On Experimental Algorithmics
Research Methods in Computer Science 2010
Pekka Kilpeläinen
University of Eastern Finland
School of Computing

Good Experimental Research on Algorithms?

Why Experimentation?
- It is difficult to produce generally valid results by experiments, and impossible to obtain them with certainty. (Cf. testing)
- On the other hand,
  - theory alone does not give full picture of efficiency
  - experiments may verify theoretical results and raise new hypotheses
  - experimenting is an important route to applications
- \( \rightarrow \) workshops and journals on experimentation and algorithm engineering

Principles of good experimentation
1. Newsworthy experiments
2. Connection to literature
3. Arrangements that support general conclusions
4. Efficient and effective experimental designs
5. Reasonably efficient implementations
6. Reproducibility and comparability
7. Comprehensive reporting
8. Justified conclusions and explanations
9. Informative presentation of results

Basis of the Principles
- **Goal:** To produce
  - new and interesting/useful knowledge in such ways that
  - the researcher is confident of correctness and presented so that
  - the reader can verify correctness
- That is, aim at which
  - is accepted for publications by reviewers and
  - is of long-term value

1. Newsworthy experiments
- Results to-be-published should be new and interesting
  - algorithms (at least potentially) useful
- What if the method appears to be dominated, in all cases less efficient than another one?
- **Typical mistake:** Insufficient background research. *Not good!*
Negative results?

- Positive results preferable
- Results on a dominated algorithms acceptable, if
  - they are surprising, say, the method in wide use
  - they provide some methodological insight (say, on some new control or data structures)
  - they have not been reported before
- Call for thorough justification, and explanation
- Better if published together with positive results

Risk: Wasting time on wrong questions

- Try to test systematically, using selection of different cases
  - test cases – sample from infinite population (cf. estimating people’s avg length)
- Do not run full experiments too early
  - explore first, before tuning implementations and completing experimental design
  - > What to measure? How? On which instances?
  “Think before you compute”

Johnson’s Approach

1. Spend ~ 1/2 of time generating lots of test data, looking for patterns and anomalies
2. Fix research questions; Finalize implementation; Design and carry out comprehensive experiments;
3. Analyze data; Go back to 2 if failed to answer questions, or new ones raised

Risk: Unnecessary tuning;
Goal is to answer research questions, only

Eventually, STOP the experiments!

Factors of Newsworthyness

- The generality, relevance, and credibility of results
- Why would the results be more than specifics of experimental setting and environment?
  - For example, counting of suitable basic operations or profiling (vs. plain running time)
- Can we get more general view, say, of asymptotic behavior, or precision of approximated algorithms?

Algorithm vs. Implementation

- Experiments provide data of implementations only
- Questions for getting more general info about algorithms:
  1. Effect of implementation details, parameters, heuristics and data structures on execution time?
  2. Scalability wrt instance size? Dependency on instance structure?
  3. Explaining execution times by operation counts? Bottlenecks? Deviations from theoretical worst-case analysis?

Algorithm vs. Implementation (2)

4. Effect of processor architecture? Predictability of execution times?
5. Comparison of execution times vs. top competitor? Effect of above factors on this?
6. If a new class of instances introduced, does it cause significant changes in behavior of previously studied algorithms?

Risk: Ending up investigating properties of instance classes (~ experimental math) instead of algorithm behavior (~ CS)
2. Tie your work to the literature

- First study prior literature
  - to avoid unnecessary experiments
  - to find interesting questions
  - to find baseline methods for comparison
- Try to obtain the code of previous implementations, or develop own comparable implementation
  - Baseline for a totally new solution?
  - More on comparability later

3. Settings that support general conclusions

- Two types of test instances: real[istic] and randomly generated
  - with controlled randomness - instance structure should be realistic
- Generated instances support scalability studies
- Real cases suggest applicability to real applications
  - Combinations of the two recommended
  - relevance and precision of execution times close to milliseconds?

4. Efficient and Effective Experimental Designs

- Minimize effects of variation. For example, use the same random instance for each algorithm
- Manage your data well
  - arrange in directories/folders, use README files, ...
  - implement self-documenting programs that write
    - what was measured, by which algorithm & version, using which processor, date, input instance, parameters, ...
- May need data after years, when it is difficult to interpret and impossible to re-generate

5. Reasonably efficient implementations

- Why? To support conclusions based on execution times
  - Claiming comparative performance with others if our implementation was just made efficient is a suspicious argument
- Note: Reasonably efficient, concentrating implementation effort to bottle-necks
  - Avoid unnecessary code tuning

6. Reproducibility and Comparability

- Should be able to check correctness of the results by repeating experiments
  - detailed measurements will differ, but conclusions should remain the same
  - the researcher shall become confident with the validity of his/her conclusions
  - all relevant factors must be reported: algorithms, instances, environment (processor, RAM, OS, language, compiler; their versions and optimizing params)
- Good to make implementations and instances available (on the Web)

On Comparability

- Further research happens most likely in a different environment
- How to compare execution times (if not possible to repeat the previous experiments)?
- Using benchmark programs to calibrate machine speeds
  - code that carries out operations typical to the algorithms in question
  - precision of normalized times within factor 2 (only!)
7. Comprehensive Reporting

- Must interpret and simplify data, but **within limits**
- Data behind conclusions must be given
  - representative samples of different instance classes
  - Tables, at least in Appendix (or on the Web)
  - Distributions behind averages?

**Risk:** presenting averages with too much accuracy  
(-> false conclusions)

---

Comprehensive reporting (2)

- Any anomalies must be remarked
  - and explained (or stated unexplained)
  - unremarked anomalies potentially caused by typos, or bugs  
  (-> risk of being rejected)
- Report full execution times, not just of the own method we’ve measuring
  - > relevance of our method in context
  - > need of resources for those who will repeat the experiments

8. Justified conclusions and Explanations

- Experiments should produce and support conclusions
- **Typical mistake:** Uninterpreted data
  - summarize any patterns
  - try to pose more general conjectures
- **Typical mistake:** Unsupported conclusions
  - e.g. claims of asymptotic behaviour from too small instances
- Profiling or instrumenting code with counters useful for understanding and explaining execution times

---

9. Informative Presentation of Data

- See also Sanders, and Ch. 6 (Graphs, figures, and Tables) in Zobel
- Presentation shall support conclusions
  - visualization useful both for observing and demonstrating
  - emphasize observable trends and differences
- **Typical mistake:** Tables without graphics
  (Summary? Conclusions?)
  - and poorly organized tables: Order of rows/columns; derived columns like perf. ratios; Meaning and units of items

---

Examples of my experimentation

- **Topic:** Efficiency of existing vs. optimal UPA test in XML Schema processors
- Examples of scripts for
  - generating test instances
  - running test-bed programs on instances
  - plotting collected data
- Mainly “quick-and-dirty” solutions
- **Main message:**
  - automated use of tools helps a lot
  - hopefully some inspiration from details, too