Can a Competition Be Scientific?

John Hooker
Carnegie Mellon University
September 2007

- Why algorithmic competition is unscientific.
- What to do about it.

Why competition is unscientific.

- The results depend on extraneous factors, such as…

- The results depend on extraneous factors, such as…
  - Coding skill.

- The results depend on extraneous factors, such as…
  - Coding skill.
    - Competitiveness may equalize coding skill among best entries.
    - But this is wasteful.

- The results depend on extraneous factors, such as…
  - Parameter tuning.

- The results depend on extraneous factors, such as…
  - Parameter tuning.
    - “Vanilla” code is undefined.
    - Parameters are problem dependent.

- The results depend on the choice of test problems.

- Random problem instances are unrealistic.

- Random problem instances are unrealistic.
  - Real problems are structured.
  - Choice of distribution may favor certain algorithms.

- A real problem set may be unrepresentative.

- A real problem set may be unrepresentative.
  - Selection may favor certain algorithms.
  - Many important problem instances are proprietary.
  - Benchmark problems tend to be instances on which previous algorithms have performed well.
  - The problem instances design the algorithms.
  - What does "representative" mean?

- Competitions tell us which codes are faster, but not why.

- Competitions tell us which codes are faster, but not why.
  - Fast codes are full of tricks.
  - What is responsible for the code’s performance?
  - The real testing occurs while tinkering to find the right tricks.

What to do about it.

- Controlled experimentation.

- Controlled experimentation.
  - Get rid of benchmark problems.
  - Factorial design.
  - Control for factors that may influence performance.
    - Other characteristics random.
  - Cautionary example – phase transition.

- Ultimate aim – an empirical *theory* that predicts algorithmic performance.

- Ultimate aim – an empirical *theory* that predicts algorithmic performance.
  - Empirical ≠ nontheoretical

- Ultimate aim – an empirical *theory* that predicts algorithmic performance.
  - Empirical ≠ nontheoretical
  - Example: NP-completeness theory.
    - It is useful and explanatory only to the extent that it is viewed as an empirical theory.
      - NP is NP-complete.
      - P + TSP is NP-complete.
      - P + TSP instances to which SAT is reduced is NP-complete.

- Ultimate aim – an empirical *theory* that predicts algorithmic performance.
  - Example: Branching rules for SAT

Markov chain model:

What happens in a unit resolution step.

Each time a variable is fixed:

$$
\begin{aligned}
&Pr(C_i \text{ eliminated}) = \frac{k}{2n},\\
&Pr(C_i \text{ reduced to } k-1 \text{ literals}) = \frac{k}{2n},\\
&Pr(C_i \text{ unchanged}) = 1 - \frac{k}{n}
\end{aligned}
$$

C_i = clause i k = #literals in C_i n = # variables

Resulting transition matrix (state = # literals in clause):

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & \dots \\ 0 & 1 & 0 & 0 & 0 & \\ \frac{2}{2n} & \frac{2}{2n} & 1-\frac{2}{n} & 0 & 0 & \\ \frac{3}{2n} & 0 & \frac{3}{2n} & 1-\frac{3}{n} & 0 & \\ \frac{4}{2n} & 0 & 0 & \frac{4}{2n} & 1-\frac{4}{n} & \\ \vdots & & & & & \end{bmatrix}$$

This model predicts relative performance of several branching rules.

No theorems – only empirical testing.

- Don’t measure running time.

- Don't measure running time.
  - Measure what an algorithmic theory might predict.
    - Subroutine calls, elementary data structure operations, etc.
  - *Simulate* an algorithm.

- Controlled experimentation addresses the shortcomings of competitive testing…

- The results depend on extraneous factors, such as…
  - Coding skill.

- The results depend on extraneous factors, such as…
  - Coding skill.
  - *The speed of the code is irrelevant, only the number of subroutine calls.*
  - *One could conceivably write the code in Mathematica.*

- The results depend on extraneous factors, such as…

- The results depend on extraneous factors, such as…
  - Parameter tuning.
  - *Test the algorithm across a range of parameters.*
  - *Factorial design includes parameters.*

- Random problem instances are unrealistic.

- Random problem instances are unrealistic.
- *Control for problem structure.*
- *Realism is irrelevant.*
  - *Performance on real problems is predicted by their characteristics.*

- A real problem set may be unrepresentative.

- A real problem set may be unrepresentative.
- *Eliminate benchmark problems.*
- *Representativeness is irrelevant.*

- Competitions tell us which codes are faster, but not why.

- Competitions tell us which codes are faster, but not why.
- *Isolate the factors that influence performance.*
- *Measure interaction between parameters and problem characteristics.*

How can a **competition** address the shortcomings of competitive testing?

- Most radical proposal--have a competition of empirical **theories**.

- Most radical proposal--have a competition of empirical **theories**.
  - The code must be accompanies by a paper and problem generator.
  - The paper proposes a theory for how the code performs.
  - The competition generates problems and tests the paper's theory.

- More modest proposals…

- Create a test suite based on a factorial design.
  - Identify several factors that may influence performance.
    - Type of scheduling problem.
    - Size.
    - Width of time windows.
    - Other parameters.

- Give awards based on multiple criteria.
  - Criteria may include:
    - Performance on each problem type.
    - Measures of scalability.
  - Winners must at least be pareto optimal.
  - View competition as a tournament.
    - Each pairing of solvers on each problem type is a “game.”

- Perform statistical analysis of test results.
  - Competition organizers write a paper for publication.
  - Paper proposes one or more empirical theories.
  - Design competition to test predefined theories?

- Require the code to have switches that turn on various features and knobs to adjust parameters.
  - The tests should statistically analyze the effect of the features/parameters and their combinations.
  - One parameter is extent of search.
    - Continuum of exact/heuristic methods.

- Pre-define aspects of the algorithm that must be simulated.
  - Number of problem restrictions enumerated.
    - Branches, neighborhoods, subproblems.
  - Effectiveness of inference.
    - Filtering, propagation.
  - Strength of relaxation/bounds.
    - LP/Lagrangean bound, cutting planes.

Other ideas?