Experiments: Test Design

Hannes Moser

Fakultät für Mathematik und Informatik
Universität Jena
DFG project ITKO (NI 369/5)

GI-Dagstuhl Research Seminar 06362 “Algorithm Engineering”

Overview

- Introductory examples: Why experimentation?
- The experimentation process
- Measures
- Algorithm factors

Theoretical Superiority does not always pay off

Experimental comparison of algorithms for Minimum Spanning Tree [Moret and Shapiro, 1994]:

Theoretical Superiority does not always pay off

Experimental comparison of algorithms for Minimum Spanning Tree [Moret and Shapiro, 1994]:

Kruskal	$O(\|E\| \cdot \log \|V\|)$
Prim	$O(\|E\| \cdot \log \|V\|)$
Cheriton and Tarjan	$O(\|E\| \cdot \log \log \|V\|)$
Fredman and Tarjan	$O(\|E\| \cdot \beta(\|E\|, \|V\|))$
Gabow et al.	$O(\|E\| \cdot \log \beta(\|E\|, \|V\|))$

Theoretical Superiority does not always pay off

Experimental comparison of algorithms for Minimum Spanning Tree [Moret and Shapiro, 1994]:

Kruskal	$O(\|E\| \cdot \log \|V\|)$
Prim	$O(\|E\| \cdot \log \|V\|)$
Cheriton and Tarjan	$O(\|E\| \cdot \log \log \|V\|)$
Fredman and Tarjan	$O(\|E\| \cdot \beta(\|E\|, \|V\|))$
Gabow et al.	$O(\|E\| \cdot \log \beta(\|E\|, \|V\|))$

Fastest algorithm in practice: Prim's

Rare Worst Case

The worst-case behavior is sometimes restricted to a very small subset of problem instances.

Rare Worst Case

The worst-case behavior is sometimes restricted to a very small subset of problem instances.

Example

Simplex Method for linear programming:

- Worst Case: exponential running time
- "Practical Case": (low-degree) polynomial running time

Huge Constants

Many theoretically good algorithms are practically irrelevant due to hidden constants.

Huge Constants

Many theoretically good algorithms are practically irrelevant due to hidden constants.

- $O(n^3)$ minor testing (Robertson and Seymour).
- $O(n)$ test for treewidth (Bodlaender).

Huge Constants

Many theoretically good algorithms are practically irrelevant due to hidden constants.

- $O(n^3)$ minor testing (Robertson and Seymour).
- $O(n)$ test for treewidth (Bodlaender).

The "big Oh" notation facilitates the design of unpractical algorithms.

Difficult Analysis

Some algorithms are extremely difficult to analyze mathematically.

Difficult Analysis

Some algorithms are extremely difficult to analyze mathematically.

Examples

- Simulated Annealing
- Genetic Algorithms
- Union-Find with path compression

Other Drawbacks

- Theoretical analysis tends to simplify problems.
- Theoretical analysis tends to make unrealistic assumptions.
- Important tricks in practice are beyond the reach of mathematical analysis.
- ...

Observations

- Often a huge gap between theory and practice.
- A major progress concerning mathematical tools is not likely to occur soon.
- No well-established methodology for experimental research (like e.g. in physics).
- ...

Observations

- Often a huge gap between theory and practice.
- A major progress concerning mathematical tools is not likely to occur soon.
- No well-established methodology for experimental research (like e.g. in physics).
- ...

Demand

We need an empirical science of algorithms!

Observations

- Often a huge gap between theory and practice.
- A major progress concerning mathematical tools is not likely to occur soon.
- No well-established methodology for experimental research (like e.g. in physics).
- ...

Demand

We need an empirical science of algorithms!

Such an empirical science is meant to amend theoretical analysis, not to replace it!

The Experimentation Process

1. Define the goals of the experiment.
2. Choose the measures of performance and factors to explore.
3. Implement and execute the experiment.
4. Analyze the data and draw conclusions.
5. Report the experimental results.

The Experimentation Process

1. Define the goals of the experiment.
2. Choose the measures of performance and factors to explore.
3. Implement and execute the experiment.
4. Analyze the data and draw conclusions.
5. Report the experimental results.

In this talk: 1. and 2.

Experimentation Objectives: Examples (1)

- Show the superiority of an algorithm compared with the existing ones.
- Show the relevance of an algorithm for an application.
- Compare the performance of existing algorithms.
- Do algorithm engineering on existing algorithms.
- Development of tools.

Experimentation Objectives: Examples (2)

- Show that an existing algorithm performs surprisingly bad.
- Analyze an algorithm or a problem to better understand it.
- Support or reject conjectures.
- Investigate and refine models and optimization criteria.

Standard Measures and their Drawbacks

Standard Measures

- running time
- memory requirements
- value and/or quality of the solution

Standard Measures and their Drawbacks

Standard Measures

- running time
- memory requirements
- value and/or quality of the solution

Drawbacks

- Measures highly depend on
  - programming language, compiler, computer environment, ...
  - implementation details and the skill of the programmer,
  - choice of the test instances.
- Aggregate measures

Other measures (1)

- extended running time measures
- number of iterations
- number of calls to a crucial subroutine
- major subtasks
- mems (memory references)
- number of comparisons

Number of comparisons

Example: Sorting Algorithms

Number of comparisons depending on input size.

Elements	50	100	200	300	400
Selection Sort	1225	4950	19900	44850	79800
Exchange Sort	1410	5335	20300	45650	79866
Insertion Sort	1391	5399	20473	44449	78779
Quicksort	399	990	1954	3384	5066

(Source: http://atschool.eduweb.co.uk/mbaker/sorts.html)

Other measures (2)

- number of data moves
- number of assignments
- data structure updates
- nodes in a search tree
- representative operation counts
- (asymptotic) bottleneck operations
- virtual running time

Representative Operation Counts (1)

[Ahuja, Magnati, Orlin, 1993]

Assumption

Every line of code needs $O(1)$ time.

Representative Operation Counts (1)

[Ahuja, Magnati, Orlin, 1993]

Assumption

Every line of code needs $O(1)$ time.

Observation

Normally, it suffices to look at a very small number of lines of code.

Representative Operation Counts (1)

[Ahuja, Magnati, Orlin, 1993]

Assumption

Every line of code needs $O(1)$ time.

Observation

Normally, it suffices to look at a very small number of lines of code.

Idea

For each "important" line of code we count the number of times it is executed.

Representative Operation Counts (1)

[Ahuja, Magnati, Orlin, 1993]

Assumption

Every line of code needs $O(1)$ time.

Observation

Normally, it suffices to look at a very small number of lines of code.

Idea

For each "important" line of code we count the number of times it is executed.

Example 1

for $i := 1$ to 3 **do**
begin
$A(i) := A(i) + 1;$
$B(i) := B(i) + 2;$
$C(i) := C(i) + 3;$
end

Representative Operation Counts (2)

Example 2: Quicksort

```
1. function quicksort(q)
2.         var list less, greater;
3.         if length(q) ≤ 1 then return q;
4.         select a pivot value pivot from q;
5.         for each x in q except pivot do
6.                 if x < pivot then
7.                         add x to a list less;
8.                 else
9.                         add x to greater;
10.        return concat(quicksort(less), pivot, quicksort(greater));
```

Representative Operation Counts (3)

Advantages

- Representative operation counts are relatively easy to determine.
- There are often several possible choices.
- The representative set is typically quite small (1 to 4 in many cases).
- Representative operation counts are easier to reproduce.

Representative Operation Counts (3)

Advantages

- Representative operation counts are relatively easy to determine.
- There are often several possible choices.
- The representative set is typically quite small (1 to 4 in many cases).
- Representative operation counts are easier to reproduce.

Next Step

Find representative operations that consume a significant percentage of the execution time.

Asymptotic Bottleneck Operations (1)

Asymptotic Bottleneck Operation

An operation is an asymptotic bottleneck operation if its share in the computational time becomes larger as the problem size increases.

Asymptotic Bottleneck Operations (1)

Asymptotic Bottleneck Operation

An operation is an asymptotic bottleneck operation if its share in the computational time becomes larger as the problem size increases.

Advantages

- Determine the running times of an algorithm for sufficiently large problem sizes.
- Relatively easy to find experimentally.
- Help to better estimate the growth in the computational time.

Asymptotic Bottleneck Operations (1)

Asymptotic Bottleneck Operation

An operation is an asymptotic bottleneck operation if its share in the computational time becomes larger as the problem size increases.

Advantages

- Determine the running times of an algorithm for sufficiently large problem sizes.
- Relatively easy to find experimentally.
- Help to better estimate the growth in the computational time.

Question

How can asymptotic bottleneck operations be used to compute the algorithm's performance?

Asymptotic Bottleneck Operations (2)

Assumption

For sufficiently large problem sizes the running time of an algorithms depends mainly on its asymptotic bottleneck operations.

Asymptotic Bottleneck Operations (2)

Assumption

For sufficiently large problem sizes the running time of an algorithms depends mainly on its asymptotic bottleneck operations.

Idea

By estimating the growth rate of all asymptotic bottleneck operations, we get a good estimate of the algorithm's performance (better than the sole running time).

Asymptotic Bottleneck Operations (2)

Assumption

For sufficiently large problem sizes the running time of an algorithms depends mainly on its asymptotic bottleneck operations.

Idea

By estimating the growth rate of all asymptotic bottleneck operations, we get a good estimate of the algorithm's performance (better than the sole running time).

Question

Given such an estimate, how can we extrapolate the expected running time behavior on a given machine?

Virtual Running Time (1)

Assumption

The running time can be estimated as a linear function of the (asymptotic) bottleneck operation counts.

Virtual Running Time (1)

Assumption

The running time can be estimated as a linear function of the (asymptotic) bottleneck operation counts.

Virtual Running Time

A linear function of the (asymptotic) bottleneck operation counts, that gives the best possible estimate of the running time.

Virtual Running Time (1)

Assumption

The running time can be estimated as a linear function of the (asymptotic) bottleneck operation counts.

Virtual Running Time

A linear function of the (asymptotic) bottleneck operation counts, that gives the best possible estimate of the running time.

Approach

1. Determine running times and operation counts for various problem sizes.
2. Use (multiple) regression analysis to minimize the difference between linear estimate and actual running time.

Virtual Running Time (2)

Observations

- Virtual running time is a good estimate of the running time.
  [Ahuja, Magnati, Orlin, 1993]
- Makes some simplifying assumptions.
- Approach should be tested on more problems.
- Approach could possibly be extended.
- Automatic detection of representative operations?

How to find good measures

- Exploratory experimentation: How does the running time/space depend on ...
  - implementation details
  - parameter settings
  - heuristics
  - data structure choices
  - instance size, instance structure
  - machine architecture

How to find good measures

- Exploratory experimentation: How does the running time/space depend on ...
  - implementation details
  - parameter settings
  - heuristics
  - data structure choices
  - instance size, instance structure
  - machine architecture
- Is there a correlation between running time/space and the count of some operation?
- Try to find bottleneck operations.

How to find good measures

- Exploratory experimentation: How does the running time/space depend on ...
  - implementation details
  - parameter settings
  - heuristics
  - data structure choices
  - instance size, instance structure
  - machine architecture
- Is there a correlation between running time/space and the count of some operation?
- Try to find bottleneck operations.
- Use existing tools.
- Use profilers!

Algorithm Factors (1)

Measures are affected by a set of factors

- algorithm
- algorithm parameters
- test instances
- computing environment/machine architecture
- heuristics
- ...

Algorithm Factors (2)

Some factors will be altered in the experiment.

- In which range will they be altered?
- How many different values will be taken into account?
- How many trials for each value?

Algorithm Factors (2)

Some factors will be altered in the experiment.

- In which range will they be altered?
- How many different values will be taken into account?
- How many trials for each value?

Some factors will be fixed at some level.

- Why can they be fixed?
- At which level should they be fixed?

Algorithm Factors (2)

Some factors will be altered in the experiment.

- In which range will they be altered?
- How many different values will be taken into account?
- How many trials for each value?

Some factors will be fixed at some level.

- Why can they be fixed?
- At which level should they be fixed?

Some factors will be ignored.

- Why can they be ignored?
- Should the factor be randomized?

Conclusion/Final Remarks

- Theoretical Analysis should be amended by experimental research.
- Many approaches in literature are very specialized or too general.
- We need to establish a methodology for experimental research ("empirical science of algorithms").

Thank you!