By Catherine C. McGeoch

EXPERIMENTAL ALGORITHMICS

Theoretical questions and motivations, combined with empirical research methods, produce insights into algorithm and program performance.

Research on algorithms embraces two fundamental goals: algorithm design and algorithm analysis. Algorithm analysis is concerned with predicting how well a given algorithm will perform in a given scenario under given conditions and assumptions. Algorithm design is concerned with building better algorithms: in this context, "better" usually means faster but can also mean returning higher-quality (preferably optimal) solutions to a given problem. Algorithms that are both efficient and optimal are surprisingly difficult to find in a large number of application areas (such as sequencing the human genome, building better Internet services, scheduling airlines, and

If a problem is NP-complete, finding an algorithm for it that is always both fast and optimal, or showing that no such algorithm can exist, WOULD BE ANALOGOUS TO FINDING A CURE FOR THE COMMON COLD: fame and fortune would be yours.

routing fleets of delivery trucks). Because time is money, the quest for faster algorithms and the fundamental trade-off between "speed" and "quality" in applications important to industry, government, and science drives modern research in algorithm design.

Traditionally, algorithm design and analysis have been conducted using the tools of abstraction and mathematical proof. An algorithm is typically described in pseudocode, intended to run only on a generic and abstract model of a modern computer. The first goal of analysis is to find an asymptotic upper bound on algorithm performance using this model (such as *"Quicksort is $O(n^2)$ in the worst case"*). This kind of bound guarantees that no matter what programming language or platform is used, Quicksort will perform no more than cn^2 generic machine instructions (for some unspecified constant c) to sort a list of n numbers.

The abstract approach to design and analysis allows us to discover universal truths about fundamental algorithmic properties. The result is insight into what makes algorithms more and less efficient, which can lead to the discovery of even better algorithms. When adopted by the software engineer, better algorithms become faster programs that produce higher-quality results. However, it is quite difficult to translate an asymptotic time bound into an accurate time prediction. One difficulty is the dependence on "worst case" assumptions. For example, Quicksort's performance depends significantly on the input order of the n numbers to be sorted. On an input of 1,000 numbers, Quicksort might run hundreds of times faster than the worst-case prediction. Furthermore, the gap increases with n: on an input of one million numbers, Quicksort can run thousands of times faster than the worst-case prediction. A second difficulty arises when we try to translate "cn generic machine instructions" into microseconds; unlike an algorithm, a real program encounters compiler optimization, pipelining, caching, and other platform-specific phenomena that dramatically reduce the predictive power of simple instruction counts.

Where theoretical analysis may fall short, computational experiments have long been used to measure program performance under real-world conditions. The classic "field experiment," which might involve CPU runtime measurements taken on specific platforms using input instances from real applications, can give much more precise information about program performance in practice.

In addition, in the past few decades, experimental methodology in this context has evolved into something resembling a classic "laboratory experiment" for analyzing algorithms. This approach emphasizes highly controlled parameters, carefully placed data-collection probes, and sophisticated data analysis. Rather than simply measure performance, the goal of the experiment is to generalize away from context-specific measurements and build insight into fundamental structures and properties.

Indeed, since experiments can be used to measure any aspect of algorithm/program performance, not just CPU times, this approach can stimulate new forms of interplay between theory and practice. The term "experimental algorithmics" describes this new approach to algorithm design and analysis, combining theoretical questions and motivations with empirical research methods. The related term "algorithm engineering" is almost synonymous but emphasizes design over analysis.

Three Examples

Here, I describe three example areas in which experimental algorithmics has produced new results and significant progress in algorithm research. They also illustrate the variety of research questions that can be investigated through experimental methods. (For several more examples and methods of experimental algorithmics, see the references in the sidebar "Resources.")

Memory-efficient models of computation. The traditional theoretical approach to analysis involves counting basic operations performed on an abstract computer. These operations are generic abstractions of CPU instructions, each assumed to have some unit cost. However, on modern architectures, instructions have varying costs; in particular, a memory access can

be several orders of magnitude more time consuming than any single machine instruction, depending on where the operands are located. New abstract models are needed to describe these new types of costs. Recent progress in the development of memory access models has been greatly stimulated and guided by experimental research.

For example, experiments have guided the development of new analytical models of computation to describe the cache performance of several sorting algorithms [4]. These new analyses produced more accurate predictions of program running times and turned some conventional wisdom about best programming practice on its head. For example, Quicksort's cache performance depends closely on how the partitioning code is structured and at what time during algorithm execution small sublists are processed. It turns out that the well-established design goal of minimizing instruction counts may in fact be counterproductive if it produces poor cache behavior.

Subsequently, many researchers have combined experiments and theory in this way to analyze memory access patterns and engineer better algorithms and programs for a variety of problem domains, including search, dynamic tree and trie structures, matrix operations, and permutation problems. These new algorithms are "cache-efficient," which can mean "cache-aware," able to adjust to and exploit known cache properties, or "cache-oblivious," robust with respect to variations in caches. These ideas have been exploited to engineer sorting programs that are very fast indeed [1].

Phase transitions in combinatorial problems. Another area that has seen much recent growth is the study of "phase transition behaviors" in algorithms for a large variety of NP-complete problems. If a problem is NP-complete, finding an algorithm for it that is always both fast and optimal, or showing that no such algorithm can exist, would be analogous to finding a cure for the common cold: fame and fortune would be yours. Like the Quicksort example, a given algorithm for an NP-complete problem may run fast or slow depending on properties of individual inputs, but these properties are poorly understood. Furthermore, the gap between worst case and typical case for these problems is enormous. In some practical situations, we cannot reliably predict algorithm performance to within years.

The phase-transition phenomenon was first observed in the late 1980s/early 1990s by several researchers; subsequently, both experimental and theoretical tools were used to develop new ways to classify input difficulty in these domains.

A single example problem serves to illustrate the larger questions. In the K-satisfiability problem, we are given a Boolean formula B in conjunctive normal form, containing n variables, m clauses, and k variables per clause. Here is an example formula with five variables, four clauses, and three variables per clause:

$$B = (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_4}) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_5}) \wedge (x_3 \vee x_4 \vee x_5).$$

The problem is to determine whether truth values can be assigned to the variables in such a way that B evaluates to True. If it does, the instance B is "satisfiable," and if not, it is "unsatisfiable." In this case, B can be satisfied by setting x_1, x_3 to True, and x_2, x_4, x_5 to False. (When k is greater than or equal to 3, this problem is NP-complete; when k is less than 3 an efficient algorithm is known.)

K-satisfiability is a universal problem in the sense that an efficient algorithm for it can be translated by a simple process into an efficient algorithm for any NP-complete problem. To date, the best algorithm known for determining K-satisfiability is simply to check all 2^n possible truth assignments. This is extremely inefficient. For example, assuming one check per microsecond, checking a formula with a mere 50 variables would require about 435 years of computation.

Now consider generating problem instances B_{nmk} uniformly at random, for given parameter values n, m, and k. What is the probability that B_{nmk} is satisfiable? What is the probability that a solution for B_{nmk} can be found efficiently? As it turns out, these two questions are closely related. Let x be the ratio m/n. Experiments performed by a number of researchers demonstrate convincingly that for each k greater than or equal to 2 there is a transition value x_k such that the following property holds: when x is well below x_k, the probability that B_{nmk} is satisfiable is near 1; when x is well above the threshold, the probability is near 0; and that probability makes a sharp change—a phase transition—when x is near x_k.

Furthermore, the parameter x can be used to predict the difficulty of solving B_{nmk}. Heuristic algorithms for satisfiability have little trouble solving instances that are far from the threshold value x_k, and

[1] In this formula, $\wedge$ represents the logical AND function, $\vee$ represents the logical OR function, and $\overline{x_1}$ represents the logical NOT function applied to variable x_1.

The well-established design goal of minimizing instruction counts MAY IN FACT BE COUNTERPRODUCTIVE if it produces poor cache behavior.

great difficulty solving instances that are near it. Intuitively, it is easy to find a satisfying truth assignment when x is low, because there are many more variables than clauses and therefore few conflicts among the variables; many satisfying truth assignments exist. Similarly, it is easy to demonstrate unsatisfiability when x is large, because the clauses impose too many conflicts on the variables. It is remarkable, however, that the difficulty of solving a given random instance can be predicted so precisely based on just the simple parameter x.

Put another way, random satisfiability instances that tend to be difficult to solve may be found in a fairly localized region in the space of random instances—specifically the region near $m/n = x_k$. Much experimental effort has focused on locating the threshold values x_k for $k \geq 3$ and quantifying the difficulties of specific algorithms solving these parameterized instances.

In 1991, the extensive experimental results in [2] demonstrated that this easy-hard-easy phenomenon can be observed in a variety of NP-complete problems, and for each problem a simple input parameter with threshold behavior (such as x for K-satisfiability) can be found. The authors conjectured that this type of phase transition for random instances may in fact be a property of all NP-complete problems. Subsequently, a large body of experimental research has emerged to guide and inspire new theoretical results, and conversely, new theorems have been used to guide more experiments.

The existence of a simple parameter (such as x), a threshold value, and an easy-hard-easy transition in problem difficulty that depends on the parameter has been observed for many problems besides satisfiability, including Graph Coloring, Number Partitioning, and Traveling Salesman. This vigorous interaction between experimental and theoretical analysis has led to the development of new classification schemes for "hard instances" for combinatorial problems, with deep implications for both theory and practice.

Algorithm engineering. The third example concerns contributions to research in algorithm engineering, which emphasizes the design aspect of algorithmic research. The aim is to transform abstract algorithms into well-implemented programs with an emphasis on efficiency and generality.

The DIMACS Implementation Challenge has stimulated much research in algorithm engineering. Since 1990, the Center for Discrete Mathematics and Theoretical Computer Science at Rutgers University in New Jersey has sponsored a quasi-annual Implementation Challenge that identifies a small set of problem areas and invites researchers to carry out design and analysis projects for algorithms that address the problems.

Participants (in research groups worldwide) adopt a common input format and common sets of input instances to test their implementations. Each Challenge involves a year-long cooperative research effort to find algorithms and implementations that are most efficient in general, most robust with regard to particular input categories, and most highly tuned for practical use. In addition to the testbed experimental results, participants produce new analyses, data structures, and implementation strategies for these algorithms.

Most Challenges culminate with a workshop (dimacs.rutgers.edu/Challenges) where participants present their results and conclusions. This novel cooperative approach to algorithm research has generated much more new information—and software—than could ever by produced by researchers working in isolation. Two recent challenges are discussed here.

The Ninth Challenge, 2005–2006, was organized by Camil Demetrescu of the University of Rome "La Sapienza," Andrew Goldberg of Microsoft Research, and David S. Johnson of AT&T Labs-Research to address variations in shortest-path problems. At the workshop held in November 2006, participants presented practical and efficient algorithms for solving K-shortest paths, techniques for exploiting precomputation in memory-restricted applications, and efficient strategies for dynamic query-based problems. These types of algorithms are used in, for example,

RESOURCES

Here's a primer on experimental algorithmics, including articles with commentary on methodology and meetings and publications with examples of high-quality research:

- Demetrescu, C. and Italiano, G. What do we learn from experimental algorithmics? In *Proceedings of the Mathematical Foundations of Computer Science 25th International Symposium, LNCS 1893* (Bratislava, Slovak Republic, Aug. 28–Sept. 1). Springe LNCS Computer Science Editorial, Heidelberg, Germany, 2000, 36–51.
- Johnson, D. A theoretician's guide to the experimental analysis of algorithms. In *Data Structures, Near Neighbor Searches, and Methodology: Fifth and Sixth DIMACS Implementation Challenges DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Vol. 59,* M. Goldwasser, D. Johnson, and C. McGeoch, Eds. American Mathematical Society, Providence, RI, 2002, 215–250.
- McGeoch, C. Toward an experimental method for algorithm simulation. *INFORMS Journal on Computing 8,* 1 (Winter 1995), 1–15.
- The DIMACS Challenges, since 1990, have been sponsored by the Center for Discrete Mathematics and Theoretical Computer Science. Proceedings are published by the American Mathematical Society (dimacs.rutgers.edu/Volumes/index.html) as part of the DIMACS Series in Discrete Mathematics and Theoretical Computer Science; dimacs.rutgers.edu/Challenges.
- Three annual workshops cover research in experimental algorithmics and algorithm engineering: The Workshop on Algorithm Engineering (1997–2001) became (in 2002) the European Symposium on Algorithms, Engineering, and Applications Track. The Workshop on Algorithm Engineering and Experiments has been held since 1999. And the Workshop on Efficient and Experimental Algorithms has been held since 2001.
- The *Journal of Experimental Algorithms* is published by ACM; www.jea.acm.org.

mapping and direction-finding systems for automobiles and other consumer applications. Comparing them revealed much new information about effective design strategies for the problems.

The Eighth Challenge, 2000–2001, was organized by David S. Johnson, Lyle McGeoch of Amherst College, Fred Glover of the University of Colorado, and Cesar Rego of the University of Mississippi to evaluate a collection of algorithms for the Traveling Salesman problem. Participants contributed experimental results involving more than 150 implementations of state-of-the art algorithms. A companion Web site (www.research.att.com/~dsj/chtsp/) allows the visitor to perform direct comparisons (of both time and tour quality) using a large collection of input instances. Based on this experimental data, the organizers [3] presented a detailed discussion of how to select the best algorithms for solving real-world applications of the Traveling Salesman problem. These applications arise in, for example, delivery truck routing and fleet scheduling.

Conclusion

Experimental analysis of algorithms is an evolving discipline. Together with a growing body of experimental results on algorithm performance, a collection of articles addressing methodological issues has also emerged. Since the experimental subject and the research questions are somewhat unusual compared to other problem domains (for example, doing experiments on Dijkstra's algorithm is very different from doing experiments on, say, a microprocessor architecture) much more work is needed to identify the statistical and data analysis tools most appropriate to these types of problems. C

References

1. Brodal, G., Fagerberg, R., and Vinther, K. Engineering a cache-oblivious sorting algorithm. *Journal of Experimental Algorithmics 12* (2007).
2. Cheeseman, P., Kanefsky, B., and Taylor, W. Where the really hard problems are. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence,* J. Mylopoulos and R. Reiter, Eds. (Sydney, Australia, Aug. 24–30). Morgan Kaufman, San Mateo, CA, 1991, 331–337.
3. Johnson, D. and McGeoch, L. Experimental analysis of heuristics for the STSP. In *The Traveling Salesman Problem and Its Variations,* G. Gutin and A. Putin, Eds. Kluwer Academic Publishers, Boston, 2002, 369–443.
4. LaMarca, A. and Ladner, R. The influence of caches on the performance of sorting. *Journal of Algorithms 31,* 1 (Apr. 1999), 66–104.

Catherine C. McGeoch (ccm@cs.amherst.edu) is a professor of computer science and chair of the Department of Mathematics and Computer Science at Amherst College, Amherst, MA.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

© 2007 ACM 0001-0782/07/1100 $5.00