Project Development Phase

Utilization Of Algorithms,Dynamic Programming,Optimal Memory Utilization

Team Id	NM2023TMID04415
Project Name	Block Chain Technology For Electronic Health Records

Background:

Covariance models (CMs) are probabilistic models of RNA secondary structure, analogous to profile hidden Markov models of linear sequence. The dynamic programming algorithm for aligning a CM to an RNA sequence of length N is $O(N^3)$ in memory. This is only practical for small RNAs.

Results:

I describe a divide and conquer variant of the alignment algorithm that is analogous to memory-efficient Myers/Miller dynamic programming algorithms for linear sequence alignment. The new algorithm has an $O(N^2 \log N)$ memory complexity, at the expense of a small constant factor in time.

Conclusions:

Optimal ribosomal RNA structural alignments that previously required up to 150 GB of memory now require less than 270 MB.

From guide tree to covariance model:

The final CM is an array of *M* states, connected as a directed graph by transitions $t_v(y)$ (or probability 1 transitions $v \to (y,z)$ for bifurcations) with the states numbered such that $(y,z) \geq v$. There are no cycles in the directed graph other than cycles of length one (e.g. the self-transitions of the insert states). We can think of the CM as an array of states in which all transition dependencies run in one direction; we can do an iterative dynamic programming calculation through the model states starting with the last numbered end state *M* and ending in the root state 1. An example CM, corresponding to the input alignment of Figure Figure1,1, is shown in Figure Figure33.

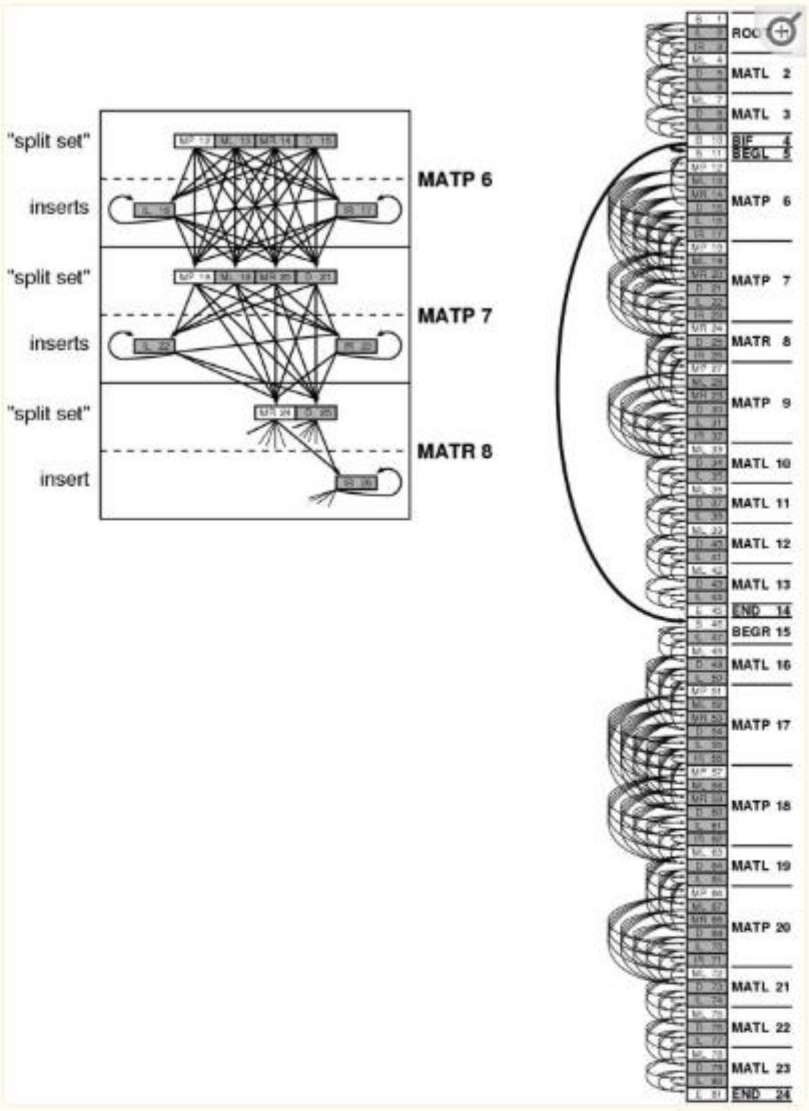

Figure 3

Parameterization:

Using the guide tree and the final CM, each individual sequence in the input multiple alignment can be converted unambiguously to a CM parse tree, as shown in Figure Figure4.4. Counts for observed state transitions and singlet/pair emissions are then collected from these parse trees. The observed counts are converted to transition and emission probabilities by standard procedures. I calculate maximum a posteriori parameters, using Dirichlet priors.

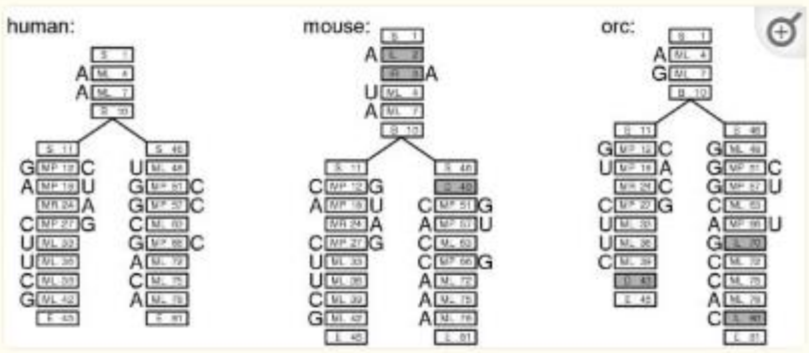

Figure 4

Using CYK/inside and CYK/outside to divide and conquer:

- The bifurcation-dependent strategy is a special case of this more general splitting strategy, where the B state is the only member of its split set, and where we also take advantage of the fact that $\alpha_v(i,j) = \max_k \alpha_w(i, k) + \alpha_y(k + 1,j)$. By carrying out the $\max_k$ operation during the split, rather than before, we can split the current problem into three optimal pieces instead of just two.

- If we look at the consequences of these splitting strategies, we see we will have to deal with three types of problems (Figure (Figure55):

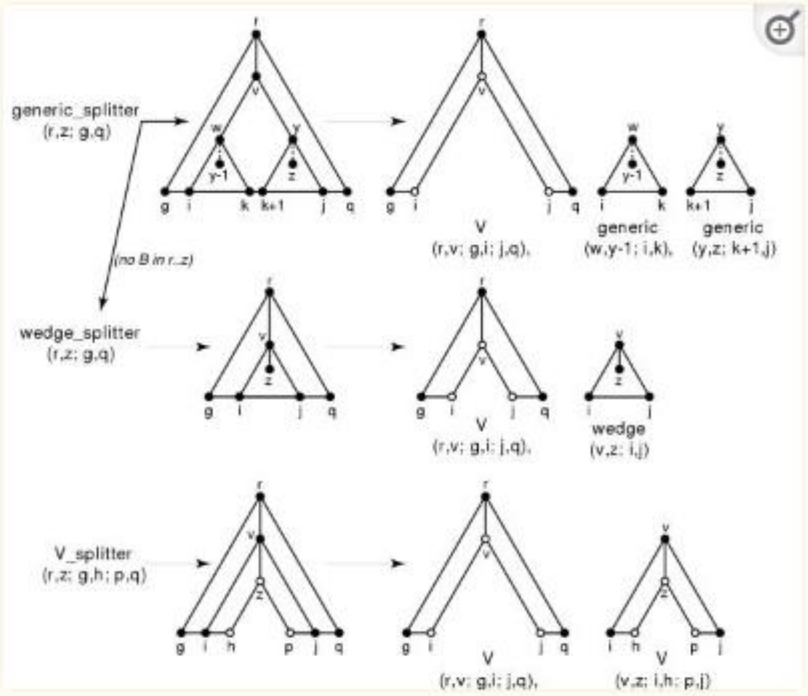

Figure 5

Empirical results:

The same results are plotted in Figure Figure6.6. Memory requirements scale as expected: N^3 for standard CYK alignment, and better than $N^2 \log N$ for the divide and conquer algorithm. Empirical CPU time requirements scale similarly for the two algorithms ($N^{3.24}$ – $N^{3.29}$). The observed performance is better than the theoretical worst case of $O(N^4)$. The proportion of extra time required by divide and conquer is roughly constant over a wide range of RNAs. The difference shown in

Figure Figure66 is exaggerated because times are plotted for score-only CYK, not complete CYK alignment, in order to include CPU times for SSU and LSU rRNA. Because score-only CYK does not keep a shadow traceback matrix nor perform the traceback, it is about 20% faster than CYK alignment, as seen in the data in Table Table44.

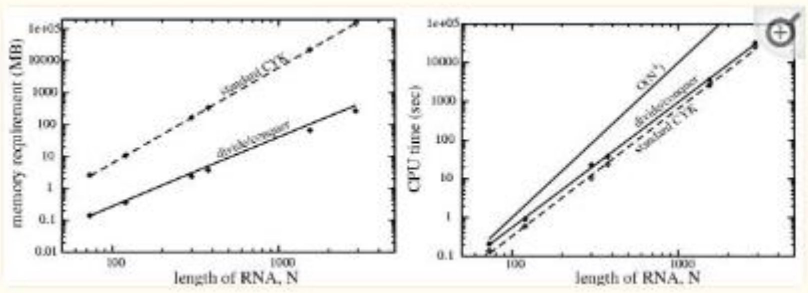

Figure 6