Mauve: Supplementary algorithm description

Aaron C. E. Darling

March 29, 2004

Appendix A: Description of the Multi-MUM search algorithm

The multi-MUM search algorithm described herein is a seed-and-extend method based on the method that can identify both multi-MUMs occurring in all genomes under study in addition to those occurring only in subsets of the genomes being searched. The multi-MUM search algorithm has time complexity $O(G^2n + Gn \log Gn)$ where G is again the number of genomes and n the length of the longest genome. Further, the random-access memory requirements are proportional to the number of multi-MUMs found, not n, allowing it to efficiently tackle large data sets. $O(Gn)$ disk space is used to store sequentially accessed data structures.

The algorithm proceeds by constructing a sorted list of k-mers for each genome $g \in G$. The sorted k-mer lists are then scanned to identify k-mers that occur in two or more sequences but that occur at most once in any sequence. If a multi-MUM that subsumes the k-mer match has not yet been discovered, then the match seeds an extension in each genome until a mismatch occurs. When a mismatch occurs an extension is seeded

in the subset of sequences that are still identical, but only if a subsuming multi-MUM has not yet been discovered.

Given a match seed, a key feature of our algorithm is its ability to efficiently determine whether an existing multi-MUM subsumes the seed. Mauve uses a hash table to track known matches. The hash function $h(M)$ for a match M yields a quantity we refer to as the *generalized offset* of a match M. Using the notation of multi-MUMs introduced in the primary manuscript, $h(M)$ can be written as $h(M) = \sum_{j=1}^{G} |M.S_j - M.S_1|$. In order to mitigate the effects of potential hash collisions, each bucket of the hash table uses a binary search tree to store matches.

For the purposes of time complexity analysis, the matching algorithm can be deconstructed into four primary components: Sorted Mer List (SML) construction, seed match identification, seed lookup in the known match hash table, and seed extension. SML construction can be accomplished in $O(Gn)$ (linear) time using radix sort methods. Identifying seed matches from the Sorted Mer Lists requires a single sequential scan through each SML and is thus also $O(Gn)$.

The seed lookup phase can be executed at most once for every multi-MUM seed. Because there are Gn mers, the largest possible number of unique mer-matches is $\frac{Gn}{2}$. If all of these mer-matches were to hash to the same bucket then a tree search and insertion would be required for every seed match. Using a splay tree (Sleator and Tarjan, 1985), the amortized time complexity for Gn tree lookups and insertions is $O(Gn \log Gn)$.

The amount of match extension depends on the number and size of multi-MUMs identified. Because we are identifying MUMs, each nucleotide can be a part of at most 2 MUMs on the forward strand and 2 MUMs on the reverse strand, for a total of 4 MUMs. Furthermore, it holds that any

nucleotide can be a part of at most 4 multi-MUMs with a given *multiplicity*. Thus each nucleotide can be a part of $4G$ multi-MUMs, or just $O(G)$ multi-MUMs. For a given multiplicity m, the largest possible amount of extension work depends on the maximum possible number of matching mers at that multiplicity: $\frac{Gn}{m}$. Further, each extension at a particular multiplicity m requires m character comparisons. Thus the maximum number of character comparisons for a given multiplicity is $m\frac{Gn}{m}$ or just Gn, and since there are G multiplicity levels, the maximum number of comparisons to find all multi-MUMs is G^2n

By adding the contributions each of the algorithm's four components make toward the total running time, we arrive at $Gn + Gn + Gn \log Gn + G^2n$. In asymptotic notation, the Gn terms are subsumed by G^2n, leaving $O(G^2n + Gn \log Gn)$. It is important to note that although suffix tree algorithms provide better asymptotic time complexity than our seed-and-extend method, in practice our implementation is very fast and space efficient. Furthermore, the seed matching technique can be easily modified to use weighted/spaced seeds, allowing inexact string matching not possible with suffix tree-like data structures in the same low asymptotic time complexity.

Appendix B: Partitioning M into collinear subsets

As part of the anchor selection process, Mauve must partition the initial set of multi-MUMs into collinear subsets. To do so, Mauve implements a breakpoint analysis algorithm based on the description of breakpoints given by Blanchette et al. (1997), We refer to the resulting collinear sets of multi-MUMs as LCBs. An LCB can be defined formally as a maximal collinear

subset of the matches in $\mathbf{M}$, or $lcb \subseteq \mathbf{M}$ where M_i is the i^{th} multi-MUM in the LCB. The MUMs that constitute an LCB must satisfy a total ordering property such that $M_i.S_j \leq M_{i+1}.S_j$ holds for all i, $1 \leq i \leq |lcb|$ and all j, $1 \leq j \leq G$.

Mauve uses a standard breakpoint determination algorithm to partition the set of multi-MUMs into a set of LCBs. First, Mauve orders the multi-MUMs in $\mathbf{M}$ on $|M_i.S_0|$. Next, a monotonically increasing label between 1 and $|\mathbf{M}|$ is assigned to each MUM corresponding to the index of the MUM in the ordering on $|M_i.S_0|$. We will refer to the label of the i^{th} multi-MUM as $M_i.label$. Note that $M_i.label \in \mathbb{N}$. Next, the set of multi-MUMs is repeatedly reordered based on $|M_i.S_j|$ for $j = 2...G$. After each reordering, the set of multi-MUMs are examined for breakpoints. A breakpoint exists between M_i and M_{i+1} if $M_i.label + 1 \neq M_{i+1}.label$ and both M_i and M_{i+1} are in the forward orientation, or if $M_i.label - 1 \neq M_{i+1}.label$ and both M_i and M_{i+1} are in the reverse complement orientation. A breakpoint also exists if M_i is in a different orientation than M_{i+1} in sequence j, e.g. the sign of $M_i.S_j$ is different than the sign of $M_{i+1}.S_j$.

Finally, the multi-MUMs are re-ordered on $M.label$ and the LCBs are then any maximal length subsequence of multi-MUMs $M_i...M_{i+j}$ that does not contain any recorded breakpoints between multi-MUMs.

References

Blanchette, M., Bourque, G., and Sankoff, D. (1997). Breakpoint Phylogenies. *Genome Inform Ser Workshop Genome Inform*, 8:25–34.

Sleator, D. D. and Tarjan, R. E. (1985). Self-adjusting binary search trees. *J. ACM*, 32(3):652–686.