et al.* [Zachariah et al. 2005]. Similar to Altschul, they derived a set of parameters from phylogenetic and homology data sets. They found that the generalized affine gap model performs just as well as the traditional affine gap alignment in detection of related structures in evolutionarily related species. While the generalized model aligned fewer amino acids compared to the traditional alignment model, the per-residue accuracy was higher. They concluded that this generalized extension is applicable in situations where the accuracy of the alignment is more desirable than the length of sequences produced in the alignment [Thompson et al. 1994].

2.4 Multiple sequence alignments

Extensions on the number of input sequences accepted by the algorithm has also been explored, and has been used extensively by bioinfomatics researchers. Compared to pairwise sequence alignment, for which dynamic programming algorithms exist that always provide the optimal alignment, multiple sequence alignments are more complex and suffers from high input dimensionality. The problem of multiple sequence alignment has been shown to be NP-complete [Wang and Jiang 1994]. Thus, finding the optimal alignment between numerous input sequences is only feasible on small datasets. Instead, multiple sequence alignment algorithms make use of heuristics and approximations to reduce the computational requirements to find an acceptable alignment. One of the most commonly used implementations of a multiple sequence alignment algorithm is Clustal [Higgins and Sharp 1988]. The complex problem of multiple sequence alignment is approached by "progressive alignment". The problem is separated into smaller sub-problems of a series of pairwise sequence alignments. A guide tree is constructed based on the phylogenetic relation between the sequences. The optimal pairwise alignment can be obtained, but a greedy approach must be used when traversing the tree. Thus, it is possible that the optimal alignment for multiple sequences is only a local maximum.

2.5 Other applications

Numerous applications of affine gap penalty exist outside of a bioinfomatics and proteomics context. These applications range from $diff$-like UNIX functions to plagiarism detection. One implementation of affine gap penalty outside in the comparison of run-length encoded strings was explored by Kim *et al.* [Kim et al. 2008]. They demonstrated that optimization of alignment scores from an affine gap penalty function could be transformed into a path problem on a directed acyclic graph. The performance of the algorithm implementing affine gap penalty scoring was comparable with other algorithms performing string comparisons in terms of performance. For example, compared to the well studied "longest common subsequence" algorithm, both scoring metrics have a time complexity in $O(nm' + n'm)$, where n and m are the length of the input sequences.

3 APPROACH

The Needleman-Wunsch algorithm finds the optimal global alignment for two strings with a given gap penalty. Since we could not improve upon the alignment result of the original algorithm, we opted to implement extensions to increase the ease of use, promote flexibility, and optimize run-time in general situations. With this approach we hope to create an algorithm that could be used for much broader applications than the original, with a smaller resource cost. We then built test-case experiments to compare our extended algorithm against the original, to highlight the strengths of our algorithm in areas we feel the original is weak.

3.1 Detailed result visualization

The first extension to the base algorithm that we considered was making the results more readable. Examining the alignment in plaintext was very difficult for long alignments, and we wished to alleviate that problem. The first measures we added to the output were similarity and gap percentages. These percentages indicate what percent of the aligned sequence are exact matches and gaps. These measures should allow for a rough idea of how good the alignment was without examining the entire alignment. The second measure we added in order to improve the usability of the algorithm was colour coding the alignment. While gaps were already the most readable part of the alignment, we coloured them cyan to promote quick identification. On the other hand, finding areas of low conservation, where there were mismatched pairings, was very difficult. We colour coded all pairings with a score greater than or equal to zero to be green, while all negative-scoring pairings were coloured red. In this way we hope to greatly improve the readability of the sequences aligned by our algorithm.

3.2 Custom weighting

The catalytic site of a protein, and the nucleotides that encode them, are often highly conserved across species. These sites require precise three dimensional orientation and specific biochemical properties in order to perform their functions in the cell. However, these sites do not make up the bulk of the protein sequence, and structural elements unrelated to a catalytic function are more susceptible to mutations. While global alignment will find the best possible alignment of an entire string against a reference, we wish to add the ability to define a protein motif in order to conserve that sequence and prevent gaps of being opened at that location. This way it will be possible to check the score of an alignment in which a suspected motif is forced to align. If the forced motif score is very low, it is likely that the alignment is not correct. However, if the score of the forced motif is very similar to the optimal alignment score,

this might indicate that your gap weighting or alignment matrix is resulting in a conserved area being split.

We plan to implement this by allowing the user to input a custom amino acid sequence with a custom alignment score. We will then go into the scoring matrix and change the scores such that the score for matching the correct characters will be equal to the custom alignment score. This will tend the algorithm to create gaps in order to keep those characters aligned in the final output. We feel this will conserve the weighted characters, at the cost of creating more gaps in the sequence. We will then test the original algorithm against our extended algorithm on alignments with a specific sub-sequence inserted manually, in order to see the difference our extension has on the alignment.

3.3 Semi-global alignment

A global alignment cause cause issues if the two strings are very different sizes. There are multiple potential practical situations where this could occur. For example, trying to find a gene against a genome, finding the best attachment site for a polymerase chain reaction primer, aligning a single read with an assembly, or placing a marker onto a chromosome. In all these cases, we are querying a smaller sequence against a much larger one. In these cases, attempting a global alignment could cause the smaller sequence to be split quite heavily along the larger one to maximize the alignment score. Consider the following example:

```
      CAGCGTGG
...CAGCACTTGGTTCTCGG...
```

We have the above sequences, one of which is much smaller than the other, and we wish to align them. We know the smaller one is a highly conserved region, and we do not want it to be split up. If we use our regular algorithm to generate an alignment, we get the following result:

```
...CAGCACTTGGTTCTCGG...
   CAGC-----GT----GG
```

Two gaps have been opened in the sequence, in order to perfectly align each nucleotide in the smaller sequence. We will use a semi-global implementation of our algorithm to solve this problem. We implement this by initializing the first row and first column of the dynamic programming matrix to 0. Additionally, the penalty for opening a gap from the first row and first column will be set to 0. Finally, instead of forcing the algorithm to trace a path from A(0,0) to A(N,M), we will allow the program to start on any index in the first row or column, and end on any index in last row or column. As a result of these changes, we would expect our result to look like this:

```
...CAGCA-CTTGGTTCTCGG...
   ---CAGCGTGG-------
```

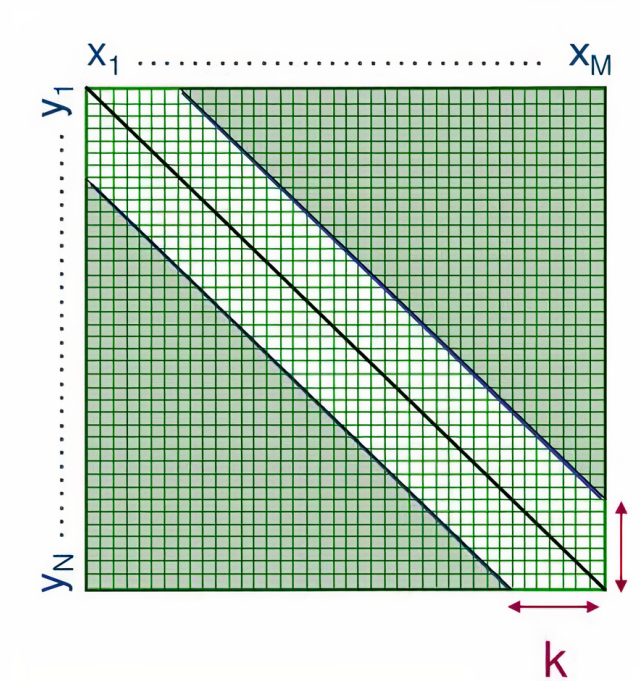
As can be seen above, there are three total gaps in this alignment. With the original algorithm we would expect this to cost more than an alignment with two gaps. However, because two of the gaps are at the beginning and end of a string, they incur no penalty. In this way we extend our algorithm's ability to match smaller strings against much larger ones.

We will test this functionality by comparing smaller peptide substrings against larger ones, using the original algorithm and our extended semi-global alignment algorithm. We will compare the output sequences, as well as the scores they receive in order to ensure the functionality of the feature.

3.4 Bounded dynamic programming

For the final extension to increase the usability of our algorithm, we plan to implement a variation which enables bounded dynamic programming. Due to the global nature of the original algorithm, the space and time complexity are both $O(mn)$, where m is the number of rows and n is the number of columns. This makes it an extremely inefficient for large alignments. However, we can make use of certain properties of the algorithm in order to lower the run-time. In the case of two sequences that are similar in length and similar in sequence, the optimal traceback of the score is likely to run along the central diagonal of the scoring matrix.

As can be seen in Figure 1, we can provide a variable that will be used used to restrict the scoring algorithm to some distance k from the central diagonal line running through the alignment matrix. The reason we wish to add this feature to our algorithm is that deviating by a large margin from the central axis indicates a large insertion or deletion in the alignment. While the original alignment algorithm and our extension utilize three matrices in order to implement an affine gap penalty, the principle is still applied. We simply enforce the restriction on all three matrices rather than one.



Source: MIT Course - Algorithms for Computational Biology

Figure 1: Visualization of the k-value restriction

We implement this by adjusting the rules for updating the alignment score matrix for affine gap penalties, adding a condition to the dynamic programming loop, and changing the way we initialize the matrix. We will initialize the upper and lower scoring matrices, which refer to insertions and deletions in the alignment, with a default value of *minInt*. Then we adjust the loop condition to

```
for i = 1 to M
    for j = max(1, i - k) to min(N, i+k)
```

so that we only iterate over the indices that matter for the algorithm. Finally, we adjust the update rules for the upper and lower matrices. If the original rule for updating the upper matrix that holds the score when character $sequence_1(i)$ matches a gap at $sequence_2(j)$ is represented as follows, where $sequence_1(i) = s(i)$ and $sequence_2(j) = t(j)$, g is the opening penalty, e is the extension penalty, and the upper and middle matrices are denoted by U and M:

$$U(i,j) = max \begin{cases} M(i,j-1) - g \\ U(i,j-1) - e \end{cases}$$

Our implementation is simply to add a condition to the update rule:

$$U(i,j) = max \begin{cases} M(i,j-1) - g & \text{iff j > i - k(N)} \\ U(i,j-1) - e & \text{iff j > i - k(N)} \end{cases}$$

By applying this rule to the update functions for the upper and lower matrices, it can be seen that the matrix will not be updated if the index would fall outside the boundary. this means that all of these indices will keep their initialization value of *minInt*, and not need to be updated by the algorithm. We expect this to greatly decrease the run-time of the algorithm on sequences which are similar in length. While the original algorithm ran in $O(mn)$ time, in this extension for every m and n we check k values. As such, the algorithm now runs in $O((m+n)k)$ time. However, by applying this change we no longer guarantee that the algorithm will find an optimal alignment of the two sequences. We will test this extension by comparing the run-time of the original algorithm against the extension we created.

4 RESULT & DISCUSSION

4.1 Custom weighting experiment

In this experiment, our goal is to compare the custom weighted algorithm with the regular algorithm. We use -11 as the gap opening penalty, -1 as the gap extension penalty for both algorithms. We assume "S" "T" "A" "Y" as the strongly conserved amino acids and add a large positive custom weight to it so that it is more likely to be used in any given alignment. The results are shown in Figure 4. Although there are more gaps present, the conserved sequences "S" "T" "A" "Y" tends to align in our modified algorithm.

4.2 Semi-global alignment experiment

Our modified algorithm allows for semi-global alignment, which is a variant of global alignment that allows for gaps at the beginning and/or the end of one of the sequences. We test it on a modified mRNA sequence. In the central dogma of biology, after being transcribed from DNA, pre-mRNA goes through stages of processing before being sent to the ribosomes for protein construction. Part of this process is known as RNA splicing. During RNA splicing, exons (coding regions) are joined together and introns (non-coding regions) are removed. Also, specially altered nucleotides will be added to the 5' end of the mRNA (5' cap) and a poly(A) tail comprised of many adenine nucleotides will be attached to the 3' end of RNA (polyadenylation). Our experiment mimics an attempt to align a sequence of the original mRNA with the processed RNA so we can understand modified regions. We use -15 as the gap opening penalty, -2 as the gap extension penalty as the algorithm parameter. The Alignment results are shown below:

```
Semi-global alignment:
Align score 2
Similarity: 13 / 28 :  53.57 %
Gaps: 14 / 28 :  50.0 %
GCAGCACTTGGATTCTCGG---------
-CAGCG--TGGA--CTCGGAAAAAAAAA

Original alignment:
Align score 2
Similarity: 10 / 24 :  58.33 %
Gaps: 6 / 24 :  25.0 %
GCAGC----ACTTGG-ATTCTCGG
-CAGCGTGGACTCGGAAAAAAAAA
```

When using semi-global alignment, we could "jump" over the five-prime cap and/or poly(A) tail. This extension allows our algorithm to be used with "messy" data, where the sequences have not necessarily be isolated and cleaned through experimentation.

Additionally, semi-global alignment can be useful in all kinds of problems where one sequence is significantly shorter than the other. For example, we can use semi-global alignment to find a specific gene in a genome by placing splicing a marker into a specific position on the chromosome and find a best-aligned primer to a DNA sequence.

4.3 Bounded dynamic programming experiment

We first compared the efficiency between unbounded and bounded algorithms. Using -15 as the gap opening penalty, -2 as gap extension penalty and 10 as the bound size, we iterated through DNA bases ranging from 10 to 600 (inclusive) in a step size of 10. For each iteration, we repeat each alignment method 10 times and took the average to reduce the noise from the timed measurements. The algorithm was run on a computer with Intel(R) Core(TM) i9-10900F CPU @ 2.80GHz 2.81 GHz and 64.0 GB RAM. The results are shown in Figure 2. The graph demonstrates that time increases much faster in the unbounded algorithm as the DNA length increases, due to the polynomial nature of the affine gap alignment algorithm. The

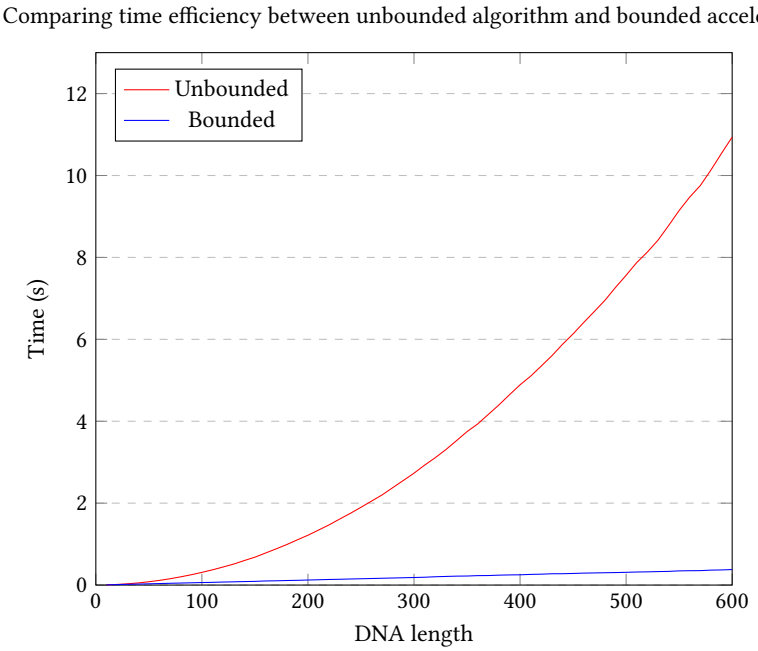


Figure 2: Two sets of DNA were generated randomly from length 10 to 600 (inclusive) in a step size of 10 and used by unbounded and bounded alignment algorithms. The time of execution has been recorded at each length.

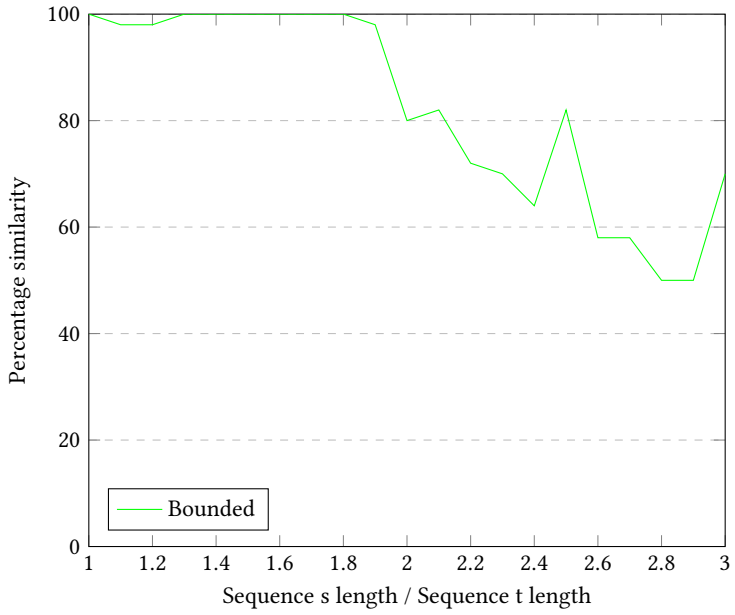


Figure 3: The x-axis is the difference factor between sequence s and sequence t. It is generated from 1 to 3 (inclusive) in a step size of 0.1. The y-axis is the percentage of the number of same scores between bounded algorithm and unbounded original algorithm at different DNA length.

```
Extended Global Sequence Alignment:
Scoring matrix used: BLOSUM62.
Align score 1344
Similarity: 72 / 137 :  47.45 %
Gaps: 56 / 137 :  40.88 %
TAK----YNPGASTA----STAYYTLCGEDHFNNIS------STAYCQGAWAVQFN-NQPSSTAYTAYTTQKECVKIHMHLQT-----ESSTAY----------ATLLDQMHASMEYESTSTAYAY--PYASTAHY
TANYSTAYNPGASTAYTRRSTAY---CGEDHFHNISCQGAWASTAY------VQFNCNDLHSTAY-QYTTQK---------STAYECYKISTAYHMHKCYAQTVSAT--DQMHASM--ESTSTAYAYYEPYASTAHY


Original Global Sequence Alignment:
Scoring matrix used: BLOSUM62.
Align score 251
Align score 251
Similarity: 77 / 127 :  39.37 %
Gaps: 36 / 127 :  28.35 %
TAK----YNPGASTA----STAYYTLCGEDHFNNISSTAYCQGAWA----VQFN-NQPSSTAYTAYTTQK----ECVKI----HMHLQTESSTAYATLLDQMHASMEYESTSTAYAY--PYASTAHY
TANYSTAYNPGASTAYTRRSTAY---CGEDHFHNIS----CQGAWASTAYVQFNCNDLHSTAY-QYTTQKSTAYECYKISTAYHMH-KCYAQTVSAT--DQMHASM--ESTSTAYAYYEPYASTAHY
```

Figure 4: Comparison between our custom weighted algorithm and the unmodified algorithm using a conserved "s", "t", "a" and "y" and weight of 30.

unbounded algorithm shows a quadratic growth, whereas bounded algorithms increase at a linear rate.

Because the bounded algorithm doesn't guarantee the global optimal solution, in practice, we want to know the situations in which we can confidently apply this optimization. Assuming there is a randomly generated DNA sequence with length s and another sequence with length t. The bound size is automatically computed using $10 + abs(len(s) - len(t))$. And the length of DNA s equals the length of DNA t times the difference factor. The difference factor is from 1 to 3 (inclusive) in a step size of 0.1. At each difference factor, we repeat bounded and unbounded alignment using DNA t and s 50 times. Then we compute the percentage of the same scores in bounded and unbounded algorithms. The results are shown in figure 3. When the difference in the length of the input sequences is less than a factor of 2, the bounded and unbounded algorithms produce alignments with similar accuracy. We can explain this behaviour by considering that large horizontal or vertical movement on the alignment matrices would correspond to the opening of large gaps in the sequences. Because the algorithm uses a gap penalty, it well tend to prefer small sequences of mismatches to opening small gaps. Because the algorithm does not wish to open many gaps, the optimal global alignment would be predicted to occur near the diagonal axis of the matrix. This means that our bounded algorithm has a high chance of finding the global optimal alignment when there are few gaps, such as the case above. However, as the difference increases, the quality of the alignment produced by the bounded algorithm decreases noticeably. Thus, we recommend users only use the bounded algorithm when the difference in two input sequence lengths is less than a factor of 2.

5 CONCLUSION

The sequencing and analysis of DNA, RNA, and amino acid sequences have been an instrumental part of biological research following the advent of high throughput sequencing technologies and widely accessible genome databases. The first step taken in the analysis of newly sequenced proteins and poly-peptide sub-units is often a comparison with known structures. Global and localized sequence alignment algorithms have been developed for this purpose. One of the salient features of these algorithms is the introduction of gaps in the input sequences to produce better alignments. The introduction of gaps corresponds to insertion or deletion events which have occurred between two sequences which bear evolutionary similarities. A scoring function has been developed in order to represent the trade-off between flexibility via the introduction of gaps and alignment accuracy. One of the most commonly used scoring algorithms is the affine gap penalty function, which is modelled by two parameters: the penalty associated with the opening of a gap and that associated with the extension of an existing gap. Many improvements have been made to the ground-level implementations of global and local sequence alignments using affine gap penalty. These improvements varied from the introduction of non-linear scoring functions, to completely variable scoring functions, to techniques for the alignment of multiple sequences simultaneously. These improvements were generally founded upon additional knowledge regarding the nature of the input sequences. Non-linear gap penalty functions have been shown to produce higher quality alignments for genomic sequences containing long intron regions. Similarly, knowledge about conserved domains and mutation frequencies has been incorporated into variable gap penalty functions.

In this report, we claimed to make three improvements upon the affine gap penalty alignment algorithm. Firstly, we introduced a custom weight scheme for gap penalties introduced within a user-specified sequence. The intent behind this improvement is to allow users to input a sequence which represents a conserved domain and to modify the penalty weights within this sequence. This should improve the accuracy of the alignment in that any known conserved domains will not be gapped in either of the input sequences. Through the experiment, it can be seen that alignments were made involving the specified custom sequence while maintaining a high-scoring global alignment over the remainder of the sequence. Next, we modified the algorithm to allow for semi-global alignments. The semi-global sequence alignment was designed to

insert long gaps at the ends of sequences to accommodate specifically for post-transcriptional processing of messenger RNA strands. Based on experimental data, the algorithm produced high-scoring alignments over the exonic regions of the sequence. Lastly, we attempted to improve the efficiency of the scoring algorithm by introducing bounding. The introduction of bounding with the dynamic programming algorithm greatly improved the efficiency of the algorithm, as seen through the decrease in running time. However, this optimization is limited in that the algorithm is no longer guaranteed to find the best scoring alignment. The quality of the alignment is more impacted when the difference in length between the two sequences is large, as the bounds are more likely to affect the alignment. For sequences whose lengths are within a factor of two, a bounded dynamic programming approach could be a suitable candidate in finding an optimal global alignment. In conclusion, we find that extensions based on information of conserved domains and exonic regions, or about the nucleotide structure of the sequence will tend to lead to better performance than basic affine gap alignment algorithms. However, the applications of these extensions are limited in the same way. Future work could be focused on more generalizable extensions, such as those appropriate for a class of protein domains.

REFERENCES

Stephen F Altschul. 1998. Generalized affine gap costs for protein sequence alignment. *Proteins: Structure, Function, and Bioinformatics* 32, 1 (1998), 88–96.

X. Gu and W. Li. 1995. The size distribution of insertions and deletions in human and rodent pseudogenes suggests the logarithmic gap penalty for sequence alignment. *Journal of Molecular Evolutio* 40, 4 (1995), 464–473. https://doi.org/10.1007/BF00164032

Desmond G Higgins and Paul M Sharp. 1988. CLUSTAL: a package for performing multiple sequence alignment on a microcomputer. *Gene* 73, 1 (1988), 237–244.

Jin Wook Kim, Amihood Amir, Gad M. Landau, and Kunsoo Park. 2008. Computing similarity of run-length encoded strings with affine gap penalty. *Theoretical Computer Science* 395, 2 (2008), 268–282. https://doi.org/10.1016/j.tcs.2008.01.008 SAIL – String Algorithms, Information and Learning: Dedicated to Professor Alberto Apostolico on the occasion of his 60th birthday.

M.S. Madhusudhan, Marc A. Marti-Renom, Roberto Sanchez, and Andrej Sali. 2006. Variable gap penalty for protein sequence–structure alignment. *Protein Engineering, Design and Selection* 19, 3 (01 2006), 129–133. https://doi.org/10.1093/protein/gzj005 arXiv:https://academic.oup.com/peds/article-pdf/19/3/129/4371642/gzj005.pdf

R Mott. 1999. Local sequence alignments with monotonic gap penalties. *Bioinformatics* 15, 6 (06 1999), 455–462. https://doi.org/10.1093/bioinformatics/15.6.455 arXiv:https://academic.oup.com/bioinformatics/article-pdf/15/6/455/9732043/150455.pdf

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

TF. Smith and MS. Waterman. 1981. Identification of common molecular subsequences. *Journal of molecular biology* 147, 1 (1981), 185–197.

Julie D Thompson, Desmond G Higgins, and Toby J Gibson. 1994. Improved sensitivity of profile searches through the use of sequence weights and gap excision. *Bioinformatics* 10, 1 (1994), 19–29.

Lusheng Wang and Tao Jiang. 1994. On the complexity of multiple sequence alignment. *Journal of computational biology* 1, 4 (1994), 337–348.

Marcus A. Zachariah, Gavin E. Crooks, Stephen R. Holbrook, and Steven E. Brenner. 2005. A generalized affine gap model significantly improves protein sequence alignment accuracy. *Proteins: Structure, Function, and Bioinformatics* 58, 2 (2005), 329–338. https://doi.org/10.1002/prot.20299 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/prot.20299