High-Performance Algorithm Engineering for Gene-Order Phylogenies

David A. Bader, Bernard M.E. Moret, Tandy Warnow, Stacia K. Wyman, and Mi Yan

UNIVERSITY OF NEW MEXICO AND UNIVERSITY OF TEXAS AT AUSTIN

Abstract

Phylogenies derived from gene order data may prove crucial in answering some fundamental questions in biomolecular evolution. Yet very few techniques are available for phylogenetic reconstruction based upon gene order and content, and these are (for the most part) computationally expensive. High-performance algorithm engineering offers a battery of tools that can reduce, sometimes spectacularly, the running time of existing approaches. We discuss one such application, in which we started with the method known as “breakpoint analysis” (developed by Sankoff and his colleagues) and produced a software suite, GRAPPA, that demonstrated a million-fold speedup in running time (on a variety of real and simulated datasets), by combining low-level algorithmic improvements, cache-aware programming, careful performance tuning, and massive parallelism. We show how these techniques are directly applicable to a large variety of problems in computational biology.

Introduction.

Some organisms have a single chromosome or contain single-chromosome organelles (mitochondria or chloroplasts), the evolution of which is mostly independent of the evolution of the nuclear genome. Given a particular strand from a single chromosome (whether linear or circular), we can infer the ordering of the genes along with the directionality of the genes, thus representing each chromosome by an ordering of oriented genes. The evolutionary process that operates on the chromosome may include inversions and transpositions, which change the order in which genes occur in the genome as well as their orientation; and also insertions, deletions, or duplications, which change the number of times and the positions in which a gene occurs.

A natural optimization problem for phylogeny reconstruction from such data attempts to reconstruct a tree with a minimum number of permitted evolutionary events. Versions of this problem are all known or conjectured to be NP-hard. Another approach is to estimate leaf-to-leaf distances between all genomes and then use a standard distance-based method (such as neighbor-joining) to construct the tree. Such approaches are fast and may prove valuable in reconstructing the underlying tree, but cannot recover the ancestral gene orders.

Sankoff and his colleagues developed a technique, breakpoint analysis, for reconstructing a phylogeny based on gene-order data and implemented it in the code BPAnalysis. Within a framework that enumerates all trees, BPAnalysis uses an iterative heuristic to label the internal nodes with signed gene orders. The outer loop enumerates all $2^n - 5$ leaf-labelled trees on $n$ leaves, an exponentially large value. The inner loop runs an unknown number of iterations (until convergence), with each iteration solving an instance of the NP-hard Travelling Salesperson problem (TSP) at each internal node. Our simulation studies suggested that the approach worked well for many datasets, but that the BPAnalysis software was too slow. For instance, we estimated that BPAnalysis would require several centuries of computation on a typical workstation in order to solve a collection of chloroplast data from 12 species of Campanulaceae (plus one outgroup), each genome consisting of 105 gene segments. Although an exhaustive search of tree topologies is clearly impossible for more than 15 genomes (there are $0.2 \cdot 10^{15}$ trees on 16 genomes), even selective exploration of tree space requires very fast labeling of the internal nodes of a tree.

We therefore decided to produce a new software suite, GRAPPA (Genome Rearrangements Analysis under Parsimony and other Phylogenetic Algorithms), based on the approach of Blanchette and Sankoff, but using high-performance algorithm engineering to make it fast enough to be useful. High-performance algorithm engineering is a combination of low-level algorithmic improvements, data structures improvements, and coding strategies that combine to eliminate bottlenecks in the code, balance its computational tasks, and make it cache-sensitive. We also used massive parallelism to gain an additional speedup. The code for GRAPPA can be obtained from http://www.cs.unm.edu/~moret/GRAPPA/ and is freely available under the GPL.

Specifics.

We improved bounding, both within the exact TSP solver and at the level of tree enumeration—the latter by considering circular orderings rather than trees. We condensed genomes—that is, we grouped subsequences of gene fragments that appear in all genomes into a single “supergene,” thereby considerably reducing the size...
of instances. This condensing is done initially in a static pass and then dynamically after each alteration of the set of candidate labels. We scored trees incrementally in constant time. We took advantage of the special nature of the instances of the TSP generated in the reduction to speed up the exact solution of these instances. We paid special attention to the initialization phase, in which an initial label is assigned to each node of the current tree topology.

Algorithm engineering suggests a refinement cycle in which the behavior of the current implementation is studied in order to identify problem areas, which are then eliminated through algorithmic or coding changes. Constant profiling is the key to such an approach, because the identity of the principal “culprits” in time consumption changes after each improvement, so that attention must shift to different parts of the code during the process—including revisiting already improved code for further improvements.

The original BPAnalysis has a significant memory footprint and poor locality: it requires over 60MB of space and 12MB for its working set when running on our Campanulaceae dataset. GRAPPA has a tiny memory footprint (1.8MB on the same dataset) and mostly good locality (nearly all of our storage is in arrays preallocated in the main routine), which enables it to run almost completely in cache (the working set size is 600KB). Cache locality can be improved by returning to a FORTRAN-style of programming, in which records (structures/classes) are avoided in favor of separate arrays, in which simple iterative loops that traverse an array linearly are preferred over pointer dereferencing, in which code is replicated to process each array separately, etc.

Results.

BPAnalysis is highly flexible and includes various TSP solvers. In order to estimate the raw speed of each method, we ran all methods on real and synthetic datasets, letting them process thousands to hundreds of thousands of trees until a fixed time bound was attained. We then normalized the results and, since the greedy TSP solver was always the fastest, computed ratios of processing rates for the other four methods against the rate of the greedy method. Table 1 shows the results; in the table, \( n/N/r \) denotes a problem with \( n \) genomes, \( N \) genes, and \( r \) inversions per model tree edge; the first two data sets are the full Campanulaceae dataset and its first 8 members, respectively. The figure shown for the greedy method is the actual processing rate of that method, in trees processed per second. The high processing rate of our exact solver (we have observed rates from 100 to 5,000 times faster than BPAnalysis) makes it possible to solve problems with 10–12 genomes on a single processor within minutes to hours. We then parallelized the code and used Los Lobos, the 512-processor Linux supercluster at The University of New Mexico’s Albuquerque High-Performance Computing Center, to achieve a million-fold speedup on the Campanulaceae dataset: a complete breakpoint analysis (with inversion distances) for the 13 genomes in that set ran in less than 1.5 hours!

Ongoing Developments.

Our current implementation also permits (in the same running time) a search for the minimum inversion phylogeny, and we are extending GRAPPA to be able to handle a larger variety of events (deletions, insertions, duplications, transpositions, etc.). We have also developed new algorithmic strategies for solving the median phylogeny, based upon fixed-parameter approaches in which we bound the number of events on each edge. Our talk will present these new developments, as well as describe in detail the algorithmic engineering used in designing and improving GRAPPA. We will also show how the techniques we used in making the code efficient and in parallelizing it are applicable to a wide variety of algorithmic approaches in phylogeny and other areas of computational biology.