Multiple Sequence Alignment: A Brief Introduction

Anoop Sarkar
for CMPT-825: Natural Language Processing

This is a brief summary of some of the issues in multiple sequence alignment (or in other words, computing the minimum edit distance between multiple strings). The description a summary taken from the references listed at the end of this document.

1 Definitions

The multiple sequence alignment (MSA) problem is a more general form of the standard alignment problem between a pair of strings. Let $S_1, \ldots, S_k$ be the input set of strings (sequences) as assume that K is at least 2. The minimum edit distance is defined as $d(S_1, \ldots, S_K)$

Let Σ be the alphabet. We assume that the underscore character ``_'' is not part of the alphabet so that we can represent alignments based on insertions/deletions in the original string.

A multiple sequence alignment A is defined as a two-dimensional character array over the alphabet Σ', where $\Sigma' = \Sigma \cup \{\_\}$. Given an input set of strings $S_1, \ldots, S_k$, the alignment array A has K rows and each row A_i is the alignment for string S_i. Each MSA induces a pairwise alignment $A_{i,j}$ on the pair of sequences S_i, S_j.

Example: Let $S_1 =$ `charge`, $S_2 =$ `barge` and $S_3 =$ `sarge`. Then the MSA is represented as the alignment array A shown below with the minimum edit distance $d(S_1, \ldots, S_K)$:

```
c h a r g e
_ b a r g e
_ s a r g e
```

2 Simple Algorithm

The obvious method for computing the alignment array A is to extend the $\mathcal{O}(m \times n)$ algorithm for two strings S_1, S_2 of length m and n respectively. The algorithm uses a two-dimensional edit-distance matrix of size $m \times n$.

We can extend this algorithm to K input strings $S_1, \ldots, S_k$ of length $l_1, \ldots, l_K$. Create a K-dimensional matrix with each dimension i of size l_i. The algorithm is similar to the 2-dimensional case, except it will have K `for` loops. The complexity will be $\mathcal{O}(l_1 \times \ldots \times l_K)$.

3 Single Source – Single Destination Shortest Path

One obvious method to improve the running time of the trivial MSA algorithm given above is to use a single-source shortest path algorithm (like Dijkstra's algorithm) instead of the straightforward $\mathcal{O}(m \times n)$ algorithm. In the case for two strings, the intuition is that for the problem of aligning a pair of strings, you can consider each cell of the $m \times n$ array as a node in a weighted graph with the edit distance costs as the weights on the arcs.

While this is overkill for two strings, the average case speedup means that it pays off for multiple strings. For MSA, the graph is represented with vertices at coordinates in K-space and the string characters at the coordinate axes. A path P from vertex $s = \langle 0, \ldots, 0 \rangle$ that ends at vertex $t = \langle n_1, \ldots, n_K \rangle$ represents an alignment of the K strings.

For the example above, assuming standard Levenshtein distances, the first two edges are:

$$\langle 0, 1, 1 \rangle \rightarrow \langle 2, 2, 2 \rangle \rightarrow \langle 0, 0, 0 \rangle$$

A standard heuristic to improve the average case time and space is to use the A^* algorithm to prune away very high cost edit distance alignments, even though they could ultimately end up in the global best minimum edit distance alignment. Another more refined heuristic to prune away unlikely alignments is the Carrillo-Lipman heuristic. The Durbin, Eddy, et al. book on bioinformatics is a good source of information about MSA algorithms.

4 Carrillo-Lipman Heuristic

The algorithm for MSA is still to computationally intensive for large values of K and for long strings. Hence, in practice, various heuristics are used to provide an answer that is close to optimal but not guaranteed to be optimal, while providing massive speed up in computation.

One heuristic used by Carrillo and Lipman is search only among those alignments whose cost is $\leq U$ for some fixed value U. The value of U is determined by using edit-distance scores for pairwise alignments between strings S_i, S_j for all $i < j$. A lower bound L is defined as:

$$L = \sum_{i<j} d(S_i, S_j) \cdot \mathsf{scale}(S_i, S_j)$$

$d(S_i, S_j)$ is the optimal pairwise alignment score (computed using the standard pairwise edit-distance algorithm). $\mathsf{scale}(S_i, S_j)$ is used to make sure everything is scaled up to the longest string in the multiple strings in the input (let's assume that the scaling is 1 here).

The value of L is a lower bound on the cost of MSA because it assumes that each pair of sequences is aligned optimally, independent of the other sequences.

Carroll and Lipman realized that not all alignments A need to be considered. Recall that $d(S_1, \ldots, S_K)$ is defined as the *optimal* minimum edit distance between the K input strings. Let $\mathcal{A}$ be the alignment for $d(S_1, \ldots, S_K)$. L is the sum of pairs lower bound (defined above) and U be an upper bound on $d(S_1, \ldots, S_K)$ (note that U is user-defined) and let $d(\mathcal{A}_{i,j})$ be the global best minimum edit distance for the pairs S_i and S_j. Then:

$$U - L \geq \sum_{i<j} d(\mathcal{A}_{i,j}) - d(S_i, S_j)$$

For all alignments A, $d(\mathcal{A}_{i,j}) \geq d(S_i, S_j)$. Rearranging, we get the Carrillo and Lipman bound

$$d(\mathcal{A}_{i,j}) \leq d(S_i, S_j) + U - L$$

This constraint on values for the optimal alignment $\mathcal{A}$ is used to throw away any alignments that do not satisfy this bound. Carroll and Lipman show that $\Omega(K^2)$ terms are thrown away and while the heuristic might still find the optimal sequence, the difference $d(\mathcal{A}_{i,j}) - d(S_i, S_j)$ can be overestimated by a factor $\mathcal{O}(K^2)$.

For very small values of U, the heuristic may not find any feasible solution. The only way out in this case is to re-run the algorithm with increasing values of U. However, the larger the value of U, the more space and time the algorithm will take.

References

S. K. Gupta, J. Kececioglu, A. A. Schäffer. Making the Shortest-Paths Approach to Sum-of-Pairs Multiple Sequence Alignment More Space Efficient in Practice. Proceedings of the 6th Annual Symposium on Combinatorial Pattern Matching, 1995.

H. Carrillo and D. Lipman. The multiple sequence alignment problem in biology. SIAM J. Appl. Math, 48:1073-1082, 1988.