it 6/2011

Algorithm Engineering

Algorithm Engineering

Peter Sanders, Dorothea Wagner, Karlsruhe Institute of Technology

Algorithms and data structures are at the heart of every computer application and thus of decisive importance for permanently growing areas of engineering, economy, science, and daily life. The searching and ranking algorithms in search engines and the pattern matching algorithms crucial for reading the human genome are only a few spectacular examples how algorithms can change our life.

But how is algorithmic innovation transferred to applications? Traditionally, algorithmics used the methodology of *algorithm theory* which stems from mathematics: Algorithms are designed using simple models of problem and machine. Main results are provable performance guarantees for all possible inputs. This approach often leads to elegant, highly reliable, and timeless solutions that can be adapted to many applications. From the point of view of algorithm theory, taking up and implementing an algorithmic idea is part of application development. Unfortunately, it can be universally observed that this mode of transferring results is a slow process. With growing requirements for innovative algorithms, this causes growing gaps between theory and practice: Realistic hardware with its parallelism, memory hierarchies etc. is diverging from traditional machine models. Applications grow more and more complex. At the same time, algorithm theory develops more and more elaborate algorithms that may contain important ideas but are usually not directly implementable. Furthermore, real-world inputs are often far away from the worst case scenarios of the theoretical analysis. In extreme cases, promising algorithmic approaches are neglected because a mathematical analysis would be difficult.

Since the early 1990s it therefore became more and more apparent that algorithmics cannot restrict itself to theory. So, what else should algorithmicists do? *Experiments* play a pivotal here. Algorithm engineering (AE) is therefore sometimes equated with *experimental algorithmics*. However, this view is too limited. First of all, to do experiments, you also have to *implement* algorithms. This is often equally interesting and revealing as the experiments themselves, needs its own set of techniques, and is an important interface to software engineering. Furthermore, it makes little sense to view design and analysis on the one hand and implementation and experimentation on the other hand as separate activities. Rather, a feedback loop of design, analysis, implementation, and experimentation that leads to new design ideas materializes as the central process of algorithmics.

This cycle is quite similar to the cycle of theory building and experimental validation in Popper's scientific method. We can learn several things from this comparison. First, this cycle is driven by *falsifiable hypotheses* validated by experiments – an experiment cannot prove a hypothesis but it can support it. However, such support is only meaningful if there are conceivable outcomes of experiments that prove the hypothesis wrong. Hypotheses can come from creative ideas or result from *inductive reasoning* stemming from previous experiments. Thus we see a fundamental difference to the *deductive reasoning* predominant in algorithm theory. Experiments have to be *reproducible*, i. e., other researchers have to be able to repeat an experiment to the extent that they draw the same conclusions or uncover mistakes in the previous experimental setup.

There are further aspects of AE as a methodology for algorithmics, outside the main cycle. Design, analysis and evaluation of algorithms are based on some realistic *model* of the problem and the underlying machine. *Applications* are coupled to the AE process in many ways. In particular, *realistic inputs* are important for meaningful experiments. *Application engineering* is the process of adapting experimental implementations to the particular requirements of an application. Since algorithm engineers may not know all the applications for which their algorithms will be used, *algorithm libraries* of highly tested codes with clear simple user interfaces are a further important link between AE and applications.

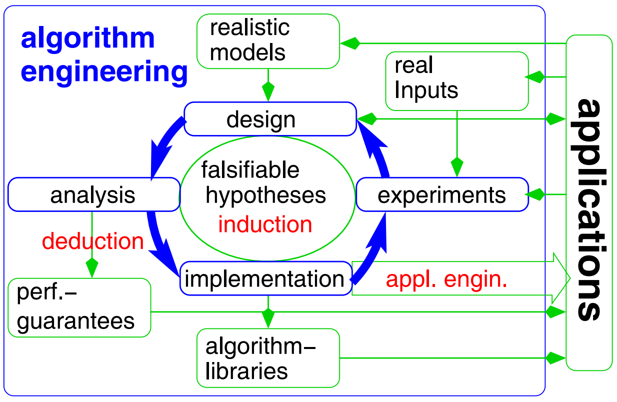

Figure 1 Algorithm engineering as a cycle of design, analysis, implementation, and experimental evaluation driven by falsifiable hypotheses.

Figure 1 summarizes the resulting schema for AE as a methodology for algorithmics. The articles in this special issue have been selected to exemplify the key activity areas in this diagram. Some of them come from participants in the DFG priority program on algorithm engineering (http://www.algorithm-engineering.de/) which has greatly strengthened AE research in Germany.

Bader, Ediger, and Kang discuss the challenges coming from realistic *machine models*: ever increasing parallelism, deep memory hierarchies, heterogeneous architectures, and the need to adapt every single performance critical application because the free lunch of ever increasing clock speeds is over.

Real time scheduling used to be an area with important practical applications but considerable gaps to theory. Eisenbrand et al. report on the solution of a scheduling problem arising in the avionics industry where current state-of-the-art approaches to tackle the problem are by far not powerful enough to solve instances of real-world size. Using the AE paradigm they analyzed the mathematical properties of the scheduling problem and *designed* sophisticated solution methods based on the structural insights. The result is a model that outperforms current state-of-the-art approaches by several orders of magnitude and solves industrial size real-world instances to optimality.

Manthey and Röglin show how AE can imply interesting new challenges for algorithm analysis beyond traditional worst case analysis. They give two interesting examples how *smoothed analysis* can fathom the gap between worst case (which is often too pessimistic) and plain average case analysis (which is usually unrealistic). A knapsack algorithm that takes exponential time in the worst case becomes efficient in this model. Similarly, smoothed analysis explains why a well known local search algorithm for a facility location problem works well in practice.

The article on *certifying algorithms* by Mehlhorn et al. explains an effective way to ensure correct results of algorithm *implementations*. The idea is to check each returned result for correctness using a simple checker. It then suffices to test or perhaps verify the checker. Making checking fast implies interesting algorithmic questions when checking is aided by *certificates* of correctness computed by the main algorithm.

Delling, Goldberg and Werneck report on techniques for fast *route planning* that have developed into a show piece of AE in the last years – leading to exact route computation in continental size road networks within less than a microsecond. *Experiments* play a pivotal role here because real world networks have special properties like a pronounced hierarchy that enormously simplify the problem compared to the worst case. They also give an example, where closing the AE cycle and going back to theory gives additional insights and motivates further improved algorithms.

The last article by Möhring illustrates the differences in methodology of algorithm engineering in a lab environment and in real world *applications*. The article selects two applications – a project on improving the routing of automated guided vehicles in a container terminal and a project on improving the coating of coils in a steel company. The author shows the different aspects of a close interaction and collaboration with industrial partners. In particular, he gives some insight into his experiences with "playing this game" and how it can lead to great and unexpected scientific results.

[illegible]

Prof. Dr. Peter Sanders

[illegible]

Prof. Dr. Dorothea Wagner

Received: July 20, 2011

Prof. Dr. rer. nat. Peter Sanders received his Ph.D. in computer science from Universität Karlsruhe, Germany in 1996. After 7 years at the Max-Planck-Institute for Informatics in Saarbrücken he returned to Karlsruhe as a full professor in 2004. In 2004 he was also awarded the Alcatel SEL Research Prize. He has more than 150 publications, mostly on algorithm for large data sets. This includes parallel algorithms, memory hierarchies, graph algorithms (route planning, graph partitioning etc.), randomized algorithms, full text indices, etc.

Address: Karlsruhe Institute of Technology, Am Fasanengarten 5, 76137 Karlsruhe, Germany

Prof. Dr. rer. nat. Dorothea Wagner obtained her diploma and Ph.D. degrees in mathematics from the RWTH Aachen in 1983 and 1986, respectively, and 1992 the Habilitation degree from the TU Berlin. 1994–2003 she was a full professor in Konstanz before she moved to Karlsruhe in 2003 as a full professor for Informatics. Among other activities she is vice president of the DFG (Deutsche Forschungsgemeinschaft – German Research Foundation) and speaker of the scientific advisory board of Dagstuhl – Leibniz Center for Informatics. Her research interests include design and analysis of algorithms and algorithm engineering, graph algorithms, computational geometry and discrete optimization, particularly applied to transportation systems, network analysis, data mining and visualization.

Address: Karlsruhe Institute of Technology, Am Fasanengarten 5, 76137 Karlsruhe, Germany