Algorithmen II

Peter Sanders, Thomas Worsch, Simon Gog

Übungen:
Demian Hespe, Yaroslav Akhremtsev

Institut für Theoretische Informatik, Algorithmik II

Web:
http://algo2.itl.kit.edu/AlgorithmenII_WS17.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

- models
- design
- analysis
- perf. guarantees

Practice

- implementation
- applications

deduction
### Gaps Between Theory & Practice

<table>
<thead>
<tr>
<th><strong>Theory</strong></th>
<th><strong>Theory ↔ Practice</strong></th>
<th><strong>Practice</strong></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>
<tr>
<td>complex</td>
<td>algorithms</td>
<td>simple</td>
</tr>
<tr>
<td>advanced</td>
<td>data structures</td>
<td>arrays,…</td>
</tr>
<tr>
<td>worst case</td>
<td>complexity measure</td>
<td>inputs</td>
</tr>
<tr>
<td>asympt.</td>
<td>efficiency</td>
<td>42% constant factors</td>
</tr>
</tbody>
</table>
Algorithmics as Algorithm Engineering

algorithm engineering

models

design

analysis

deduction

perf.– guarantees

? experiments

?
Algorithmics as Algorithm Engineering

algorithm engineering

models

design

falsifiable hypotheses

induction

analysis
deduction

perf.- guarantees

experiments

implementation
Algorithmics as Algorithm Engineering

Algorithm engineering

realistic models

real. design

falsifiable hypotheses
induction

experiments

implementation

real. analysis
deduction

perf.- guarantees
Algorithmics as Algorithm Engineering

Algorithm engineering

- realistic models
- design
- analysis
- falsifiable hypotheses
- implementation
- experimentation
- real inputs
- perf. guarantees
- induction
- deduction

Algorithm engineering diagram with nodes and arrows representing the process of algorithm development and analysis.
Algorithmics as Algorithm Engineering

algorithm engineering

realistic models

design

falsifiable hypotheses

induction

experiments

applications

real Inputs

algorithm−libraries

appl. engin.

deduction

falsifiable hypotheses

analysis
deduction

perf.−guarantees
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 (LEDA) [K. Mehlhorn]

1997– Workshop on Algorithm Engineering

⇝ ESA applied track [G. Italiano]

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

1998 Alex workshop in Italy ⇝ 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

  Beispiel: 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 $\leftrightarrow$ generality $\leftrightarrow$ simplicity
- applications are a priori unknown
- result checking, verification

<table>
<thead>
<tr>
<th>Applications</th>
<th>STL–user layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Containers:</td>
<td>vector, stack, set</td>
</tr>
<tr>
<td>Algorithms:</td>
<td>priority_queue, map</td>
</tr>
<tr>
<td></td>
<td>sort, for_each, merge</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Streaming layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pipelined sorting, zero-I/O scanning</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Block management layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>typed block, block manager, buffered streams, block prefetcher, buffered block writer</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Asynchronous I/O primitives layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>files, I/O requests, disk queues, completion handlers</td>
</tr>
</tbody>
</table>

MCSTL

OpenMP, Atomic Ops

Parallel STL Algorithms

Serial STL Algorithms

STL Interface
Problem Instances

Benchmark instances for NP-hard problems

- TSP
- Steiner-Tree
- SAT
- set covering
- graph partitioning
- ...

have proved essential for development of practical algorithms

Strange: much less real world instances for polynomial problems
(MST, shortest path, max flow, matching...)
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 × 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)  

**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,…

**Communication Networks:** mobile, P2P, cloud, selfish users,…
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 inputs as possible
- its fine if you are better just on some (important) inputs
Not Here but Important

- describing the setup
- 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, \(|V|, |E|, \ldots\) or problem instances from applications)

Usually there is not a lack but an abundance of data \(\neq\) many other sciences
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 $\implies$ 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

$\mapsto$ new input instances