Appendix A: The Candidate-List Method

In this appendix, we will describe the candidate-list method of Miller and Myers (1988) as is implemented for log-affine gap costs in Ngila. In a candidate-list method, the alignment algorithm maintains a list of potential gaps for each column and row. The search is enhanced by removing gaps from the list when it is possible to show that other gaps will always prove to be better. The method of Miller and Myers (1988) provides an efficient way of calculating when gaps should be removed from the list, if the cost of a gap is monotonic. Ngila uses the monotonic log-affine gap cost, $w_k = a + bk + c \ln k$. We will focus on a single row when describing this method.

For this method, our goal is to globally align two sequences, X and Y, using dynamic programing and find the alignment with lowest cost. This will be done by progressively filling in a table, C, such that $C(x, y)$ is the minimum cost of an alignment of subsequence $X_{1\ldots x}$ and subsequence $Y_{1\ldots y}$, where $0 \le x \le N_X$ and $0 \le y \le N_Y$. Note that subsequences $X_{1\ldots 0}$ and $Y_{1\ldots 0}$ are considered to be empty.

Focusing on row y, let $C_x = C(x, y)$ and $N = N_X$. Now, the total cost of a horizontal gap of length k ending at position x, y is calculated as $G(x, y, k) = C_{x-k} + w_k$ or alternatively as $C_{x'} + w_{x-x'}$, where $x - x' = k$. The alignment algorithm maintains two sorted lists: $\xi \subseteq \{0 \ldots N-1\}$ and $\chi \subseteq \{1 \ldots N\}$. The former is the list of candidates for gap starting positions. The latter is an associated list of crossing points that are used to cull the list when a new candidate is added.

This lists are updated as the algorithm proceeds down the row. When $x = 0$, both lists are empty, and when $x = 1$, the list is $\xi_1 = \{0\}$ and $\chi_1 = \{N\}$. When $x = 2$, the lists will be updated only if $x' = 1$ is a better gap starting position than $x' = 0$ for some $x \in \{2 \ldots N\}$. This will be true if and only if $C_1 + w_1 < C_0 + w_2$ or $C_1 + w_{N-1} \le C_0 + w_N$. (See Miller and Myers (1988) for a proof of this). Assuming that this is true, then $\xi_2 = \{1\}$ and $\chi_2 = \{N\}$ if $C_1 + w_{N-1} \le C_0 + w_N$, and $\xi_2 = \{1, 0\}$ and $\chi_2 = \{x^\star(0, 1), N\}$ otherwise. Here, $x^\star(0, 1)$ is the point at which gap cost curve starting at $x' = 1$ crosses the curve starting at $x' = 0$. For log-affine gap costs,

$$x^\star(p, q) = p + \left\lfloor \frac{q - p}{1 - \exp\left(\frac{C_p - C_q + b(q-p)}{c}\right)} \right\rfloor$$

At $x = 3$, updating the list becomes more complicating because $x' = 2$ needs to be compared against each candidate, and candidates that are dominated by $x' = 2$ need to be eliminated from the list. This comparing and culling is provided by the second method of Miller and Myers (1988), which is shown in the following pseudocode.

```
if x == 1 then xi = {0}; chi = {N}; return; endif
if x > chi[1] then shift(xi); shift(chi); endif
if gcost(x-1, x) < gcost(xi[1], x) or gcost(x-1, N) <= gcost(xi[1], N) then
    while |xi| > 0 and ( gcost(x-1, chi[1]) < gcost(xi[1], chi[1]) or
            gcost(x-1, N) <= gcost(xi[1], N) ) do
        shift(xi); shift(chi); done
    if |xi| == 0 then unshift(chi, N); else unshift(chi, xstar(xi[1],x-1)); endif
    unshift(xi, x-1);
endif
```

Appendix B: Divide and Conquer

The divide-and-conquer algorithm of Hirschberg (1975) provides a way of calculating maximum alignments in linear space. In this algorithm, the traceback table is replaced by successive alignments of subsequences. We will use a modified version of divide-and-conquer to demonstrate this.

Let C contain row y of the alignment table, and C' contain row $y - 1$. Furthermore, let Ξ contain the set of $N_X + 2$ candidate lists needed for the Miller and Myers (1988) algorithm. Now it can be easily shown

that all the information for filling in C is contained by the set of C', Ξ, X, Y, and the alignment parameters. After C has been filled in, setting $C' = C$ allows the algorithm to procede to the next row. When the final row has been filled out, the algorithm can determine whether the last event in the alignment is a homology, a gap on X, or a gap on Y. Next, the algorithm realigns X and Y excluding the nucleotides that were covered by the last event. For example, if the last event in the best alignment of X and Y is a homology, then the next step of the algorithm aligns $X_{1 \cdots N_X - 1}$ and $Y_{1 \cdots N_Y - 1}$, finding its final event. This continues until there are no more nucleotides that need to be aligned.

This algorithm for finding the best alignment has achieved a linear space complexity at a cost of increasing its time complexity because the majority of the sequence will be aligned multiple times. The cost of the algorithm can be improved by making the split in the middle and not at the end of the best alignment. This is achieved by filling in the top half of the table using a forward algorithm and the bottom half of the table using a backward algorithm. When the two algorithms meet, the best middle event can be chosen.

References

Hirschberg, D. S. (1975). A linear space algorithm for computing maximal common subsequences. *Comm. ACM.*, **18**, 341–343.

Miller, W. and Myers, E. W. (1988). Sequence comparison with concave weighting functions. *Bull. Math. Biol.*, **50**, 97–120.