Algorithmen II

Peter Sanders

Übungen:
Moritz Laupichler, Nikolai Maas

Institut für Theoretische Informatik

Web:
algo2.iti.kit.edu/AlgorithmenII_WS23.php
1 Algorithm Engineering

A detailed definition

☐ In general
[with Kurt Mehlhorn, Rolf Möhring, Petra Mutzel, Dorothea Wagner]

☐ A few examples, usually sorting

☐ A little bit on experimental methodology
(Caricatured) Traditional View: Algorithm Theory

Theory
- Design
- Analysis
- Implementation
- Performance Guarantees

Practice
- Model
- Applications

Other Disciplines
- Time Scale?
- Publication Culture?
# Gaps Between Theory & Practice

<table>
<thead>
<tr>
<th>Theory</th>
<th>↔️</th>
<th>Practice</th>
</tr>
</thead>
<tbody>
<tr>
<td>simple</td>
<td></td>
<td>appl. model</td>
</tr>
<tr>
<td>machine model</td>
<td></td>
<td>complex</td>
</tr>
<tr>
<td>algorithms</td>
<td></td>
<td>real</td>
</tr>
<tr>
<td>data structures</td>
<td></td>
<td>simple</td>
</tr>
<tr>
<td>complex</td>
<td></td>
<td></td>
</tr>
<tr>
<td>advanced</td>
<td></td>
<td>FOR</td>
</tr>
<tr>
<td>complexity measure</td>
<td></td>
<td>arrays,…</td>
</tr>
<tr>
<td>asympt.</td>
<td></td>
<td>efficiency</td>
</tr>
<tr>
<td>max</td>
<td></td>
<td>inputs</td>
</tr>
<tr>
<td>O(·)</td>
<td></td>
<td>42%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>constant factors</td>
</tr>
</tbody>
</table>
Algorithmics as Algorithm Engineering
Algorithmics as Algorithm Engineering

- Bridge gaps between theory and practice
Algorithmics as Algorithm Engineering

- Bridge gaps between theory and practice
- Integrated interdisciplinary research
Algorithmics as Algorithm Engineering

Algorithm Engineering

- design
- analyse
- implement
- model
- experiment
- benchmarks
- perf. guarantees
- alg. libraries
- induction
- falsifiable hypotheses
- deduction
- appl. engineering
Bits of History

1843– Algorithms in theory and practice

1950s, 1960s Still infancy

1970s, 1980s Paper and pencil algorithm theory.

Exceptions exist, e.g., [D. Johnson], [J. Bentley]

1986 Term used by [T. Beth],

lecture “Algorithmentechnik” in Karlsruhe.

1988– Library of Efficient Data Types and Algorithms

(LEDÁ) [K. Mehlhorn]

1997– Workshop on Algorithm Engineering

エネル ESA applied track [G. Italiano]

1997 Term used in US policy paper [Aho, Johnson, Karp, et. al]

1998 Alex workshop in Italyエネル ALENEX
# Realistic Models

<table>
<thead>
<tr>
<th>Theory</th>
<th>→→→</th>
<th>Practice</th>
</tr>
</thead>
<tbody>
<tr>
<td>simple</td>
<td>appl. model</td>
<td>complex</td>
</tr>
<tr>
<td>simple</td>
<td>machine model</td>
<td>real</td>
</tr>
</tbody>
</table>

- **Careful** refinements

- Try to preserve (partial) analyzability / simple results
Design of algorithms that work well in practice

- simplicity
- reuse
- constant factors
- exploit easy instances
Analysis

☐ Constant factors matter
  Example: quicksort

☐ Beyond worst case analysis

☐ Practical algorithms might be difficult to analyze
  (randomization, meta heuristics, . . .)
Implementation

sanity check for algorithms!

Challenges

Semantic gaps:

Abstract algorithm
⇔
C++...
⇔
hardware
Experiments

☐ Sometimes a good surrogate for analysis

☐ Too much rather than too little output data

☐ Reproducibility (10 years!)

☐ Software engineering

Stay tuned.
Algorithm Libraries — Challenges

- Software engineering
- Standardization, e.g. java.util, C++ STL and BOOST
- Performance ↔ generality ↔ simplicity
- Applications are a priori unknown
- Result checking, verification

STXXL

- STL–user layer
  - Containers: vector, stack, set, priority_queue, map
  - Algorithms: sort, for_each, merge
- Streaming layer
  - Pipelined sorting, zero-I/O scanning
- Block management layer
  - Typed block, block manager, buffered streams, block prefetcher, buffered block writer
- Asynchronous I/O primitives layer
  - Files, I/O requests, disk queues, completion handlers
- Operating System

Applications

STL Interface
- Serial STL Algorithms
- OpenMP

Extensions
- Parallel STL Algorithms
- Atomic Ops
Problem Instances

Benchmark instances are essential for development of practical algorithms
Example: Sorting Benchmark (Indy)

100 byte records, 10 byte random keys, with file I/O

<table>
<thead>
<tr>
<th>Category</th>
<th>data volume</th>
<th>performance</th>
<th>improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>GraySort</td>
<td>100 000 GB</td>
<td>564 GB / min</td>
<td>17×</td>
</tr>
<tr>
<td>MinuteSort</td>
<td>955 GB</td>
<td>955 GB / min</td>
<td>&gt; 10×</td>
</tr>
<tr>
<td>JouleSort</td>
<td>100 000 GB</td>
<td>3 400 Recs/Joule</td>
<td>???×</td>
</tr>
<tr>
<td>JouleSort</td>
<td>1 000 GB</td>
<td>17 500 Recs/Joule</td>
<td>5.1×</td>
</tr>
<tr>
<td>JouleSort</td>
<td>100 GB</td>
<td>39 800 Recs/Joule</td>
<td>3.4×</td>
</tr>
<tr>
<td>JouleSort</td>
<td>10 GB</td>
<td>43 500 Recs/Joule</td>
<td>5.7×</td>
</tr>
</tbody>
</table>

Also: PennySort
GraySort: inplace multiway mergesort, exact splitting

- Xeon Xeon
- 16 GB RAM
- 240 GB
- Infiniband switch
- 400 MB/s node all–all
JouleSort

- Intel Atom N330
- 4 GB RAM
- $4 \times 256$ GB SSD (SuperTalent)

Algorithm similar to GraySort
Applications that “Change the World”

Algorithmics has the potential to SHAPE applications (not just the other way round) [G. Myers]

**Bioinformatics:** sequencing, proteomics, phylogenetic trees, . . .

**Information Retrieval:** Searching, ranking, . . .

**Traffic Planning:** navigation, flow optimization, adaptive toll, disruption management

**Geographic Information Systems:** agriculture, environmental protection, disaster management, tourism, . . .

**Machine Learning/AI:** translation, chatbots, . . .
Conclusion:

Algorithm Engineering $\leftrightarrow$ Algorithm Theory

- Algorithm engineering is a wider view on algorithmics
  (but no revolution. None of the ingredients is really new)

- Rich methodology

- Better coupling to applications

- Experimental algorithmics $\ll$ algorithm engineering

- Algorithm theory $\subset$ algorithm engineering

- Sometimes different theoretical questions

- Algorithm theory may still yield the strongest, deepest and most persistent results within algorithm engineering
More On Experimental Methodology

Scientific Method:

☐ Experiment need a possible outcome that falsifies a hypothesis

☐ Reproducible
  - Keep data/code for at least 10 years
  + documentation (aka laboratory journal (Laborbuch))
  - Clear and detailed description in papers / TRs
  - Share instances and code
Quality Criteria

- Beat the state of the art, globally – (not your own toy codes or the toy codes used in your community!)
- Clearly demonstrate this!
  - Both codes use same data ideally from accepted benchmarks (not just your favorite data!)
  - Comparable machines or fair (conservative) scaling
  - Avoid uncomparabilities like: “Yeah we have worse quality but are twice as fast”
  - Real world data wherever possible
  - As much different, fresh inputs as possible
  - It’s fine if you are better just on some (important) inputs
Not Here but Important

☐ Describing the setup (machine, compiler, OS, instances, repetitions, . . .)

☐ Finding sources of measurement errors

☐ Reducing measurement errors (averaging, median, unloaded machine . . .)

☐ Measurements in the creative phase of experimental algorithmics.
The Starting Point

- (Several) Algorithm(s)

- A few quantities to be measured: time, space, solution quality, comparisons, cache faults, ... There may also be measurement errors.

- An unlimited number of potential inputs. $\rightsquigarrow$ condense to a few characteristic ones (size, $|V|$, $|E|$, ... or problem instances from applications)

Usually there is not a lack but an abundance of data
The Process

Waterfall model?

1. Design
2. Measurement
3. Interpretation

Perhaps the paper should at least look like that.
The Process

- Eventually stop asking questions (Advisors/Referees listen !)
- Build measurement tools
- Automate (re)measurements
- Choice of Experiments driven by risk and opportunity
- Distinguish mode

  **Explorative:** many different parameter settings, interactive, short turnaround times

  **Consolidating:** many large instances, standardized measurement conditions, batch mode, many machines
Of Risks and Opportunities

Example: Hypothesis \(\equiv\) my algorithm is the best

**Big risk:** untried main competitor

**Small risk:** tuning of a subroutine that takes 20 % of the time.

**Big opportunity:** use algorithm for a new application

\(\leadsto\) new input instances