From Algorithm to Program to Software Library

Kurt Mehlhorn
http://www.mpi-sb.mpg.de/~mehlhorn

Max-Planck-Institut für Informatik, 66123 Saarbrücken, Germany

The area of algorithmics has made significant progress over the past ten years.

- *New concepts were invented and flourished*: probabilistically checkable proof, non-approximability, approximation algorithms, on-line algorithms, to name some of them. The books [Hoc96,ACG$^+$99,BEY98] give accounts.
- *Many improved algorithms were found*, for example, the $O(nm)$ barrier for maximum network flow, which resisted for more than three decades, was finally broken [GR98].
- *Programs became first class citizens.* Ten years ago, algorithmicists considered programs only as an afterthought. Our work was done, when algorithms were formulated and analyzed. Today, algorithmicists also think about implementations, develop software libraries, and experiment with algorithms. Many algorithms are now available in software libraries. Think of CPLEX, MAPLE, MATHEMATICA, CGAL, LEDA, ILOG-solver. I was involved in the design of LEDA [LED] and CGAL [CGA] and want to share with you some of the experiences made. A much more detailed account can be found in [MN99].

I concentrate on four aspects: (1) algorithms are usually designed for a machine that can compute with real numbers (in the sense of mathematics). That's a problem in general, it is a big problem for geometric algorithms. (2) Programmers are not perfect, they make mistakes. (3) RAMs, our model for theoretical analysis, differ from real machines in many ways. (4) Libraries must be simple to use.

Implementing Geometric Algorithms: Geometric algorithms rely on geometric predicates for their flow of control. Examples of geometric

predicates are the orientation predicate for three points or the incircle predicate for four points. The evaluation of geometric predicates usually amounts to computing the sign of an arithmetic expression in the coordinates of the points involved. For example, three points p, q, r form a left-turn, lie on a straight-line, form a right-turn iff the sign[1] of $q_x r_y - q_y r_x - p_x r_y + p_y r_x + p_x q_y - p_y q_x$ is positive, zero, or negative, respectively. When evaluated with floating point arithmetic (or more generally, any arithmetic that may incur roundoff error), the wrong sign may be obtained and hence the geometric predicate may be decided incorrectly. *What is so bad about this? After all, numerical analysis has dealt with roundoff errors for decades.* The problem is that decisions in geometric algorithms are not independent. The figures on the next page illustrate the problem.

The figures illustrate an update step in an incremental Voronoi diagram computation; for the argument to follow, the reader needs only a superficial knowledge of Voronoi diagrams. A Voronoi diagram is a graph like structure embedded into the plane. It can be constructed incrementally. In an update step, part of the current diagram is removed; in the figure on the left the current diagram is drawn solid and the part to be removed is enclosed in a dashed ellipse. It is a theorem that the part to be removed forms a subtree of the current graph structure. The update algorithm uses geometric predicates to determine the part of the diagram to be removed. In particular, it is determined, which vertices of the current graph structure are to be removed. Assume now that (inexact) evaluation of geometric predicates tells us to remove precisely the vertices indicated in the right part of the figure, contradicting the theorem that the subgraph to be removed is a tree and hence connected. Most

[1] We use p_x and p_y to denote the Cartesian coordinates of point p and analogously for the other points.

implementations of the update algorithm will go astray when faced with this situation.

There are two approaches to the problem: one can either reformulate the algorithms [SOI90,Mil88,FM91,GSS93] or one can resort to exact arithmetic [KLN91,FvW96,YD95,MN94]. We use the latter approach in LEDA and CGAL. The exact arithmetic approach proved to be very successful for computations staying within the rational numbers. Efficient filter technology and effective special purpose algorithms for particular predicates were developed [FvW96,MN94,BBP98,BFS98,Cla92,ABDP97,BEPP97].

The situation is less clear as soon as non-rational algebraic numbers are required. Algebraic numbers are, for example, needed to handle curves and surfaces, and for problems involving distances. LEDA's sign computations for algebraic numbers are based on the following lemma [BFMS00,MS00b]. Similar theorems were shown in [KLPY99,LY00].

Let E be an expression with integer operands and operators $+$, $-$, $*$ and $\sqrt{\ }$. Let $u(E)$ be the value of E after replacing $-$ by $+$ and let $k(E)$ be number of distinct square roots in E. Then

$$E = 0 \qquad \textit{or} \qquad E \,\% \;\geq B = \frac{1}{u(E)^{2^{k(E)}}}$$

Sign computation for an expression E amounts to computing an approximation $\tilde{E}$ of E with $E - \tilde{E}\,\% \; < B/2$: if $\tilde{E}\,\% \; < B/2$ then $E = 0$ and if $\tilde{E}\,\% \; \geq B/2$ then $sign(E) = sign(\tilde{E})$. Numerical analysis techniques and arbitrary precision floating point arithmetic are used to obtain an approximation of the required quality. An interesting alternative approach stays within the integers [BMS94,DFMT00].

A challenge for the next ten years is to develop methods that can efficiently handle curves and surfaces.

Program Result Checking: In LEDA we had a planarity test starting in 91. A planarity test takes a graph and outputs "Yes", if the graph is planar, and outputs "No", otherwise. Several years later, we were sent a graph that our implementation declared non-planar. The planarity of the graph was witnessed by a planar drawing. When Stefan

Näher and I analyzed the situation, we concluded that we should try to make our programs "convincing". They should not only output the desired result, but also a "proof", which makes it easy to check that the program worked correctly on a particular input. In the case of the planarity test, this meant to output a so-called combinatorial embedding in the case of a planar graph and a Kuratowski subgraph in the case of a non-planar graph. We also wrote programs that check the proofs. The proof checkers are typically so simple that their correctness is "obvious". We refer the reader to [WB97,MN99] for a more detailed discussion of program result checking.

A challenge for the coming years is to combine program result checking and program proving.

Consider the following example[2]: A sorting routine working on a set S

(a) must not change S and
(b) must produce a sorted output.

The first property is hard to check (provably as hard as sorting), but usually trivial to prove, e.g., if the sorting algorithm uses a *swap*-subroutine to exchange items. The second property is easy to check by a linear scan over the output, but hard to prove (if the sorting algorithm is complex).

RAMs and Real Machines: In the machine model used for the theoretical analysis of algorithms, instructions and memory accesses are unit cost. Real machines are different, they have pipelines, memory hierarchies, ... ; the distance between the model and real machines will further increase and hence the predictive value of theoretical analysis will further decrease. The area of external memory algorithms [Vit98] addresses the issue for the high-end of the memory hierarchy. A study of the algorithmic aspects of cache memories is emerging [LL97,FLPR99,MS00a].

The challenge for the coming years is to make more aspects of real machines amenable to theoretical analysis. Another challenge is to combine theoretical analysis and experimental evaluation of algorithms.

[2] The author does not recall where he learned about this example.

Software Engineering: LEDA is by now widely used. This is certainly partly due to its broad scope and the fact that the algorithms and data structures in LEDA are correct[3] and efficient. The main reason for the wide use of LEDA is probably its ease of use[4]. The architecture of LEDA boils down to small number of concepts: items, iterators, graphs, ... We refer the reader to the discussion in [MN99, Chapter II] for details. Ease of use and flexibility or generality are goals that are frequently in conflict. LEDA is usually siding with the ease of use side, CGAL [CGA] and STL [MS96] are siding with the generality side.

A challenge is to combine the ease of use of LEDA with the generality of STL and CGAL.

Conclusion: An answer to the challenges formulated in the text above would allow us to build libraries of correct, efficient, predictable, certified, flexible implementations of data structures, combinatorial and geometric algorithms.

Remark: The slides for the talk given at the Dagstuhl 10th Anniversary Meeting are available at the author's web page.

References

[ABDP97] F. Avnaim, J.-D. Boissonnat, O. Devillers, and F.P. Preparata. Evaluating signs of determinants with floating point arithmetic. *Algorithmica*, 17(2):111–132, 1997.

[ACG+99] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and Approximation: Combinatorial Optimization Problems and their Approximability Properties.* Springer Verlag, 1999.

[BBP98] H. Brönnimann, C. Burnikel, and S. Pion. Internal arithmetic yields efficient arithmetic filters for computational geometry. In *Proceedings of the 14th Annual Symposium on Computational Geometry (SCG'98)*, pages 165–174, 1998.

[BEPP97] H. Brönnimann, I. Emiris, V. Pan, and S. Pion. Computing exact geometric predicates using modular arithmetic with single precision. In *Proceedings of 13th Annual ACM Symposium on Computational Geometry (SCG'97)*, pages 174–182, 1997.

[BEY98] A. Borodin and R. El-Yaniv. *ONLINE COMPUTATION AND COMPETITIVE ANALYSIS.* Cambridge University Press, 1998.

[3] We have not had a bug report for any of the major algorithms in months.

[4] Correctness plays a major role here; we all know that there is little fun in using buggy software.

[BFMS00] C. Burnikel, R. Fleischer, K. Mehlhorn, and S. Schirra. A strong and easily computable separation bound for arithmetic expressions involving radicals. *Algorithmica*, 27:87–99, 2000.

[BFS98] C. Burnikel, S. Funke, and M. Seel. Exact arithmetic using cascaded computation. In *Proceedings of the 14th Annual Symposium on Computational Geometry (SCG'98)*, pages 175–183, 1998.

[BMS94] C. Burnikel, K. Mehlhorn, and S. Schirra. How to compute the Voronoi diagram of line segments: Theoretical and experimental results. In Springer, editor, *Proceedings of the 2nd Annual European Symposium on Algorithms - ESA'94*, volume 855 of *Lecture Notes in Computer Science*, pages 227–239, 1994.

[CGA] CGAL (Computational Geometry Algorithms Library). www.cs.ruu.nl/CGAL.

[Cla92] K.L. Clarkson. Safe and effective determinant evaluation. In *Proceedings of the 31st Annual Symposium on Foundations of Computer Science (FOCS'92)*, pages 387–395, 1992.

[DFMT00] O. Devillers, A. Fronville, B. Mourrain, and M. Teillaud. Algebraic methods and arithmetic filtering for exact predicates on circle arcs. In *16th ACM Symposium on Computational Geometry*, pages 139–147, 2000.

[FLPR99] Frigo, Leiserson, Prokop, and Ramachandran. Cache-oblivious algorithms. In *FOCS: IEEE Symposium on Foundations of Computer Science (FOCS)*, 1999.

[FM91] S. Fortune and V.J. Milenkovic. Numerical stability of algorithms for line arrangements. In *Proceedings of the 7th Annual ACM Symposium on Computational Geometry (SCG'91)*, pages 334–341. ACM Press, 1991.

[FvW96] S. Fortune and C. van Wyk. Static analysis yields efficient exact integer arithmetic for computational geometry. *ACM Transactions on Graphics*, 15:223–248, 1996.

[GR98] A.V. Goldberg and S. Rao. Beyond the flow decomposition barrier. *JACM*, 45(5), 1998.

[GSS93] L.J. Guibas, D. Salesin, and J. Stolfi. Constructing strongly convex approximate hulls with inaccurate primitives. *Algorithmica*, 9:534–560, 1993.

[Hoc96] D. S. Hochbaum, editor. *Approximation Algorithms for NP-Hard Problems.* PWS Publishing Company, 1996.

[KLN91] M. Karasick, D. Lieber, and L.R. Nackman. Efficient Delaunay triangulation using rational arithmetic. *ACM Transactions on Graphics*, 10(1):71–91, January 1991.

[KLPY99] V. Karamcheti, C. Li, I. Pechtchanski, and Chee Yap. A core library for robust numeric and geometric computation. In *Proceedings of the 15th Annual ACM Symposium on Computational Geometry*, pages 351–359, Miami, Florida, 1999.

[LED] LEDA (Library of Efficient Data Types and Algorithms). www.mpi-sb.mpg.de/LEDA/leda.html.

[LL97] A. LaMarca and R.E. Ladner. The influence of caches on the performance of sorting. In *Proceedings of the 8th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'97)*, pages 370–379, 1997.

[LY00] Z. Li and C.K. Yap. A new constructive root bound for algebraic expressions. 2000.

[Mil88] V.J. Milenkovic. *Verifiable Implementations of Geometric Algorithms Using Finite Precision Arithmetic.* PhD thesis, Carnegie Mellon University, 1988.

[MN94] K. Mehlhorn and S. Näher. The implementation of geometric algorithms. In *Proceedings of the 13th IFIP World Computer Congress*, volume 1, pages 223–231. Elsevier Science B.V. North-Holland, Amsterdam, 1994. www.mpi-sb.mpg.de/ mehlhorn/ftp/ifip94.ps.

[MN99] K. Mehlhorn and S. Näher. *The LEDA Platform for Combinatorial and Geometric Computing*. Cambridge University Press, 1999. 1018 pages.

[MS96] D. R. Musser and A. Saini. *STL Tutorial and Reference Guide: C++ Programming with the Standard Template Library*. Addison-Wesley, Reading (MA), USA, 1996.

[MS00a] K. Mehlhorn and P. Sanders. Scanning multiple sequences via cache memory. www.mpi-sb.mpg.de/ mehlhorn/ftp/cache.ps, February 2000.

[MS00b] K. Mehlhorn and S. Schirra. A generalized and improved constructive separation bound for real algebraic expressions. Technical Report MPI-I-2000-004, Max-Planck-Institut für Informatik, 2000. www.mpi-sb.mpg.de/˜mehlhorn/ftp/improved-sepbound.ps.

[SOI90] K. Sugihara, Y. Ooishi, and T. Imai. Topology-oriented approach to robustness and its applications to several voronoi-diagram algorithms. In *Proceedings of the 2nd Canadian Conference in Computational Geometry (CCCG'90)*, pages 36–39, 1990.

[Vit98] J. S. Vitter. External memory algorithms. *Lecture Notes in Computer Science*, 1461:1–??, 1998.

[WB97] H. Wasserman and M. Blum. Software reliability via run-time result-checking. *Journal of the ACM*, 44(6):826–849, 1997.

[YD95] C.K. Yap and T. Dube. The exact computation paradigm. In *Computing in Euclidean Geometry II*. World Scientific Press, 1995.