Computational Optimization
ISE 407

Lecture 11

Dr. Ted Ralphs

Reading for This Lecture

- Assessing the Effectiveness of (Parallel) Branch-and-Bound Algorithms
- A Theoretician's Guide to the Experimental Analysis of Algorithms
- Statistical Analysis of Computational Tests of Algorithms and Heuristics

Analysis with Curated Test Sets

- It is common in optimization to do testing with relatively small curated test sets.
- Such test sets cannot usually be considered to be a random smaple from a larger class.
- The instances may vary substantially from each other in many ways (sometimes by design).
- The difficulty of the instances may vary widely.
- It is thus difficult to generalize results beyond the test set.
- Nevertheless, these test sets often represent problems we care about and we would like to compare performance of different algorithms on them.
- Such analysis typically involves pairwise comparison of performance on individual instances.

Summary Statistics

- It is common for comparison to be done using summary statistics across a test set.
- Summary statistics may be useful as a first cut, but they hide information useful for comparison.
- They can also result in wildly incorrect conclusions due to outliers.
- The best analyses allow multi-faceted conclusions: "Algorithm A is better on small instances, while algorithm B is better on larger..."
- Because of the relatively small test sets, the results must be put in the proper context.
- It may be difficult to draw "statistically valid" conclusions.

Visualization

- There are a number of visualization techniques that can allow for analysis that is more refined than that allowed by summary statistics.
- These are based on the same idea of constructing empirical distribution functions, but for different, more restricted distributions.
- Existing methods of visualizing algorithmic effectiveness data.
  - Performance profiles
  - Baseline profiles
  - Cumulative profiles
  - Pair plots
  - Interactive plots

Performance Profiles

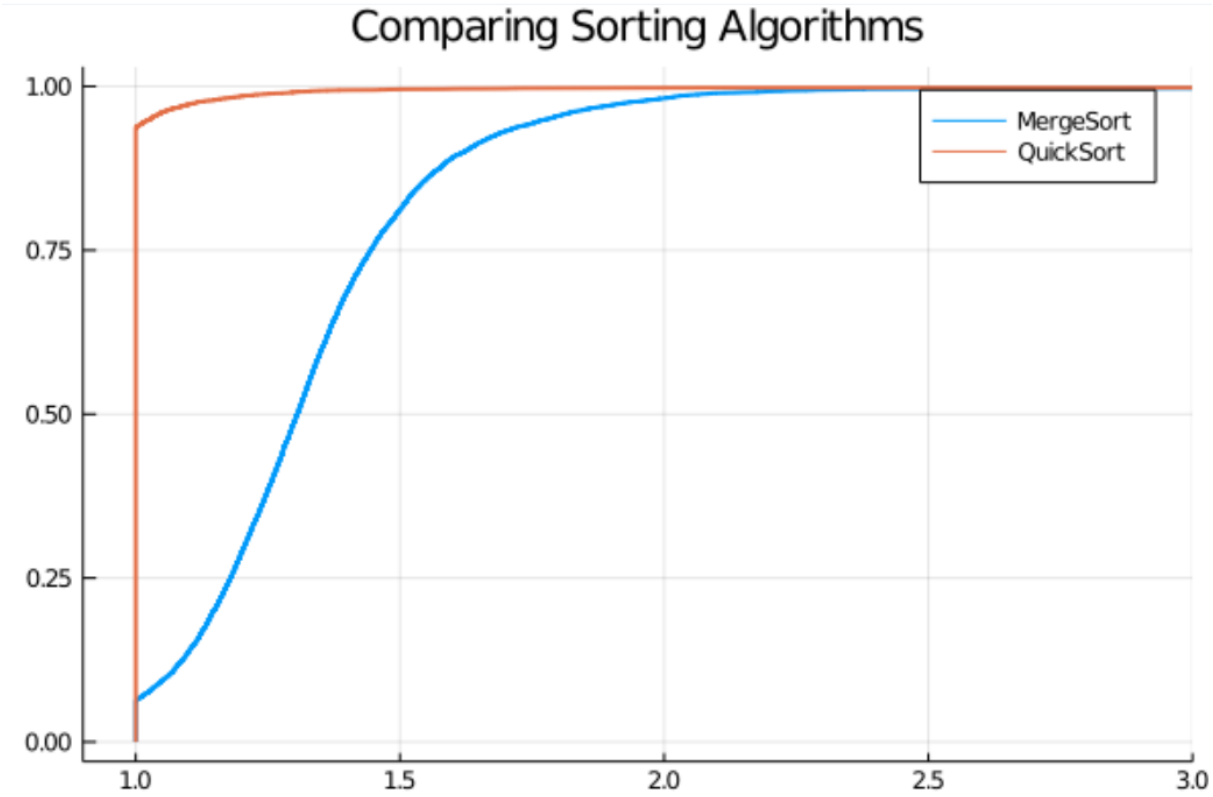
- Performance profiles provide a visual summary of how algorithms compare on a measure of efficiency across a test set.
- For algorithm a and instance i, we compute a performance ratio

$$r_{ai} = \frac{t_{ai}}{\min_{a' \in A} t_{a'i}}.$$

- Instances that fail to solve (e.g., time out) are given a ratio of ∞.
- We then plot the empirical cumulative distribution function of the performance ratio for each algorithm.
- The idea of is to create a scale-invariant way of comparing across a set of instances with disparate running times.
- This allows for test sets with instances of different sizes and difficulties.
- This method has some drawbacks
  - It can end up giving too much weight to "easy" instances.
  - The "virtual best" is also not necessarily a realistic baseline.
  - There is no additional information on instances that fail.

Performance Profile

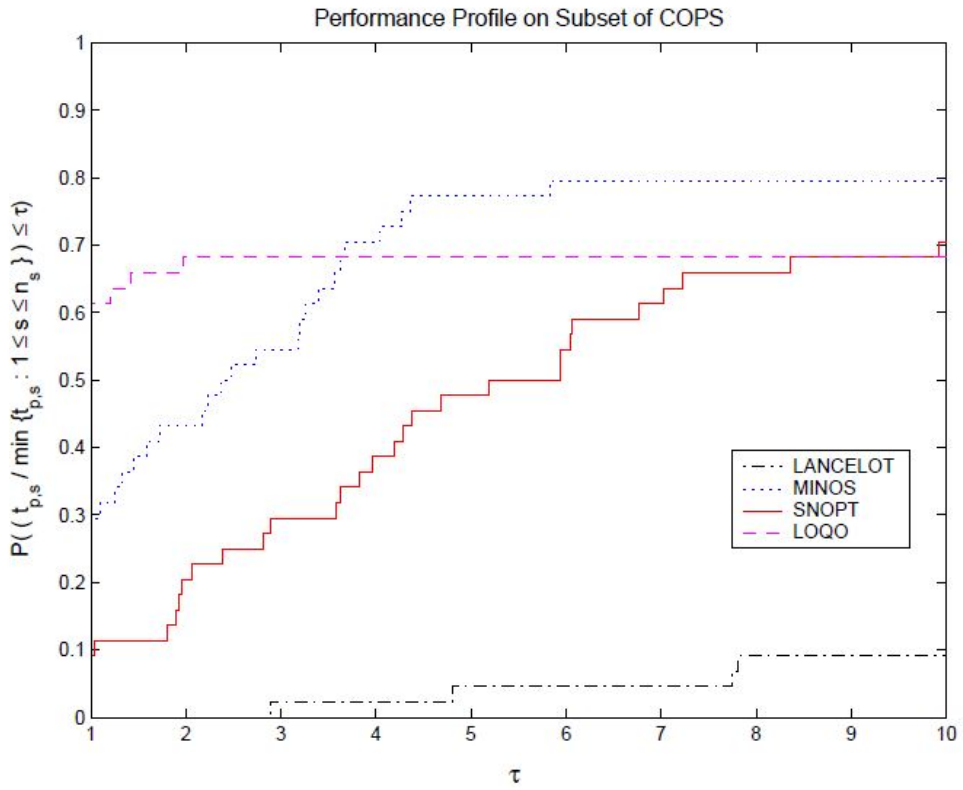
```
julia> rng =  MersenneTwister(12345);
julia> q = @benchmark sort(x, alg=QuickSort) evals=10 samples=10000 setup=(x=rand(rng, 1000)) seconds=10000
julia> rng =  MersenneTwister(12345);
julia> m = @benchmark sort(x, alg=MergeSort) evals=10 samples=10000 setup=(x=rand(rng, 1000)) seconds=10000
julia> mm = sort(m.times ./ min.(q.times, m.times))
julia> qq = sort(q.times ./ min.(q.times, m.times))
julia> plot(mm, pc(mm), l=2, label="MergeSort")
julia> plot(qq, pc(qq), l=2, label="QuickSort")
```



The above code assumes the times are not sorted (use my fork of `BenchmarkTools`)!

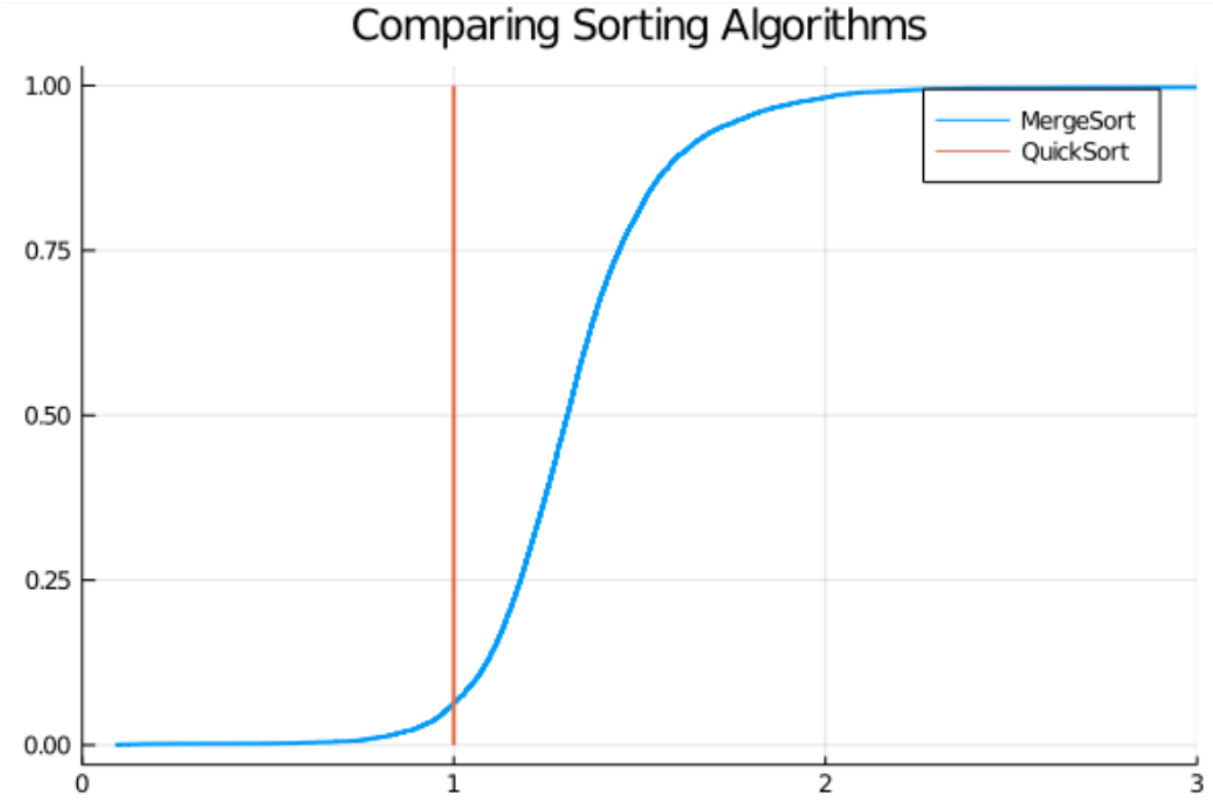
Performance Profile on a Smaller Test Set

- Below is a profile that is typical of what would be produced with a curated test set.
- There is not really enough data to estimate any true underlying distribution.
- Still, it may be possible to compare performance on the test set itself.



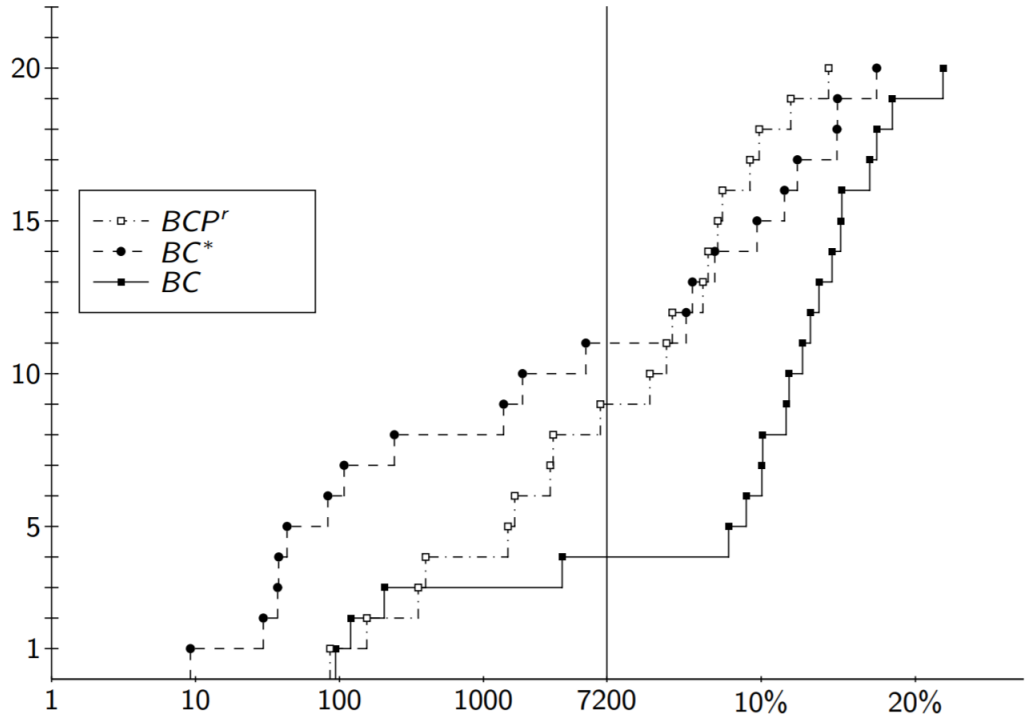
Baseline Profiles

- Note that we could easily extend the concept of performance profile to ratios using any baseline.
- If there is a natural baseline for comparison (such as the best previously existing algorithm), this may make a better baseline.
- Using "virtual best" can create misleading results when there are more than two solvers.



Cumulative Profiles

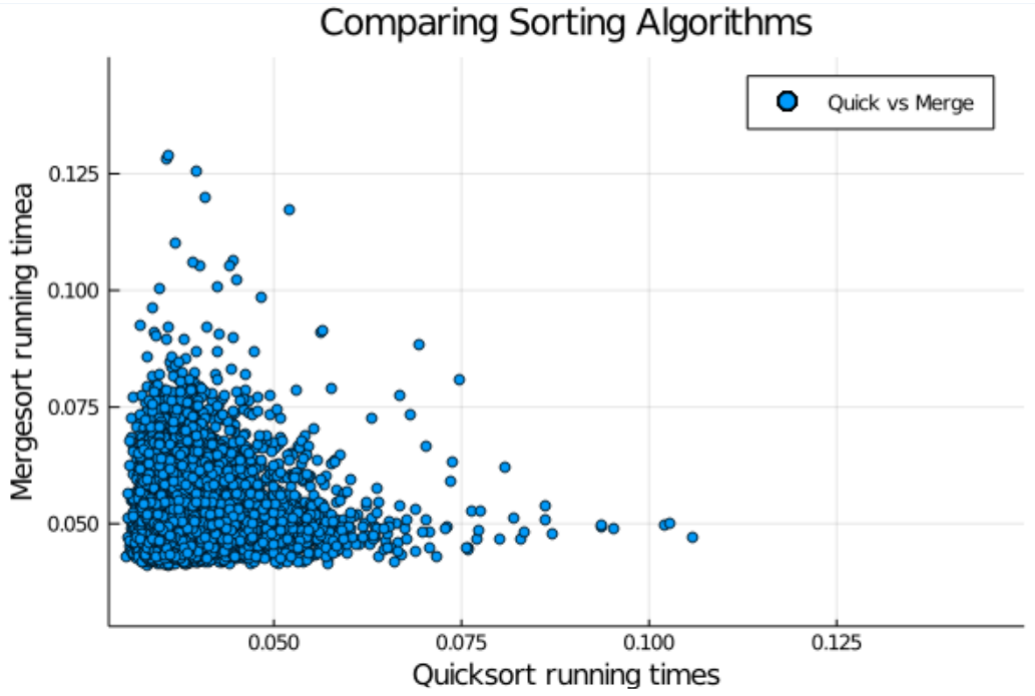
- Cumulative profiles plot the empirical cumulative distribution function of the resource consumption, as before.
- This is only an estimate of the distirbution on this test set, not the "true" distribution on some underlying larger class.
- We can be extend beyond the time limit used in the actual computation by use of an appropriate "measure of progress."



Pair Plots

```
julia> scatter(q.times*1e-6, m.times*1e-6, xlabel="Quicksort running times",
ylabel="Mergesort running timea", label="Quick vs Merge", title="Comparing
Sorting Algorithms", xlim=(0.028, .15), ylim=(0.028,.15))
```

- Pair plots display ordered pairs of a given measure of efficiency for two algorithms.
- Each plotted point represents one instance.
- Algorithms are compared based on the number of plotted points above/below the center line.



Interactive Plots

- The gold standard for data visualization is an interactive plot that allows readers/users to display algorithmic data in an interactive way.
- This is becoming possible with new visualizations sites/packages.
  - Plot.ly
  - Bokeh
- Be creative and invent your own data visualization!

Additional Challenges

- Accounting for variability, non-determinism, and stochasticity
- Comparison to existing algorithms
- Comparing algorithms for difficult problems
- Comparing to existing algorithms
- Drawing valid conclusions
- Ensuring replicability

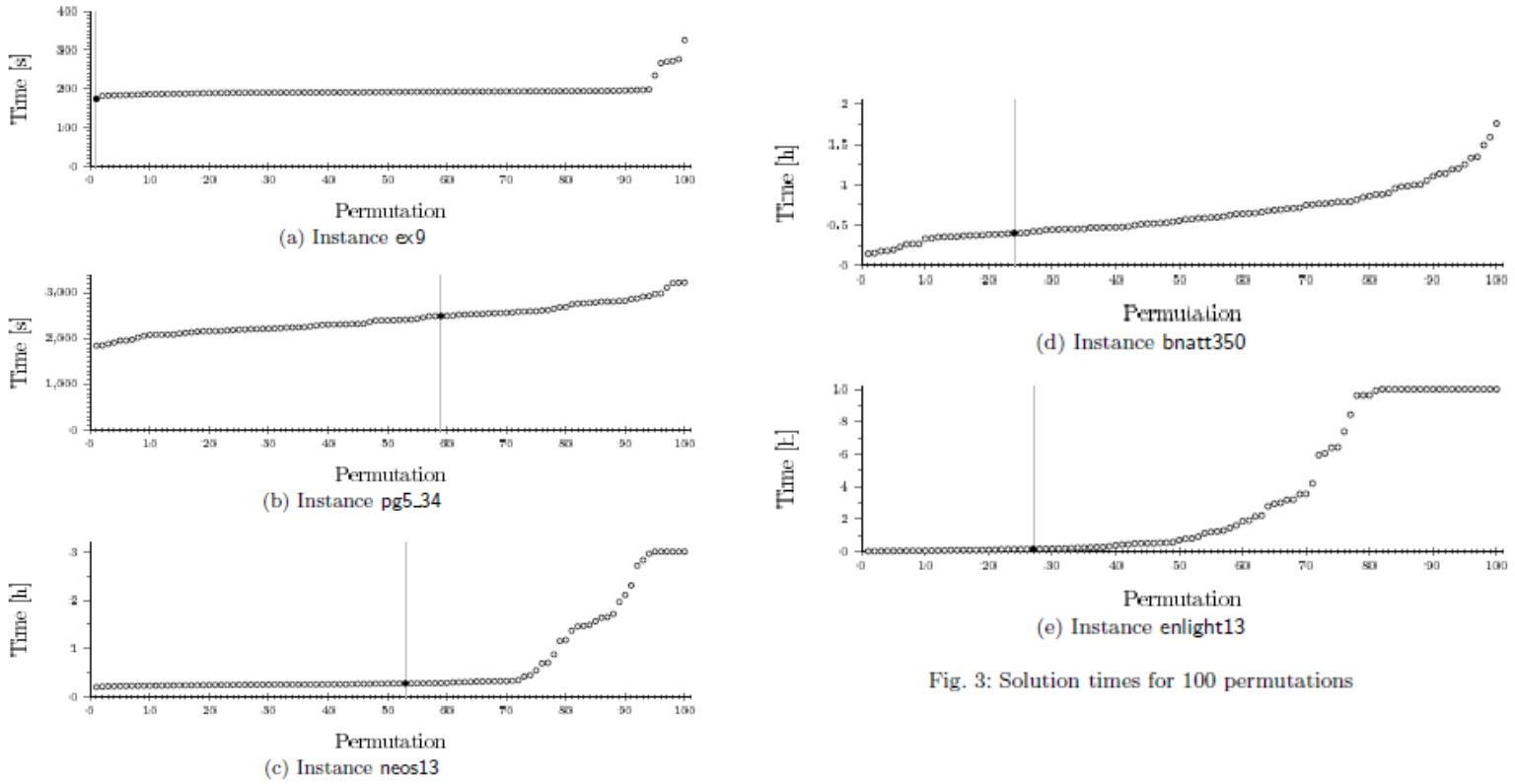
Accounting for Variability

- In empirical analysis, we must take account of the fact that running times are inherently variable for multiple reasons.
- If we are measuring wallclock time, times may vary substantially, even for identical executions.
- It helps to control the environment using tools, such as `cpuset` that reserve resources for use only for a specific purpose.
- In the case of parallel processing, stochasticity may also arise due to non-determinism in the case of asynchronous implementations (more on this later).
- Even sequential algorithms can be non-deterministic due to randomization.
- In such case, multiple evaluations may be used to estimate the affect of this randomness.
- If necessary, statistical analysis may be used to analyze the results, but this is beyond the scope of this course.

Performance Variability on Difficult Problems

- Algorithms for problems in the class NP-complete are generally much more difficult to assess and display a great degree of unpredictability.
- These algorithms are more susceptible to random fluctuations from apparently incidental environmental differences.
- Minor differences in parameter settings or input format can lead to wild fluctuations in the measured performance.
- Algorithms for polynomially solvable problems tend to be more predictable, though even these can exhibit large fluctuations in behavior in some cases.
- It is important to first understand the features of the problem class of interest in selecting the proper approach to analysis.

Performance Variability



(a) Instance ex9

(b) Instance pg5_34

(c) Instance neos13

(d) Instance bnatt350

(e) Instance enlight13

Fig. 3: Solution times for 100 permutations

Source: `https://opus4.kobv.de/opus4-zib/files/1295/miplib5.pdf`

Comparing Efficiency on Difficult Instances

- A particularly challenging aspect of empirical testing is comparing algorithms for difficult classes of problems.
- When one or more algorithms do not terminate within a reasonable time, we need something beyond just a measure of efficiency.
- One alternative is to simply report the fraction of instances successfully solved as another statistic.
- Another is to use a *measure of progress* or a *measure of work*.

Measures of Work/Progress

Definition 1. *A* measure of progress *is an estimate or proxy for the fraction of a full computation performed by an algorithm, given a fixed bundle of resources.*

- Measures of progress may be very difficult to derive in some cases.
- A *measure of work* is a direct measure of the work that has been performed in the computation and is much easier to derive.
- A measure of work may be a proxy for a measure of progress, but not always.

Difficulty of Measuring Progress

- For a measure of work to serve as a measure of progress, we need to know the total amount of work expected to solve the problems.
- In the case of NP-complete problems, this depends highly on the "guided luck" we discussed earlier.
- The "luck" involves avoiding dead ends and this is what sophisticated algorithms attempt to do.
- The "dead ends" contribute to work, but not necessarily "progress."
- Execution may vary dramatically based on seemingly inconsequential perturbations to the algorithm.
- In general, one can only expect to derive reliable such measures in cases where the computation is somewhat predictable.
- Unfortunately, these are the "easy" cases.

What is a "Fair" Comparison?

- Test sets may create bias if not chosen properly.
- The testing platforms can also create bias if hardware favors a particular implementation.
- Hold as many things constant as possible.
- Seemingly inconsequential or irrelevant differences in implementation can cloud results.
  - Underlying data structures.
  - Memory allocation patterns and cache effects.
  - Implementation of low-level operations.
  - Compiler differences (optimization level).
  - Operating system.

Comparing to Other Codes

- Rigorously comparing to an algorithm implemented by someone else can be difficult.
- It is nearly impossible to fairly compare to results reported in the literature.
- Ideally, however, obtaining the source code for any alternative implementations is best.
- In some cases, it may be possible to re-implement an algorithm from the literature, but you are unlikely to do this fairly.
- If all else fails, scaling running times from previous computational experiments may give some idea.

Drawing Valid Conclusions

- If nothing else, be sure to draw only truly valid conclusions from your results.
- Doing so requires first and foremost that your code is (nearly) error free.
- You must also provide some honest answers to soul-searching questions.
  - How general are the results truly?
  - Do they generalize
    - to other platforms,
    - To other instances,
    - To other implementations,
    - ...
  - Has the test set been hand-massaged in any way?
  - How fair is the comparison to other algorithms?
- In almost all cases, the degree to which the results can be generalized is very limited and it is important to state this.

Replicability and Generalizability

- Ultimately, your results will only be important if others can replicate them.
- Allowing replication involves multiple good practices.
  - Reporting all details of experiments.
  - Tracking changes carefully.
  - Describing important details.
  - Providing versioned open source code.

Tracking Changes

- It is extremely important to be diligent in tracking changes while developing implementations.
- There are multiple reasons for this.
- Most importantly, it makes debugging performance issues much easier!
- But it also makes it possible to archive precise version that were used for particular experiments.
- Even if further development has happened in the meantime, published results should always be possible to replicate.
- This can be done with version control software, such as `git`.
- We'll discuss this more later.

Good Publication Practices

- To make your mark and do good research, the final step is to publish well.
  - Don't fail to reveal important details
  - Publish source if at all possible.
  - Draw only valid conclusions.
  - Report all relevant (and even irrelevant) information about the experiments!
- This is important both for your own integrity and that of the scientific establishment.

Empirical versus Theoretical Analysis

- For sequential algorithms, asymptotic analysis is often good enough for choosing between algorithms.
- It is less ideal with respect to tuning of implementational details.
- For parallel algorithms, asymptotic analysis is far more problematic.
- The details not captured by the model of computation can matter much more.
- There is an additional dimension on which we must compare algorithms: scalability.