High-Performance Algorithm Engineering for Computational Phylogenetics

Bernard M.E. Moret[1], David A. Bader[2], and Tandy Warnow[3]

[1] Department of Computer Science, University of New Mexico, Albuquerque, NM 87131, USA, moret@cs.unm.edu, URL www.cs.unm.edu/~moret/
[2] Department of Electrical and Computer Engineering, University of New Mexico, Albuquerque, NM 87131, USA, dbader@eece.unm.edu, URL www.eece.unm.edu/~dbader/
[3] Department of Computer Sciences, University of Texas, Austin, TX 78712, USA, tandy@cs.utexas.edu, URL www.cs.utexas.edu/users/tandy/

Abstract. Phylogeny reconstruction from molecular data poses complex optimization problems: almost all optimization models are NP-hard and thus computationally intractable. 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. 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 million-fold speedup in running time (on a variety of real and simulated datasets) over the original implementation. We show how algorithmic engineering techniques are directly applicable to a large variety of challenging combinatorial problems in computational biology.

1 Background

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 the 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 specifically devoted to the development of

V.N. Alexandrov et al. (Eds.): ICCS 2001, LNCS 2074, pp. 1012–1021, 2001.
© Springer-Verlag Berlin Heidelberg 2001

methods, tools, and practices for assessing and refining algorithms through experimentation. The online *ACM Journal of Experimental Algorithmics (JEA)*, at URL `www.jea.acm.org`, is devoted to this area and also publishes selected best papers from the WAE and ALENEX workshops. Notable efforts in algorithm engineering include the development of LEDA [19], attempts at characterizing the effects of caching on the behavior of implementations [1,11,16,17,18,27,29], ever more efficient implementation of network flow algorithms [7,8,13], and the characterization of the behavior of everyday algorithms and data structures such as priority queues [15,32], shortest paths [6], minimum spanning trees [22], and sorting [21]. More references can be found in [20] as well as by going to the web site for the *ACM Journal of Experimental Algorithmics* at `www.jea.acm.org`.

High-Performance Algorithm Engineering. High-Performance Algorithm Engineering focuses on one of the many facets of algorithm engineering. The high-performance aspect does not immediately imply parallelism; in fact, in any highly parallel task, most of the impact of high-performance algorithm engineering tends to come from refining the serial part of the code. For instance, in the example we will use throughout this paper, the million-fold speed-up was achieved through a combination of a 512-fold speedup due to parallelism (one that will scale to any number of processors) and a 2,000-fold speedup in the serial execution of the code.

All of the tools and techniques developed over the last five years for algorithm engineering are applicable to high-performance algorithm engineering. However, many of these tools will need further refinement. For example, cache-aware programming is a key to performance (particularly with high-performance machines, which have deep memory hierarchies), yet it is not yet well understood, in part through lack of suitable tools (few processor chips have built-in hardware to gather statistics on the behavior of caching, while simulators leave much to be desired) and in part because of complex machine-dependent issues (recent efforts at cache-independent algorithm design [5,12] may offer some new solutions). As another example, profiling a running program offers serious challenges in a serial environment (any profiling tool affects the behavior of what is being observed), but these challenges pale in comparison with those arising in a parallel or distributed environment (for instance, measuring communication bottlenecks may require hardware assistance from the network switches or at least reprogramming them, which is sure to affect their behavior).

Phylogenies. A phylogeny is a reconstruction of the evolutionary history of a collection of organisms; it usually takes the form of an evolutionary tree, in which modern organisms are placed at the leaves and ancestral organisms occupy internal nodes, with the edges of the tree denoting evolutionary relationships. Figure 1 shows two proposed phylogenies, one for several species of the *Campanulaceae* (bluebell flower) family and the other for Herpes viruses that are known to affect humans. Reconstructing phylogenies is a major component of modern research programs in many areas of biology and medicine (as well as linguistics). Scientists are of course interested in phylogenies for the usual reasons of scien-

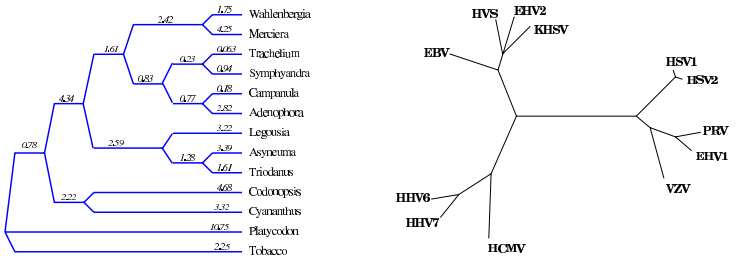

Fig. 1. Two phylogenies: some plants of the *Campanulaceae* family (left) and some Herpes viruses affecting humans (right)

tific 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.

Computational Phylogenetics. Phylogenies have been reconstructed "by hand" for over a century by taxonomists, 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 22 characters—in spite of the 64 possible codes, only 22 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 gene rearrangement, duplication, and 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. Appro-

priate 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.[24,25,28] There is no doubt that, as our understanding of evolution improves, new types of data will be collected and well need to be analyzed in phylogeny reconstruction.

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 no quality guarantees and may not even have a well defined optimization criterion, such as the popular *neighbor-joining* heuristic [31]; 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.

2 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, 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* [31] 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.* [4] 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 inversions (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 [30], is too slow to be used on anything other than small datasets with a few genes [9,10].

3 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 [26]. Sankoff and Blanchette developed a reduction from MPB to the Travelling Salesman Problem (TSP), perhaps the most studied of all optimization problems [14]. Their reduction produces an undirected instance of the TSP from the directed 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.

`BPAnalysis` (see Fig. 2) is the method developed by Blanchette and Sankoff

```
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
```

Fig. 2. `BPAnalysis`

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.[1] 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 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.

4 Re-engineering BPAnalysis for Speed

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 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 one order of magnitude on the *Campanulaceae* dataset.

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

[1] The double factorial is a factorial with a step of 2, so we have $(2n-5)!! = (2n-5) \cdot (2n-7) \cdot \ldots \cdot 3$

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 [1,11,16,17,18,33] 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 exceed two orders of magnitude.

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. 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. 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.

5 A High-Performance Implementation

Our resulting implementation, *GRAPPA*,[2] incorporates all of the refinements mentioned above, plus others specifically made to enable the code to run efficiently in parallel (see [23] for details). Because the basic algorithm enumerates and independently scores every tree, it presents obvious parallelism: we can have each processor handle a subset of the trees. 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 regular 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 $\{i + cn \mid n \in \mathcal{N}\}$. 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

[2] Genome Rearrangement Analysis through Parsimony and other Phylogenetic Algorithms

algorithm engineering, our run on the *Campanulaceae* dataset demonstrated a *million-fold* speed-up over the original implementation [2].

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.

6 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, see `www.ncbi.nlm.nih.gov/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 [20] 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 million-fold speed-up with *GRAPPA* only enables us to move from about 10 taxa to about 13 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 [3]. 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-sized" problems, 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 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, even when it remains impractical for real-world problems, would nevertheless enable these researchers to test simpler scenarios, compare models, develop intuition on small instances, and perhaps even form serious conjectures about biological mechanisms.

Acknowledgments

This work was supported in part by NSF ITR 00-81404 (Moret and Bader), NSF 94-57800 (Warnow), and the David and Lucile Packard Foundation (Warnow).

References

1. Arge, L., Chase, J., Vitter, J.S., & Wickremesinghe, R., "Efficient sorting using registers and caches," *Proc. 4th Workshop Alg. Eng. WAE 2000*, to appear in LNCS series, Springer Verlag (2000).
2. Bader, D.A., & Moret, B.M.E., "GRAPPA runs in record time," *HPC Wire*, **9**, 47 (Nov. 23), 2000.
3. Bader, D.A., Moret, B.M.E., & Yan, M., "A fast linear-time algorithm for inversion distance with an experimental comparison," Dept. of Comput. Sci. TR 2000-42, U. of New Mexico.
4. Blanchette, M., Bourque, G., & Sankoff, D., "Breakpoint phylogenies," in *Genome Informatics 1997*, Miyano, S., & Takagi, T., eds., Univ. Academy Press, Tokyo, 25–34.
5. Bender, M.A., Demaine, E., and Farach-Colton, M., "Cache-oblivious search trees," *Proc. 41st Ann. IEEE Symp. Foundations Comput. Sci. FOCS-00*, IEEE Press (2000), 399–409.
6. Cherkassky, B.V., Goldberg, A.V., & Radzik, T., "Shortest paths algorithms: theory and experimental evaluation," *Math. Progr.* **73** (1996), 129–174.
7. Cherkassky, B.V., & Goldberg, A.V., "On implementing the push-relabel method for the maximum flow problem," *Algorithmica* **19** (1997), 390–410.
8. Cherkassky, B.V., Goldberg, A.V., Martin, P, Setubal, J.C., & Stolfi, J., "Augment or push: a computational study of bipartite matching and unit-capacity flow algorithms," *ACM J. Exp. Algorithmics* **3**, 8 (1998), `www.jea.acm.org/1998/CherkasskyAugment/`.
9. Cosner, M.E., Jansen, R.K., Moret, B.M.E., Raubeson, L.A., Wang, L.-S., Warnow, T., & Wyman, S., "A new fast heuristic for computing the breakpoint phylogeny and experimental phylogenetic analyses of real and synthetic data," *Proc. 8th Int'l Conf. Intelligent Systems Mol. Biol. ISMB-2000*, San Diego (2000).
10. Cosner, M.E., Jansen, R.K., Moret, B.M.E., Raubeson, L.A., Wang, L.S., Warnow, T., & Wyman, S., "An empirical comparison of phylogenetic methods on chloroplast gene order data in Campanulaceae," *Proc. Gene Order Dynamics, Comparative Maps, and Multigene Families DCAF-2000*, Montreal (2000).

11. Eiron, N., Rodeh, M., & Stewarts, I., "Matrix multiplication: a case study of enhanced data cache utilization," *ACM J. Exp. Algorithmics* **4**, 3 (1999), `www.jea.acm.org/1999/EironMatrix/`.
12. Frigo, M., Leiserson, C.E., Prokop, H., & Ramachandran, S., "Cache-oblivious algorithms," *Proc. 40th Ann. Symp. Foundations Comput. Sci. FOCS-99*, IEEE Press (1999), 285–297.
13. Goldberg, A.V., & Tsioutsiouliklis, K., "Cut tree algorthms: an experimental study," *J. Algs.* **38**, 1 (2001), 51–83.
14. Johnson, D.S., & McGeoch, L.A., "The traveling salesman problem: a case study," in *Local Search in Combinatorial Optimization*, E. Aarts & J.K. Lenstra, eds., John Wiley, New York (1997), 215–310.
15. Jones, D.W., "An empirical comparison of priority queues and event-set implementations," *Commun. ACM* **29** (1986), 300–311.
16. Ladner, R., Fix, J.D., & LaMarca, A., "The cache performance of traversals and random accesses," *Proc. 10th ACM/SIAM Symp. Discrete Algs. SODA99*, SIAM Press (1999), 613–622.
17. LaMarca, A., & Ladner, R., "The influence of caches on the performance of heaps," *ACM J. Exp. Algorithmics* **1**, 4 (1996), `www.jea.acm.org/1996/LaMarcaInfluence`.
18. LaMarca, A., & Ladner, R., "The influence of caches on the performance of sorting," *Proc. 8th ACM/SIAM Symp. Discrete Algs. SODA97*, SIAM Press (1997), 370–379.
19. Melhorn, K., & Näher, S. *The LEDA Platform of Combinatorial and Geometric Computing*. Cambridge U. Press, 1999.
20. Moret, B.M.E., "Towards a discipline of experimental algorithmics," *Proc. 5th DIMACS Challenge* (to appear), available at `www.cs.unm.edu/~moret/dimacs.ps`.
21. Moret, B.M.E., & Shapiro, H.D. *Algorithms from P to NP, Vol. I: Design and Efficiency*. Benjamin-Cummings, Menlo Park, CA, 1991.
22. Moret, B.M.E., & Shapiro, H.D., "An empirical assessment of algorithms for constructing a minimal spanning tree," in *Computational Support for Discrete Mathematics*, N. Dean & G. Shannon, eds., *DIMACS Monographs in Discrete Math. & Theor. Comput. Sci.* **15** (1994), 99–117.
23. Moret, B.M.E., Wyman, S., Bader, D.A., Warnow, T., & Yan, M., "A detailed study of breakpoint analysis," *Proc. 6th Pacific Symp. Biocomputing PSB 2001*, Hawaii, World Scientific Pub. (2001), 583–594.
24. Olmstead, R.G., & Palmer, J.D., "Chloroplast DNA systematics: a review of methods and data analysis," *Amer. J. Bot.* **81** (1994), 1205–1224.
25. Palmer, J.D., "Chloroplast and mitochondrial genome evolution in land plants," in *Cell Organelles*, Herrmann, R., ed., Springer Verlag (1992), 99–133.
26. Pe'er, I., & Shamir, R., "The median problems for breakpoints are NP-complete," *Elec. Colloq. Comput. Complexity*, Report 71, 1998.
27. Rahman, N., & Raman, R., "Analysing cache effects in distribution sorting," *Proc. 3rd Workshop Alg. Eng. WAE99*, in LNCS **1668**, Springer Verlag (1999), 183–197.
28. Raubeson, L.A., & Jansen, R.K., "Chloroplast DNA evidence on the ancient evolutionary split in vascular land plants," *Science* **255** (1992), 1697–1699.
29. Sanders, P., "Fast priority queues for cached memory," *ACM J. Exp. Algorithmics* **5**, 7 (2000), `www.jea.acm.org/2000/SandersPriority/`.
30. Sankoff, D., & Blanchette, M., "Multiple genome rearrangement and breakpoint phylogeny," *J. Comput. Biol.* **5** (1998), 555–570.
31. Saitou, N., & Nei, M., "The neighbor-joining method: A new method for reconstructing phylogenetic trees," *Mol. Biol. & Evol.* **4** (1987), 406–425.
32. Stasko, J.T., & Vitter, J.S., "Pairing heaps: experiments and analysis," *Commun. ACM* **30** (1987), 234–249.
33. Xiao, L., Zhang, X., & Kubricht, S.A., "Improving memory performance of sorting algorithms," *ACM J. Exp. Algorithmics* **5**, 3 (2000), `www.jea.acm.org/2000/XiaoMemory/`.