The Journal of Supercomputing, 22, 99–111, 2002
© 2002 Kluwer Academic Publishers. Manufactured in The Netherlands.

High-Performance Algorithm Engineering for Computational Phylogenetics

BERNARD M. E. MORET moret@cs.unm.edu
Department of Computer Science, University of New Mexico, Albuquerque, NM 87131

DAVID A. BADER dbader@eece.unm.edu
Department of Electrical and Computer Engineering, University of New Mexico, Albuquerque, NM 87131

TANDY WARNOW tandy@cs.utexas.edu
Department of Computer Sciences, University of Texas, Austin, TX 78712

Abstract. A phylogeny is the evolutionary history of a group of organisms; systematists (and other biologists) attempt to reconstruct this history from various forms of data about contemporary organisms. Phylogeny reconstruction is a crucial step in the understanding of evolution as well as an important tool in biological, pharmaceutical, and medical research. Phylogeny reconstruction from molecular data is very difficult: almost all optimization models give rise to NP-hard (and thus computationally intractable) problems. Yet approximations must be of very high quality in order to avoid outright biological nonsense. Thus many biologists have been willing to run farms of processors for many months in order to analyze just one dataset. High-performance algorithm engineering offers a battery of tools that can reduce, sometimes spectacularly, the running time of existing phylogenetic algorithms, as well as help designers produce better algorithms. We present an overview of algorithm engineering techniques, illustrating them with an application to the "breakpoint analysis" method of Sankoff et al., which resulted in the GRAPPA software suite. GRAPPA demonstrated a speedup in running time by over eight orders of magnitude over the original implementation on a variety of real and simulated datasets. We show how these algorithmic engineering techniques are directly applicable to a large variety of challenging combinatorial problems in computational biology.

Keywords: high-performance computing, computational genomics, phylogeny reconstruction, breakpoint analysis, genome rearrangement, sorting by reversals

1. Algorithm engineering

The term "algorithm engineering" was first used with specificity in 1997, with the organization of the first *Workshop on Algorithm Engineering (WAE 97)*. Since then, this workshop has taken place every summer in Europe and a parallel one started in the US in 1999, the *Workshop on Algorithm Engineering and Experiments (ALENEX99)*, which has taken place every winter, colocated with the *ACM/SIAM Symposium on Discrete Algorithms (SODA)*. Algorithm engineering refers to the process required to transform a pencil-and-paper algorithm into a robust, efficient, well tested, and easily usable implementation. Thus it encompasses a number of topics, from modelling cache behavior to principles of good software engineering; its main focus, however, is experimentation. In that sense, it may be viewed as a recent outgrowth of *Experimental Algorithmics*, which is specifi-

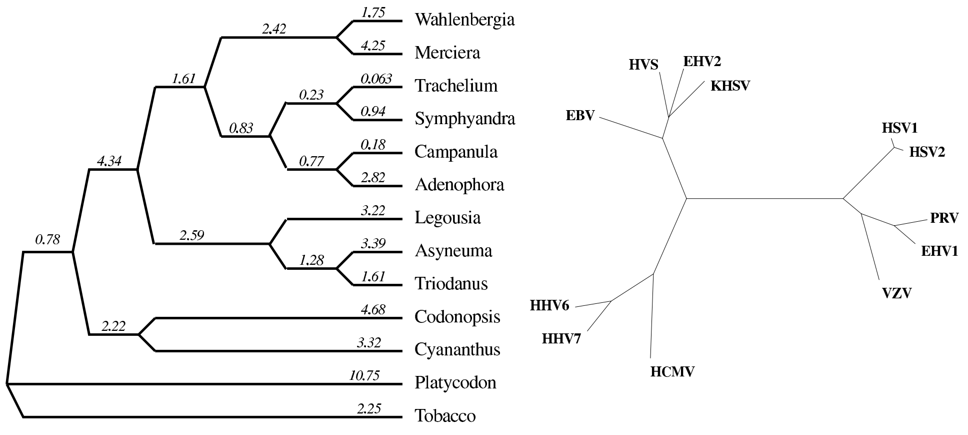

Figure 1. Two phylogenies: some plants of the *Campanulaceae* family (left) and some Herpes viruses affecting humans (right).

of biology and medicine (as well as linguistics). Scientists are of course interested in phylogenies for the usual reasons of scientific curiosity. An understanding of evolutionary mechanisms and relationships is at the heart of modern pharmaceutical research for drug discovery, is helping researchers understand (and defend against) rapidly mutating viruses such as HIV, is the basis for the design of genetically enhanced organisms, etc. In developing such an understanding, the reconstruction of phylogenies is a crucial tool, as it allows one to test new models of evolution.

Bader et al. [4] give many examples of commercial uses of phylogeny reconstruction. They state that simple identification via phylogenetic classification of organisms has, to date, yielded more patent filings than any other use of phylogeny in industry. Other uses have included vaccine development for porcine populations [18] and drawing population boundaries in the continuing study of HIV dynamics [15]; antibacterial and herbicide development through the phylogenetic distribution of biochemical pathways (glyphosate, better known as Roundup^{TM}, Rodeo^{TM}, and Pondmaster^{TM}, was the first herbicide specifically targeted at a pathway, because that pathway is not present in mammals [17]); and prediction of ligands for groups of receptors (especially the G protein coupled receptors or GPCRs), including Neuromedin U, a potent neuropeptide that causes contraction of smooth muscle [42], and a prediction of both the ligand and the pharmacology of a novel GPCR for histamine [46].

3. Computational phylogenetics

Phylogenies have been reconstructed “by hand” for over a century by systematists, using morphological characters and basic principles of genetic inheritance. With the advent of molecular data, however, it has become necessary to develop algorithms to reconstruct phylogenies from the large amount of data made available through

DNA sequencing, amino-acid and protein characterization, gene-expression data, and whole-genome descriptions. Until recently, most of the research focused on the development of methods for phylogeny reconstruction from DNA sequences (which can be regarded as strings on a 4-character alphabet), using a model of evolution based mostly on nucleotide substitution. Because amino-acids, the building blocks of life, are coded by substrings of four nucleotides known as codons, the same methods were naturally extended to sequences of codons (which can be regarded as strings on an alphabet of 20 characters—in spite of the 64 possible codes, only 20 amino-acids are encoded, with many codes representing the same amino-acid). Proteins, which are built from amino-acids, are the natural next level, but are proving difficult to characterize in evolutionary terms.

Recently, another type of data has been made available through the characterization of entire genomes: gene content and gene order data. For some organisms, such as human, mouse, fruit fly, and several plants and lower-order organisms, as well as for a large collection of organelles (mitochondria, the animal cells' "energy factories," and chloroplasts, the plant cells' "photosynthesis factories"), we have a fairly complete description of the entire genome, gene by gene. Because plausible mechanisms of evolution include genome rearrangement, gene duplication, and gene loss, and because evolution at this level (the "genome level") is much slower than evolution driven by mutations in the nucleotide base pairs (the "gene level") and so may enable us to recover deep evolutionary relationships, there has been considerable interest in the phylogeny community in the development of algorithms for reconstructing phylogenies based on gene-order or gene-content data. Appropriate tools for analyzing such data may help resolve some difficult phylogenetic reconstruction problems; indeed, this new source of data has been embraced by many biologists in their phylogenetic work [32, 33, 36].

There is no doubt that, as our understanding of evolution improves, new types of data will be collected and will need to be analyzed in phylogeny reconstruction. Because the applications of phylogeny reconstruction are growing steadily and because pharmaceutical companies find it cost-effective to run large computations rather than hire more bench researchers, the use of high-performance computing will continue to increase. Research groups at pharmaceutical companies are already running phylogenetic computations on small dedicated clusters for well over a year [37]. Computational phylogenetics thus pose a special challenge to algorithm engineering [30], especially in the area of high-performance computing [25].

4. Optimization criteria

To date, almost every model of evolution proposed for modelling phylogenies gives rise to NP-hard optimization problems. Three main lines of work have evolved: more or less *ad hoc* heuristics (a natural consequence of the NP-hardness of the problems) that run quickly, but offer limited or no quality guarantees and may not even have a well defined optimization criterion, such as the popular *neighbor-joining* heuristic [38]; optimization problems based on a *parsimony* criterion, which seeks the phylogeny with the least total amount of change needed to explain modern data (a modern version of Occam's razor); and optimization problems based on a *maximum likelihood* criterion, which seeks the phylogeny that is the most likely (under some suitable statistical model) to have given rise to the modern data. *Ad hoc* heuristics are fast and often rival the optimization methods in terms of accuracy; parsimony-based methods may take exponential time, but, at least for DNA data, can often be run to completion on datasets of moderate size; while methods based on maximum-likelihood are very slow (the point estimation problem alone appears intractable) and so restricted to very small instances, but appear capable of outperforming the others in terms of the quality of solutions. In the case of gene-order data, however, only parsimony criteria have been proposed so far: we do not yet have detailed enough models (or ways to estimate their parameters) for using a maximum-likelihood approach.

A difficult question is how to estimate the *actual* (as opposed to the minimal or *edit*) evolutionary distance: when a large number of evolutionary events have taken place on a single tree edge, the resulting string or genome may be identical to one obtained from the same parent through a much shorter sequence of events. Recovering the edit distance is a well defined computational objective (although doing so may require developing new theories and algorithms [5]), but recovering the true evolutionary distance requires specific evolutionary models (and thus may lack robustness) and can only be done on a statistical basis—usually by computing, then correcting the edit distance. Yet many studies have shown that the use of corrected distances yields significant improvements over the use of edit distances, in terms of both accuracy and speed. In particular, speed improves because corrected distances alter the optimization landscape by sharpening differences and thus improve the efficiency of bounding techniques. Figuring how to obtain a robust estimate of the true evolutionary distance from the edit distance remains an active area of research with gene-order data [29, 43, 44].

5. Our running example: Breakpoint phylogeny

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. Other events, such as insertions, deletions, or duplications, change the number of times and the positions in which a gene occurs.

A natural optimization problem for phylogeny reconstruction from this type of data is to reconstruct the most parsimonious tree, i.e., the evolutionary tree with the minimum number of permitted evolutionary events. For any choice of permitted events, such a problem is computationally very intensive (known or conjectured to be NP-hard); worse, to date, no automated tools exist for solving such problems.

Another approach is first to estimate leaf-to-leaf distances (based upon some metric) between all genomes and then to use a standard distance-based heuristic such as neighbor-joining [38] to construct the tree. Such approaches are quite fast and may prove valuable in reconstructing the underlying tree, but cannot recover the ancestral gene orders.

Blanchette et al. [7] developed an approach, which they called *breakpoint phylogeny*, for the special case in which the genomes all have the same set of genes, and each gene appears once. This special case is of interest to biologists, who hypothesize that gene-order rearrangements (which can only affect gene order, but not gene content) are the main evolutionary mechanism for a range of genomes or chromosomes (chloroplast, mitochondria, human X chromosome, etc.). Simulation studies we conducted suggested that this approach works well for certain datasets (i.e., it obtains trees that are close to the model tree), but that the implementation developed by Sankoff and Blanchette, the BPAnalysis software [40], is too slow to be used on anything other than small datasets with a few genes [11, 12].

6. Breakpoint analysis: Details

When each genome has the same set of genes and each gene appears exactly once, a genome can be described by an ordering (circular or linear) of these genes, each gene given with an orientation that is either positive (g_i) or negative ($-g_i$). Given two genomes G and G' on the same set of genes, a *breakpoint* in G is defined as an ordered pair of genes, (g_i, g_j), such that g_i and g_j appear consecutively in that order in G, but neither (g_i, g_j) nor $(-g_j, -g_i)$ appears consecutively in that order in G'. The breakpoint distance between two genomes is the number of breakpoints between that pair of genomes. The breakpoint score of a tree in which each node is labelled by a signed ordering of genes is then the sum of the breakpoint distances along the edges of the tree.

Given three genomes, we define their *median* to be a fourth genome that minimizes the sum of the breakpoint distances between it and the other three. The *Median Problem for Breakpoints* (MPB) is to construct such a median and is NP-hard [34]. Sankoff and Blanchette developed a reduction from MPB to the Travelling Salesman Problem (TSP), perhaps the most studied of all optimization problems [19]. Their reduction produces an undirected instance of the TSP from the instance of MPB by the standard technique of representing each gene by a pair of cities connected by an edge that must be included in any solution. Figure 2 illustrates the reduction.

BPAnalysis (see Figure 3) is the method developed by Blanchette and Sankoff to solve the breakpoint phylogeny. Within a framework that enumerates all trees, it uses an iterative heuristic to label the internal nodes with signed gene orders. This procedure is computationally very intensive. The outer loop enumerates all $(2n-5)!!$ leaf-labelled trees on n leaves, an exponentially large value (the double factorial is a factorial with a step of 2, so we have $(2n-5)!! = (2n-5) \cdot (2n-7) \cdot \cdots \cdot 3$). The inner loop runs an unknown number of iterations (until convergence), with each iteration solving an instance of the TSP (with a number of cities equal to

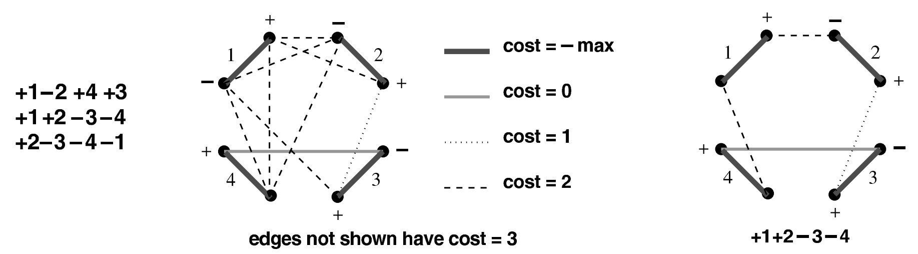

Figure 2. The reduction from MPB to TSP proposed by Sankoff and Blanchette. From left to right: the three permutations, the resulting TSP instance, and an optimal solution with its resulting permutation. The adjacency AB becomes an edge between A and −B; the cost of that edge is the number of genomes that do *not* include the adjacency AB.

twice the number of genes) at each internal node. The computational complexity of the entire algorithm is thus exponential in *each* of the number of genomes and the number of genes, with significant coefficients. The procedure nevertheless remains a heuristic: even though all trees are examined and each MPB problem solved exactly, the tree-labeling phase does not ensure optimality unless the tree has only three leaves.

7. Re-engineering BPAnalysis for speed

7.1. *Profiling*

Algorithmic engineering suggests a refinement cycle in which the behavior of the current implementation is studied in order to identify problem areas which can include excessive resource consumption or poor results. We used extensive profiling and testing throughout our development cycle, which allowed us to identify and eliminate a number of such problems. For instance, converting the MPB into a TSP instance dominates the running time whenever the TSP instances are not too hard to solve. Thus we lavished much attention on that routine, down to the level of hand-unrolling loops to avoid modulo computations and allowing reuse of intermediate expressions; we cut the running time of that routine down by a factor of at least six—and thereby nearly tripled the speed of the overall code. We

```
For each tree in turn do
  Initially label all internal nodes with gene orders
  Repeat
    For each internal node $v$, with neighbors $A$, $B$, and $C$, do
      Solve the MPB on $A$, $B$, $C$ to yield label $m$
      If relabelling $v$ with $m$ improves the score of $T$, then do it
  until no internal node can be relabelled
Return the tree of lowest score
```

Figure 3. Sankoff and Blanchette's high-level algorithm for breakpoint analysis.

lavished equal attention on distance computations and on the computation of the lower bound, with similar results. 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. These steps provided a speed-up by over one order of magnitude on the *Campanulaceae* dataset.

7.2. Cache awareness

The original BPAnalysis is written in C++ and uses a space-intensive full distance matrix, as well as many other data structures. It has a significant memory footprint (over 60MB when running on the *Campanulaceae* dataset) and poor locality (a working set size of about 12MB). Our implementation has a tiny memory footprint (1.8MB on the *Campanulaceae* dataset) and good locality (all of our storage is in arrays preallocated in the main routine and retained and reused throughout the computation), 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 storage is static, 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. While we cannot measure exactly how much we gain from this approach, studies of cache-aware algorithms [1, 13, 21–23, 45] indicate that the gain is likely to be substantial—factors of anywhere from 2 to 40 have been reported. New memory hierarchies show differences in speed between cache and main memory that reach two orders of magnitude (even on uniprocessors).

7.3. Low-level algorithmic changes

Unless the original implementation is poor (which was not the case with BPAnalysis), profiling and cache-aware programming will rarely provide more than two orders of magnitude in speed-up. Further gains can often be obtained by low-level improvement in the algorithmic details. In our phylogenetic software, we made two such improvements.

The basic algorithm scores every single tree, which is clearly very wasteful; we used a simple lower bound, computable in linear time, to enable us to eliminate a tree without scoring it. On the *Campanulaceae* dataset, this bounding eliminates over 95% of the trees without scoring them, resulting in a five-fold speed-up. Subsequence refinement of the bound, at the expense of additional computation, increased the pruning rate to over 99.9% for this dataset, for an additional 50-fold speed-up.

The TSP solver we wrote is at heart the same basic include/exclude search as in BPAnalysis, but we took advantage of the nature of the instances created by the reduction to make the solver much more efficient, resulting in a speed-up by a factor of 5–10. The principal change is to take advantage of the bounded nature of certain quantities—a general principle of fundamental importance in algorithmic engineering. For instance, the only two edge costs that the TSP solver has to consider are 1 and 2: every edge of cost 0 always belongs to any optimal solution, while edges of cost 3 can be treated as an undifferentiated pool of "completion" edges.

These improvements all spring from a careful examination of exactly what information is readily available or easily computable at each stage and from a deliberate effort to make use of all such information.

8. A high-performance implementation

Our resulting implementation, Genome Rearrangement Analysis through Parsimony and other Phylogenetic Algorithms (GRAPPA), incorporates all of the refinements mentioned above, plus others specifically made to enable the code to run efficiently in parallel (see [31] for details). Because the basic algorithm enumerates and (almost) independently scores every tree, it presents obvious ("embarrassing") parallelism: we can have each processor handle a subset of the trees, with communication limited to an (optional) broadcast of improved upper bounds. In order to do so efficiently, we need to impose a linear ordering on the set of all possible trees and devise a generator that can start at an arbitrary point along this ordering. Because the number of trees is so large, an arbitrary tree index would require unbounded-precision integers, considerably slowing down tree generation. Our solution was to design a tree generator that starts with tree index k and generates trees with indices $\{k + cn \mid n \in \mathcal{N}\}$, where k and c are positive integers, all without using unbounded-precision arithmetic. Such a generator allows us to sample tree space (a very useful feature in research) and, more importantly, allows us to use a cluster of c processors, where processor i, $0 \leq i \leq c - 1$, generates and scores trees with indices $\{i + cn \mid n \in \mathcal{N}\}$. Each tree is generated in constant time from its predecessor, a crucial aspect of the code, since every single tree must be generated, even if it is then immediately eliminated through bounding.

We ran GRAPPA on the 512-processor Alliance cluster *Los Lobos* at the University of New Mexico and obtained a 512-fold speed-up. When combined with the 2000-fold speedup obtained through algorithm engineering, our run on the *Campanulaceae* dataset demonstrated a *million-fold* speed-up over the original implementation [2]. Additional improvements in bounding have resulted in an additional 100-fold speed-up for the serial code, so that our parallel implementation run on Los Lobos now runs over *one hundred million* times faster than the original program. Because of the infrequent interprocessor communication in this application, the same performance will be observed on an inexpensive Beowulf cluster.

In addition, we made sure that gains held across a wide variety of platforms and compilers: we tested our code under Linux, FreeBSD, Solaris, and Windows, using compilers from GNU, the Portland group, Intel (beta release), Microsoft, and Sun, and running the resulting code on Pentium- and Sparc-based machines. While the gcc compiler produced marginally faster code than the others, the performance we measured was completely consistent from one platform to the other.

We did not use parallelism to speed up the search for MPB solutions, mostly because our TSP solver runs fast enough in most realistic cases. However, a shared-memory implementation of the TSP solver could take advantage of SMP nodes on a hybrid architecture (cluster of SMPs) and thus provide additional speedup.

9. Impact in computational biology

Computational biology presents numerous complex optimization problems, such as multiple sequence alignment, phylogeny reconstruction, characterization of gene expression, structure prediction, etc. In addition, the very large databases used in computational biology give rise to serious algorithmic engineering problems when designing query algorithms on these databases. While several programs in use in the area (such as BLAST) have already been engineered for performance, most such efforts have been more or less *ad hoc*. The emergence of a discipline of algorithm engineering [28] is bringing us a collection of tools and practices that can be applied to almost any existing algorithm or software package to speed up its execution, often by very significant factors. While we illustrated the approach and its potential results with a specific program in phylogeny reconstruction based on gene order data, we are now in the process of applying the same to a collection of fundamental methods (such as branch-and-bound parsimony or maximum-likelihood estimation) as well as new algorithms.

Of course, even large speed-ups have only limited benefits in theoretical terms when applied to NP-hard optimization problems: even our one-hundred-million-fold speed-up with GRAPPA only enables us to move from 9–10 taxa to 14–15 taxa. Yet the very process of algorithm engineering often uncovers salient characteristics of the algorithm that were overlooked in a less careful analysis and may thus enable us to develop much better algorithms. In our case, while we were implementing the rather complex algorithm of Berman and Hannenhalli for computing the inversion distance between two signed permutations, an algorithm that had not been implemented before, we came to realize that the algorithm could be simplified as well as accelerated, deriving in the process the first true linear-time algorithm for computing these distances [5]. We would not have been tempted to implement this algorithm in the context of the original program, which was already much too slow when using the simpler breakpoint distance. Thus faster experimental tools, even when they prove incapable of scaling to “industrial-size,” nevertheless provide crucial opportunities for exploring and understanding the problem and its solutions.

Thus we see two potential major impacts in computational biology. First, the much faster implementations, when mature enough, can alter the practice of research in biology and medicine. For instance pharmaceutical companies spend large budgets on computing equipment and research personnel to reconstruct phylogenies as a vital tool in drug discovery, yet may still have to wait a year or more for the results of certain computations; reducing the running time of such analyses from 2–3 years down to a day or less would make an enormous difference in the pace, and thus cost of drug discovery and development. Secondly, biologists in research laboratories around the world use software for data analysis, much of it rife with undocumented heuristics for speeding up the code at the expense of optimality, yet still slow for their purposes. Software that runs 3 to 4 orders of magnitude faster and still returns optimal solutions (or approximations with known guarantees) would enable these researchers to test simple scenarios, compare models, develop intuition on small instances, and perhaps form serious conjectures about biological mechanisms.

Acknowledgments

This work was supported in part by NSF grants ACI 00-81404 (Moret and Bader), ACI 00-93039 (Bader), CCR 94-57800 (Warnow), DEB 01-20709 (Moret, Bader, and Warnow), EIA 01-13095 (Moret), EIA 01-13654 (Warnow), EIA 01-21377 (Moret and Bader), and EIA 01-21680 (Warnow), and by the David and Lucile Packard Foundation (Warnow).

References

1. L. Arge, J. Chase, J. Vitter, and R. Wickremesinghe. Efficient sorting using registers and caches. In *Proceedings of the 4th Workshop on Algorithm Engineering (WAE 2000)*, Saarbrücken, Germany, 2000.
2. D. Bader and B. Moret. GRAPPA runs in record time. *HPCwire*, 9(47), 2000.
3. D. Bader, B. Moret, and P. Sanders. High-performance algorithm engineering for parallel computation. In *Experimental Algorithmics*, Lecture Notes in Computer Science. Springer-Verlag, Berlin, 2001.
4. D. Bader, B. Moret, and L. Vawter. Industrial applications of high-performance computing for phylogeny reconstruction. In H. Siegel, ed., *Proceedings of the SPIE Commercial Applications for High-Performance Computing*, vol. 4528, pp. 159–168. Denver, CO, 2001.
5. D. Bader, B. Moret, and M. Yan. A linear-time algorithm for computing inversion distance between signed permutations with an experimental study. *Journal of Computational Biology*, 8:483–491, 2001.
6. M. Bender, E. Demaine, and M. Farach-Colton. Cache-oblivious search trees. In *Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science (FOCS-00)*, pp. 399–409. Redondo Beach, Calif., 2000.
7. M. Blanchette, G. Bourque, and D. Sankoff. Breakpoint phylogenies. In S. Miyano and T. Takagi, eds., *Genome Informatics*, pp. 25–34. University Academy Press, Tokyo, Japan, 1997.
8. B. Cherkassky and A. Goldberg. On implementing the push-relabel method for the maximum flow problem. *Algorithmica*, 19:390–410, 1997.
9. B. Cherkassky, A. Goldberg, P. Martin, J. Setubal, and J. Stolfi. Augment or push: a computational study of bipartite matching and unit-capacity flow algorithms. *ACM Journal of Experimental Algorithmics*, 3(8), 1998. www.jea.acm.org/1998/CherkasskyAugment/.
10. B. Cherkassky, A. Goldberg, and T. Radzik. Shortest paths algorithms: theory and experimental evaluation. *Mathematical Programming*, 73:129–174, 1996.
11. M. Cosner, R. Jansen, B. Moret, L. Raubeson, L.-S. Wang, T. Warnow, and S. Wyman. An empirical comparison of phylogenetic methods on chloroplast gene order data in Campanulaceae. In D. Sankoff and J. Nadeau, eds., *Comparative Genomics: Empirical and Analytical Approaches to Gene Order Dynamics, Map Alignment, and the Evolution of Gene Families*, pp. 99–121. Kluwer Academic Publishers, Dordrecht, Netherlands, 2000.

12. M. Cosner, R. Jansen, B. Moret, L. Raubeson, L.-S. Wang, T. Warnow, and S. Wyman. A new fast heuristic for computing the breakpoint phylogeny and a phylogenetic analysis of a group of highly rearranged chloroplast genomes. In *Proceedings of the 8th International Conference on Intelligent Systems for Molecular Biology (ISMB00)*, pp. 104–115. San Diego, Calif., 2000.
13. N. Eiron, M. Rodeh, and I. Steinwarts. Matrix multiplication: a case study of enhanced data cache utilization. *ACM Journal of Experimental Algorithmics*, 4(3), 1999. www.jea.acm.org/1999/EironMatrix/.
14. M. Frigo, C. Leiserson, H. Prokop, and S. Ramachandran. Cache-oblivious algorithms. In *Proceedings of the 40th Annual Symposium on Foundations of Computer Science (FOCS-99)*, pp. 285–297. New York, 1999.
15. GenProbe. New oligonucleotides corresponding to HIV-1 sequences—used for selective amplification and as hybridisation probes for detection of HIV-1. Patent filing EP-617132-A (priority date), 1993.
16. A. Goldberg and K. Tsioutsiouliklis. Cut tree algorthms: an experimental study. *Journal of Algorithms*, 38:51–83, 2001.
17. E. Grossbard and D. Atkinson, eds., *The Herbicide Glyphosate*. Butterworths, Boston, 1985.
18. P. Halbur, M. Lum, X. Meng, I. Morozov, and P. Paul. New porcine reproductive and respiratory syndrome virus DNA—and proteins encoded by open reading frames of an Iowa strain of the virus; are used in vaccines against PRRSV in pigs. Patent filing WO9606619-A1 (priority date), 1994.
19. D. Johnson and L. McGeoch. The traveling salesman problem: A case study. In E. Aarts and J. Lenstra, eds., *Local Search in Combinatorial Optimization*, pp. 215–310. John Wiley, New York, 1997.
20. D. Jones. An empirical comparison of priority queues and event-set implementations. *Communications of the ACM*, 29:300–311, 1986.
21. R. Ladner, J. Fix, and A. LaMarca. The cache performance of traversals and random accesses. In *Proceedings of the 10th Annual Symposium on Discrete Algorithms (SODA-99)*, pp. 613–622. Baltimore, 1999.
22. A. LaMarca and R. Ladner. The influence of caches on the performance of heaps. *ACM Journal of Experimental Algorithmics*, 1(4), 1996. www.jea.acm.org/1996/LaMarcaInfluence/.
23. A. LaMarca and R. Ladner. The influence of caches on the performance of heaps. In *Proceedings of the 8th Symposium on Discrete Algorithms*, pp. 370–379. New Orleans, LA, 1997.
24. K. Melhorn and S. Näher. *The LEDA Platform of Combinatorial and Geometric Computing*. Cambridge University Press, 1999.
25. B. Moret, D. Bader, and T. Warnow. High-performance algorithm engineering for computational phylogenetics. In *Proceedings of the 2001 International Conference on Computational Science*, vol. 2073–2074. Lecture Notes in Computer Science. San Francisco, Calif., 2001.
26. B. Moret and H. Shapiro. *Algorithms from P to NP, Vol. I: Design and Efficiency*. Menlo Park, Calif., Benjamin-Cummings, 1991.
27. B. Moret and H. Shapiro. An empirical assessment of algorithms for constructing a minimal spanning tree. In *DIMACS Monographs in Discrete Mathematics and Theoretical Computer Science: Computational Support for Discrete Mathematics*, vol. 15, pp. 99–117. American Mathematical Society, 1994.
28. B. Moret and H. Shapiro. Algorithms and experiments: the new (and old) methodology. *Journal of Universal Computer Science*, 7:434–446, 2001.
29. B. Moret, L.-S. Wang, T. Warnow, and S. Wyman. New approaches for reconstructing phylogenies based on gene order. In *Proceedings of the 9th International Conference on Intelligent Systems for Molecular Biology (ISMB 2001)*, pp. S165–S173. In *Bioinformatics* 17, 2001.
30. B. Moret and T. Warnow. Reconstructing optimal phylogenetic trees: a challenge in experimental algorithmics. In *Experimental Algorithmics*, Lecture Notes in Computer Science. Springer-Verlag, Berlin, 2001.
31. B. Moret, S. Wyman, D. Bader, T. Warnow, and M. Yan. A new implementation and detailed study of breakpoint analysis. In *Proceedings of the 6th Pacific Symposium on Biocomputing (PSB2001)*, pp. 583–594. Hawaii, 2001.
32. R. Olmstead and J. Palmer. Chloroplast DNA systematics: a review of methods and data analysis. *American Journal of Botany*, 81:1205–1224, 1994.

33. J. Palmer. Chloroplast and mitochondrial genome evolution in land plants. In R. Herrmann, ed., *Cell Organelles*, pp. 99–133. Springer-Verlag, Berlin, 1992.
34. I. Pe'er and R. Shamir. The median problems for breakpoints are NP-complete. Technical report 71, Electronic Colloquium on Computational Complexity, 1998.
35. N. Rahman and R. Raman. Analysing cache effects in distribution sorting. In *Proceedings of the 3rd Workshop on Algorithm Engineering (WAE99)*. pp. 183–197. London, England, 1999.
36. L. Raubeson and R. Jansen. Chloroplast DNA evidence on the ancient evolutionary split in vascular land plants. *Science*, 255:1697–1699, 1992.
37. K. Rice, M. Donoghue, and R. Olmstead. Analyzing large datasets: rbcl500 revisited. *Systematic Biology* 46:554–562, 1997.
38. N. Saitou and M. Nei. The neighbor-joining method: a new method for reconstruction of phylogenetic trees. *Molecular Biology and Evolution*, 4:406–425, 1987.
39. P. Sanders. Fast priority queues for cached memory. *ACM Journal of Experimental Algorithmics*, 5(7), 2000. www.jea.acm.org/2000/SandersPriority/.
40. D. Sankoff and M. Blanchette. Multiple genome rearrangement and breakpoint phylogeny. *Journal of Computational Biology*, 5:555–570, 1998.
41. J. Stasko and J. Vitter. Pairing heaps: experiments and analysis. *Communications of the ACM*, 30:234–249, 1987.
42. P. Szekeres, A. Muir, L. Spinage, J. Miller, S. Butler, A. Smith, G. Rennie, P. Murdock, L. Fitzgerald, H. Wu, L. McMillan, S. Guerrera, L. Vawter, N. Elshourbagy, J. Mooney, D. Bergsma, S. Wilson, and J. Chambers. Neuromedin U is a potent agonist at the orphan G protein-coupled receptor FM3. *Journal of Biological Chemistry*, 275:20247–20250, 2000.
43. L.-S. Wang. Improving the accuracy of evolutionary distances between genomes. In *Proceedings of the 1st Workshop on Algorithms in Bioinformatics (WABI'01)*, *Lecture Notes in Computer Science*, vol. 2149, pp. 176–190. Århus, Denmark, 2001.
44. L.-S. Wang and T. Warnow. Estimating true evolutionary distances between genomes. In *Proceedings of the 33th Annual Symposium on Theory of Computing (STOC 2001)*, pp. 637–646, 2001.
45. L. Xiao, X. Zhang, and S. Kubricht. Improving memory performance of sorting algorithms. *ACM Journal of Experimental Algorithmics*, 5(3), 2000. www.jea.acm.org/2000/XiaoMemory/.
46. Y. Zhu, D. Michalovich, H. Wu, K. Tan, G. Dytko, I. Mannan, R. Boyce, J. Alston, L. Tierney, X. Li, N. Herrity, L. Vawter, H. Sarau, R. Ames, C. Davenport, J. Hieble, S. Wilson, D. Bergsma, and L. Fitzgerald. Cloning, expression, and pharmacological characterization of a novel human histamine receptor. *Molecular Pharmacology* 59:434–441, 2001.