The LEDA Platform for Combinatorial and Geometric Computing

Kurt Mehlhorn* and Stefan Näher** and Christian Uhrig***

Abstract. We give an overview of the LEDA platform for combinatorial and geometric computing and an account of its development. We discuss our motivation for building LEDA and to what extent we have reached our goals. We also discuss some recent theoretical developments. This paper contains no new technical material. It is intended as a guide to existing publications about the system. We refer the reader also to our web-pages for more information.

1 What is LEDA?

LEDA [MN95, MNU96] aims at being a comprehensive software platform for combinatorial and geometric computing. It provides a sizable collection of data types and algorithms. This collection includes most of the data types and algorithms described in the text books of the area ([AHU83, Meh84, Tar83, CLR90, O'R94, Woo93, Sed91, Kin90, van88, NH93]). In particular, it includes stacks, queues, lists, sets, dictionaries, ordered sequences, partitions, priority queues, directed, undirected, and planar graphs, lines, points, planes, and polygons, and many algorithms in graph and network theory and computational geometry, e.g., shortest paths, matchings, maximum flow, min cost flow, planarity testing, spanning trees, biconnected and strongly connected components, segment intersection, convex hulls, Delaunay triangulations, and Voronoi diagrams. LEDA supports applications in a broad range of areas. It has already been used in such diverse areas as code optimization, VLSI design, graph drawing, graphics, robot motion planning, traffic scheduling, machine learning and computational biology.

We discuss different aspects of the LEDA system.

Ease of Use: The library is easy to use. In fact, only a small fraction of our users are algorithms experts and many of our users are not even computer scientists. For these users the broad scope of the library, its ease of use, and the correctness and efficiency of the algorithms in the library are crucial.

* Max-Planck-Institut für Informatik, Im Stadtwald, 66123 Saarbrücken, `www.mpi-sb.mpg.de/~mehlhorn`

** Martin-Luther-Universität Halle-Wittenberg, FB Mathematik und Informatik, Weinbergweg 17, 060099 Halle, `www.informatik.uni-halle.de/~naeher`

*** LEDA Software GmbH 66123 Saarbrücken, `www.mpi-sb.mpg.de/LEDA/leda.html`

The LEDA manual [MNU96] gives precise and readable specifications for the data types and algorithms mentioned above. The specifications are short (typically not more than a page), general (so as to allow several implementations) and abstract (so as to hide all details of the implementation).

Extendibility: Combinatorial and geometric computing is a diverse area and hence it is impossible for a library to provide ready-made solutions for all application problems. For this reason it is important that LEDA is easily extendible (see also section 4.4) and can be used as a platform for further software development. In many cases LEDA programs are very close to the typical text book presentation of the underlying algorithms. The goal is the equation

$$\text{Algorithm} + \text{LEDA} = \text{Program}.$$

We give an example. Dijkstra's shortest path algorithm takes a directed graph $G = (V, E)$, a node $s \in V$, called the source, and a non-negative cost function on the edges $cost : E \to R_{\geq 0}$. It computes for each node $v \in V$ the distance from s. A typical text book presentation of the algorithm is as follows.

```
set dist(s) to 0.
set dist(v) to infinity for v different from s.

declare all nodes unreached.

while there is an unreached node
{ let u be an unreached node with minimal dist-value.          (*)

  declare u reached.

  forall edges e = (u,v) out of u
     set dist(v) = min( dist(v), dist(u) + cost(e) )
}
```

The text book presentation will then continue to discuss the implementation of line (*). It will state that the pairs $\{(v, dist(v)); v \text{ unreached}\}$ should be stored in a priority queue, e.g., a Fibonacci heap, because this will allow the selection of an unreached node with minimal distance value in logarithmic time. It will probably refer to some other chapter of the book for a discussion of priority queues.

We now give the corresponding LEDA program; it is very similar to the presentation above.

```
#include <LEDA/graph.h>
#include <LEDA/node_pq.h>

void DIJKSTRA(const graph &G, node s, const edge_array<double>& cost,
                                          node_array<double>& dist)
{ node_pq<double> PQ(G);
  node v;
  edge e;

  forall_nodes(v,G)
```

```
  { if (v == s) dist[v] = 0; else dist[v] = MAXDOUBLE;
    PQ.insert(v,dist[v]);
  }
  while ( !PQ.empty() )
  { node u = PQ.del_min();
    forall_adj_edges(e,u)
      { v = target(e);
        double c = dist[u] + cost[e];
        if ( c < dist[v] )
        {  PQ.decrease_inf(v,c);  dist[v] = c;  }
      }
  }
}
```

We start by including the graph and the node priority queue data type. We use *edge_arrays* and *node_arrays* (arrays indexed by edges and nodes respectively) for the functions *cost* and *dist*. We declare a priority queue PQ for the nodes of graph G. It stores pairs $(v, dist[v])$ and is empty initially. The `forall_nodes`-loop initializes *dist* and PQ. In the main loop we repeatedly select a pair $(u, dist[u])$ with minimal distance value and then scan through all adjacent edges to update distance values of neighboring vertices.

Correctness: We try to make sure that the programs in LEDA are correct. We start from correct algorithms, we document our implementations carefully (at least recently), we test them extensively, and we have developed program checkers (see subsection 4.1) for some of them. We want to emphasize that many of the algorithms in LEDA are quite intricate and therefore non-trivial to implement. In the combinatorial domain it is frequently possible to obtain a correct implementation by sacrificing efficiency, e.g., by using linear search in the realization of a dictionary. In the geometric domain it is usually difficult to obtain a correct implementation even if efficiency plays no role. This is due to the so-called degeneracy and precision problem [MN94]. The geometric algorithms in LEDA use exact arithmetic and are therefore free from failures due to rounding errors. Moreover, they can handle all degenerate cases.

Efficiency: LEDA contains the most efficient realizations known for its types. For many data types the user may even choose between different implementations, e.g., for dictionaries he may choose between ab-trees, $BB[\alpha]$-trees, dynamic perfect hashing, and skip lists. The declarations

```
dictionary<string,int> D1;
dictionary<string,int,skip_list> D2;
```

declare *D1* as a dictionary from *string* to *int* with the default implementation and select the skip list implementation for *D2*.

Availability and Usage: LEDA is realized in C++ and runs on many different platforms (Unix, Windows95, Windows NT, OS/2) with many different compilers.

LEDA is now used at more than 1500 academic sites. Academic use is free, see `http://www.mpi-sb.mpg.de/LEDA/leda.html`. A commercial version of LEDA is marketed LEDA Software GmbH. There are license holders in the telecommunication industry (ATR (Japan), Comptel (Finland), E-Plus (Germany), France Télécom (France), MCI (USA)), in the graphics industry (Aristo Technologies (USA), Cadabra (Canada), Compass Design (USA), Fuji (Japan), Mentor Graphics (USA), MUS (Germany)), in the automotive industrie (Daimler Benz (Germany), Ford (USA), Honda (Japan)), in the computer industry (DEC (USA), IBM (USA), Siemens AG (Germany), Silicon Graphics (USA), SUN (USA)), and other industries (Chevron (USA), CFP (Germany), Dolphin (The Netherlands), Howmedica (Germany), Lufthansa (Germany), Neovista (USA), Prediction (USA), Sony (Japan), VTT (Finland)).

History: We started the project in the fall of 1988. We spent the first 6 months on specifications and on selecting our implementation language. Our test cases were priority queues, dictionaries, partitions, and algorithms for shortest paths and minimum spanning trees. We came up with the item concept as an abstraction of the notion "pointer into a data structure". It worked successfully for the three data types mentioned above and we are now using it for most data types in LEDA. Concurrently with searching for the correct specifications we investigated several languages for their suitability as our implementation platform. We looked at Smalltalk, Modula, Ada, Eiffel, and C++. We wanted a language that supported abstract data types and type parameters (polymorphism) and that was widely available. We wrote sample programs in each language. Based on our experiences we selected C++ because of its flexibility, expressive power, and availability. We are even more convinced now that our choice was the right one.

A first publication about LEDA appeared in MFCS 1989 (Lecture Note in Computer Science, Volume 379) and ICALP 1990 (Lecture Notes in Computer Science, Volume 443). Stefan Näher became the head of the LEDA project and he is the main designer and implementer of LEDA.

In the second half of 1989 and during 1990 Stefan Näher implemented a first version of the combinatorial part (= data structures and graph algorithms) of LEDA (Version 1.0). Version 2.0 allowed to use arbitrary data types (not only pointer and simple types) as actual type parameters of parameterized data types. It included a first implementation of the two-dimensional geometry library (libP) and an interface to the X-Window system for graphical input and output (data type window). Version 3.0 switched to the template mechanism to realize parameterized data types (macro substitution was used before), introduced implementation parameters that allow to choose between different implementations, extended the LEDA memory management system to user-defined classes, and further improved the efficiency of many data types and algorithms. Version 3.1 provided a more efficient graph data type and contained new data types

(arbitrary precision number types and basic geometric objects) used for robust implementations of geometric algorithms and Versions 3.2 and 3.3 contained more geometry and new tools for documentation and manual production.

LEDA Software GmbH was founded in early 1995.

2 Why did we build LEDA?

We had four main reasons:

1. We had always felt that a significant fraction of the research done in the algorithms area was eminently practical. However, only a small part of it was actually used. We frequently heard from our former students that the effort needed to implement an advanced data structure or algorithm is too large to be cost-effective. We concluded that *algorithms research must include implementation if the field wants to have maximum impact.*
2. Even within our own research group we found different implementations of the same balanced tree data structure. Thus there was constant reinvention of the wheel even within our own tight group.
3. Many of our students had implemented algorithms for their master's thesis. Work invested by these students was usually lost after the students graduated. We had no depository for implementations.
4. The specifications of advanced data types which we gave in class and which we found in text books, including the one written by one of the authors, were incomplete and not sufficiently abstract. They contained phrases of the form: "Given a pointer to a node in the heap its key can be decreased in constant amortized time". This implied that a user of a data structure had to have knowledge of its implementation. As a consequence combining implementations was a non-trivial task. A case in point is the shortest path problem in graphs. We taught priority queues in the early weeks of an algorithm course and Dijkstra's algorithm for the shortest path problem in later weeks. Our students found it difficult to combine the programs.

The goal of the LEDA project is to overcome these shortcomings by creating a platform for combinatorial and geometric computing. The LEDA library should contain the major findings of the algorithms community in a form that makes them directly accessible to non-experts having only a limited knowledge in the area. In this way we hoped to reduce the gap between research and application.

3 Did we achieve our goals?

We believe that we have reached the last goal and have at least partially reached the first three goals.

LEDA was first distributed in the summer of 1990. Its user community has grown ever since. LEDA is now used at more than 1500 academic and industrial sites in over 50 different countries world-wide. Industrial use started in 1994.

Many users of LEDA are outside computer science and only a small fraction of our users are from the algorithms community. We therefore believe that we have reached our first two goals. The impact of algorithms research has increased and there is considerable use of LEDA and hence reuse of implementations. However, the gap between algorithms research and algorithms use is still quite large. In particular, many of the non-expert users of LEDA complain that a tutorial is missing. We hope that the forthcoming LEDAbook [MN] will help.

We have also partially achieved our third goal. We now do have a depository for our students work and we have just introduced the concept of LEDA extension packages (LEPs) that will allow a wider community to contribute. We come back to LEPs in section 4.4.

We have achieved our last goal. The specifications of our data types are sufficiently abstract and precise so as to allow their combination without any knowledge of implementation. We have seen an example in section 1. Many of our specifications are based on the so-called *item concept* which gives an abstract treatment of pointers into a data structure. Different components of LEDA can be combined without knowledge of the implementation.

The project also had a number of positive side-effects which we did not foresee. Firstly, LEDA's wide use gives us tremendous satisfaction[4]. Secondly, our experiences with the system suggested many difficult and well motivated problems for theoretical algorithms research. We will discuss program checking, running time prediction, and theoretical issues in the implementation of geometric algorithms below. *The system has changed the way we do algorithms research.*

4 Recent developments

A strength of the LEDA project is its strong theoretical underpinning. *We believe that only our strong theoretical background allowed us to build LEDA.* In the last two years we paid particular attention to program checking, running time prediction, and the correct implementation of geometric programs.

4.1 Program checking

Programming is a notoriously error-prone task; this is even true when programming is interpreted in a narrow sense: going from a (correct) algorithm to a program. The standard way to guard against coding errors is program testing. The program is exercised on inputs for which the output is known by other means, typically as the output of an alternative program for the same task. Program testing has severe limitations:

- It is usually only done during the testing phase of a program. Also, it is difficult to determine the "correct" suite of test inputs.

[4] We stated above that algorithms research must include implementation to have maximal impact. We might add: without implementation algorithm research is less rewarding.

- Even if appropriate test inputs are known it is usually difficult to determine the correct outputs for these inputs: alternative programs may have different input and output conventions or may be too inefficient to solve the test cases.

Given that program verification, i.e., formal proof of correctness of an implementation, will not be available on a practical scale for some years to come, *program checking* has been proposed as an extension to testing [BK89, BLR90]. The cited papers explored program checking in the area of algebraic, numerical, and combinatorial computing. In [MNS+96, MM95, HMN96] we discuss program checkers for planarity testing and a variety of geometric tasks. We have also added program checkers to some of the LEDA programs, e.g., the planarity test provides a planar drawing for a planar graph and a Kuratowski subgraph for a non-planar graph. A user of the planarity algorithm has thus the possibility to verify that the output of the algorithm is correct.

4.2 Running Time Prediction

Big-O analysis of algorithms is concerned with the asymptotic analysis of algorithms, i.e., with the behavior of algorithms for large inputs. It does not allow the prediction of actual running times of real programs on real machines and therefore its predictive value is limited.

- An algorithm with running time $O(n)$ is faster than an algorithm with running time $O(n^2)$ for sufficiently large n. Is $n = 10^6$ large enough? Asymptotic analysis of algorithms is of little help to answer this question. It is however true that a well-trained algorithms person who knows program and analysis can make a fairly good guess.
- For a user of LEDA statements of asymptotic running times are almost meaningless as he/she has no way to estimate the constants involved. After all, the purpose of LEDA is to hide the implementations from our users.

The two items above clearly indicate that we need more than asymptotic analysis in order to have a theory with predictive value. *The ultimate goal of analysis of algorithms must be a theory that allows to predict the actual running time of an actual program on an actual machine* with reasonable precision (say within a factor of two). We must aim for the following scenario: When a program is installed on a particular machine a certain number of well-chosen tests are executed in order to learn about machine parameters relevant for the execution of the program. This knowledge about the machine is combined with the analysis of the algorithm to predict running time on specific inputs. In the context of an algorithms library one could even hope to replace statements about asymptotic execution times by statements about actual execution times during installation of the library. In [FM97] we show for a small number of programs (Fibonacci heaps, Dijkstra's shortest path algorithm, and a maximum weight matching algorithm) that running time prediction within a factor of less than two and a wide range of machines is feasible.

4.3 Implementation of geometric algorithms

Geometric algorithms are frequently formulated under two unrealistic assumptions: computers are assumed to use exact real arithmetic (in the sense of mathematics) and inputs are assumed to be in general position. The naive use of floating point arithmetic as an approximation to exact real arithmetic very rarely leads to correct implementations. In a sequence of papers [BMS94a, See94, MN94, BMS94b, FGK+96, BRMS97] we investigated the degeneracy and precision issues and extended LEDA based on our theoretical work. LEDA now provides exact geometric kernels for two-dimensional and higher dimensional computational geometry [MMN+97] and also correct implementations for basic geometric tasks, e.g., two-dimensional convex hulls, Delaunay diagrams, Voronoi diagrams, point location, line segment intersection, and higher-dimensional convex hulls and Delaunay diagrams.

4.4 LEDA Extension Packages

LEDA extension packages are a new feature of the LEDA project structure. Up to two years ago, most of LEDA has been developed by a small group of persons under the tight supervision of Stefan Näher; no code went into the system that was not thoroughly understood by either Stefan Näher or Christian Uhrig. The growing numbers of contributors and the fact that Stefan Näher has new responsibilities as a professor has forced us to a change of the project structure. We decided to split LEDA into a core system (the actual LEDA version) and to shift enhancements into additional software packages.

LEDA extension packages (LEPs) extend LEDA into particular application domains and areas of algorithmics not covered by the core system. LEDA extension packages satisfy requirements, which guarantee compatibility with the LEDA philosophy. LEPs have a LEDA-style documentation, they are implemented as platform independent as possible and the installation process allows a close integration into the LEDA core library.

Currently, there are no released LEPs available, but there are several LEP under construction: PQ-trees (coordinated by Sebastian Leipert, Koeln), dynamic graph algorithms (coordinated by David Alberts, Halle), the homogeneous planar CGAL geokernel (coordinated by Stefan Schirra, Saarbrücken), a homogeneous d-dimensional geokernel (coordinated by Michael Seel, Saarbrücken), and a library for graph drawing (DFG-project Automatisches Graphenzeichnen).

References

[AHU83] A.V. Aho, J.E. Hopcroft, and J.D. Ullman. *Data structures and algorithms.* Addison-Wesley, 1983.

[BK89] M. Blum and S. Kannan. Programs That Check Their Work. In *Proc. of the 21th Annual ACM Symp. on Theory of Computing*, 1989.

[BLR90] M. Blum, M. Luby, and R. Rubinfeld. Self-testing/correcting with applications to numerical problems. In *Proc. 22nd Annual ACM Symp. on Theory of Computing*, pages 73–83, 1990.

[BMS94a] Ch. Burnikel, K. Mehlhorn, and S. Schirra. On degeneracy in geometric computations. In *Proc. SODA 94*, pages 16–23, 1994.

[BMS94b] Ch. Burnikel, K. Mehlhorn, and St. Schirra. How to compute the Voronoi diagram of line segments: Theoretical and experimental results. In Springer-Verlag Berlin/New York, editor, *LNCS*, volume 855 of *Proceedings of ESA '94*, pages 227–239, 1994.

[BRMS97] Ch. Burnikel, R.Fleischer, K. Mehlhorn, and S. Schirra. A strong and easily computable separation bound for arithmetic expressions involving square roots. In *Proc. SODA 97*, pages 702–709, 1997.

[CLR90] T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to Algorithms.* MIT Press/McGraw-Hill Book Company, 1990.

[FGK+96] A. Fabri, G.-J. Giezeman, L. Kettner, S. Schirra, and S. Schönherr. The CGAL Kernel: A basis for geometric computation. In *Workshop on Applied Computational Geometry (WACG96)*, LNCS, 1996.

[FM97] Ulrich Finkler and Kurt Mehlhorn. Runtime prediction of real programs on real machines. In *Proceedings 8th ACM-SIAM Symposium on Discrete Algorithms (SODA '97)*, January 1997.

[HMN96] C. Hundack, K. Mehlhorn, and S. Näher. A Simple Linear Time Algorithm for Identifying Kuratowski Subgraphs of Non-Planar Graphs. Manuscript, 1996.

[Kin90] J.H. Kingston. *Algorithms and Data Structures.* Addison-Wesley Publishing Company, 1990.

[Meh84] K. Mehlhorn. *Data structures and algorithms 1,2, and 3.* Springer, 1984.

[MM95] K. Mehlhorn and P. Mutzel. On the Embedding Phase of the Hopcroft and Tarjan Planarity Testing Algorithm. *Algorithmica*, 16(2):233–242, 1995.

[MMN+97] K. Mehlhorn, Müller, S. Näher, S. Schirra, M. Seel, C. Uhrig, and J. Ziegler. A computational basis for higher-dimensional computational geometry and its applications. In *Proceedings of the Symp. on Computational Geometry*, 1997. http://www.mpi-sb.mpg.de/~seel.

[MN] K. Mehlhorn and S. Näher. The LEDA Platform for Combinatorial and Geometric Computing. Cambridge University Press, forthcoming. Draft versions of some chapters are available at `http://www.mpi-sb.mpg.de/~mehlhorn`.

[MN94] K. Mehlhorn and S. Näher. The implementation of geometric algorithms. In *13th World Computer Congress IFIP94*, volume 1, pages 223–231. Elsevier Science B.V. North-Holland, Amsterdam, 1994.

[MN95] K. Mehlhorn and S. Näher. LEDA: A platform for combinatorial and geometric computing. *Communications of the ACM*, 38(1):96–102, 1995.

[MNS+96] K. Mehlhorn, S. Näher, T. Schilz, S. Schirra, M. Seel, R. Seidel, and Ch. Uhrig. Checking Geometric Programs or Verification of Geometric Structures. In *Proc. of the 12th Annual Symposium on Computational Geometry*, pages 159–165, 1996.

[MNU96] Kurt Mehlhorn, S. Näher, and Ch. Uhrig. The LEDA User Manual (Version R 3.4). Technical report, Max-Planck-Institut für Informatik, 1996. http://www.mpi-sb.mpg.de/LEDA/leda.html.

[NH93] J. Nievergelt and K.H. Hinrichs. *Algorithms and Data Structures.* Prentice Hall Inc., 1993.

[O'R94] J. O'Rourke. *Computational Geometry in C.* Cambridge University Press, 1994.

[Sed91] R. Sedgewick. *Algorithms.* Addison-Wesley Publishing Company, 1991.

[See94] Michael Seel. Eine Implementierung abstrakter Voronoidiagramme. Master's thesis, Max-Planck-Institut für Informatik, 1994.

[Tar83] R.E. Tarjan. Data structures and network algorithms. In *CBMS-NSF Regional Conference Series in Applied Mathematics*, volume 44, 1983.

[van88] C.J. van Wyk. *Data Structures and C programs.* Addison-Wesley Publishing Company, 1988.

[Woo93] D. Wood. *Data Structures, Algorithms, and Performance.* Addison-Wesley Publishing Company, 1993.