Lecture slides for

Automated Planning: Theory and Practice

Running Experiments for Your Term Projects

Dana S. Nau

CMSC 722, AI Planning

University of Maryland

Measuring a Program's Performance

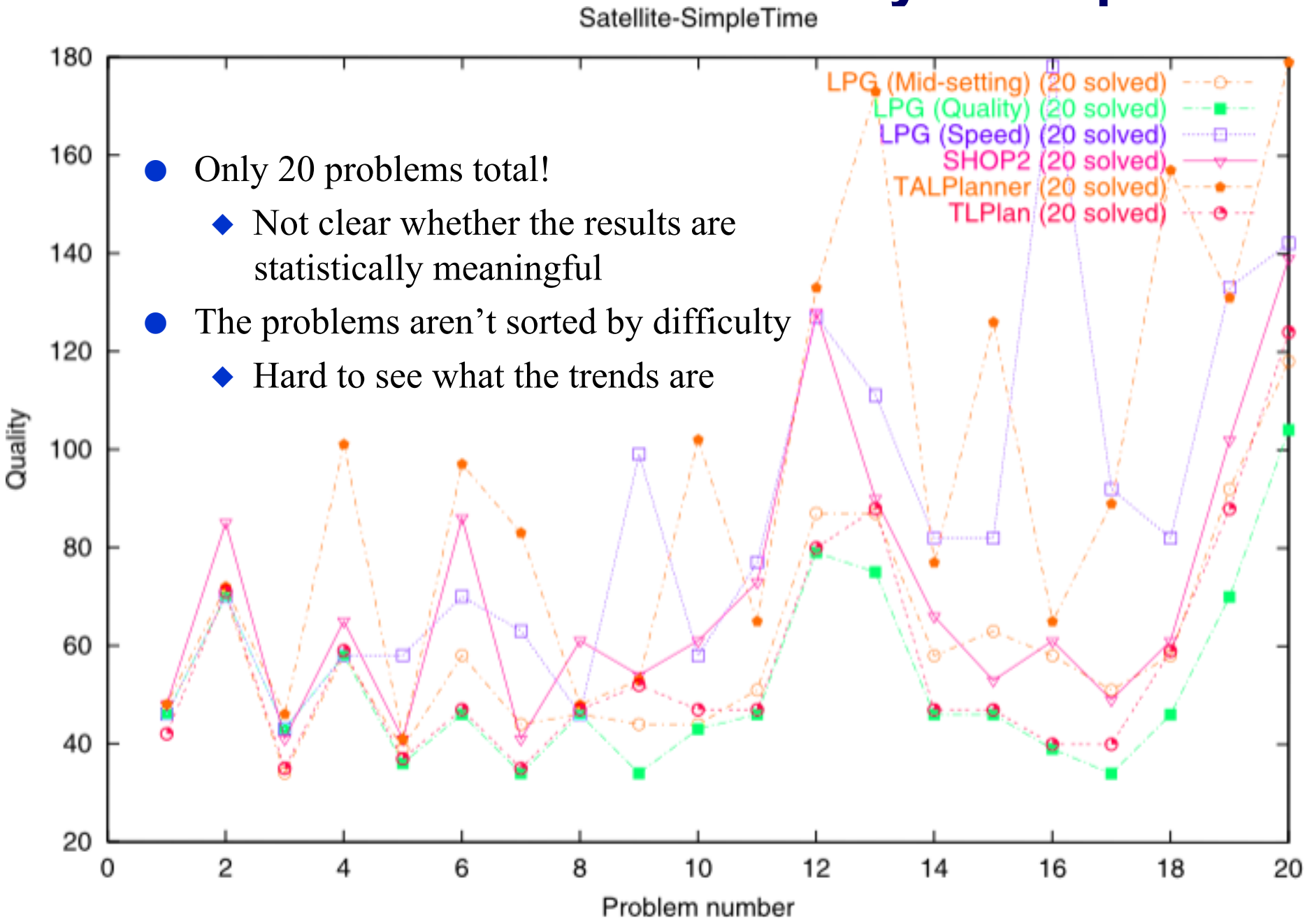
- Some of you want to run experiments to measure the performance of a program
- Possible things to measure
  - Running time
    - » usually CPU time, not clock time
  - Solution quality
    - » size? cost? some other measure?
  - Number of problems solved
  - Other things
    - » E.g., in a game you might measure the number of wins, or the number of points
- How to display the results
  - Usually as a graph
  - It can be hard to look at a large table of numbers and figure out what it means

Sources of Test Problems

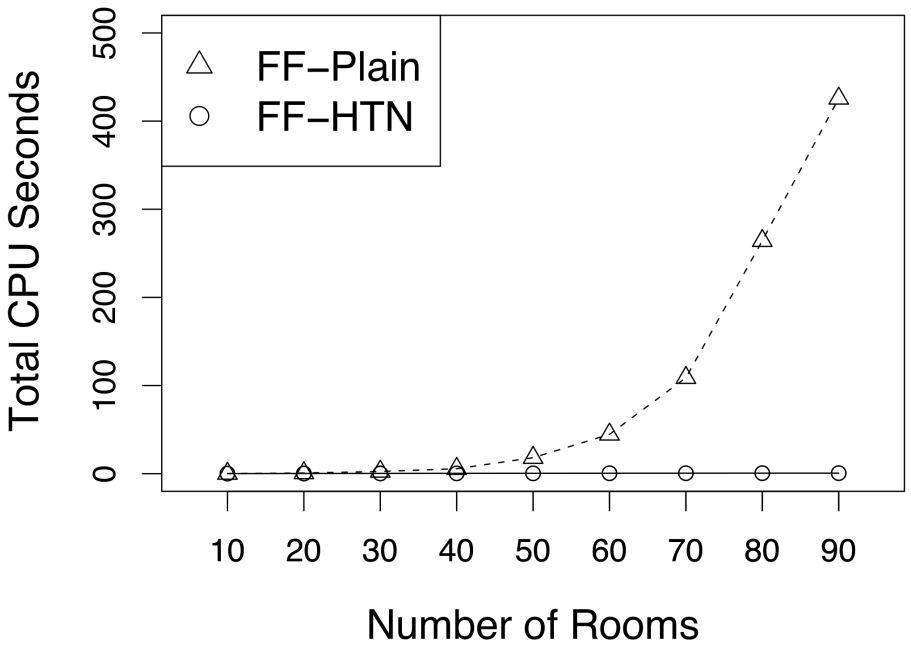
- International Planning Competition
  - http://ipc.icaps-conference.org/
- For most of the competitions, you can download the entire set of competition problems
- Usually they're written in a language called PDDL
  - http://cs-www.cs.yale.edu/homes/dvm/
- Some PDDL tutorials:
  - http://www.ida.liu.se/~TDDA13/labbar/planning/2003/writing.html
  - http://www.cs.toronto.edu/~sheila/2542/w09/A1/introtopddl2.pdf
  - http://www.cs.cmu.edu/~mmv/planning/homework/PDDL_Examples.pdf

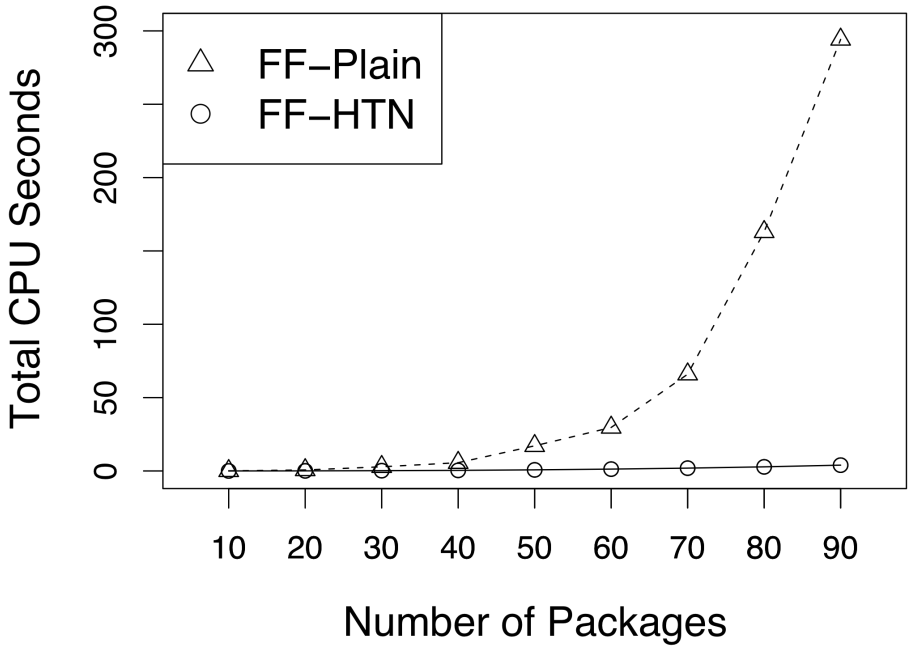
The IPC test data isn't always adequate

- Only 20 problems total!
  - Not clear whether the results are statistically meaningful
- The problems aren't sorted by difficulty
  - Hard to see what the trends are

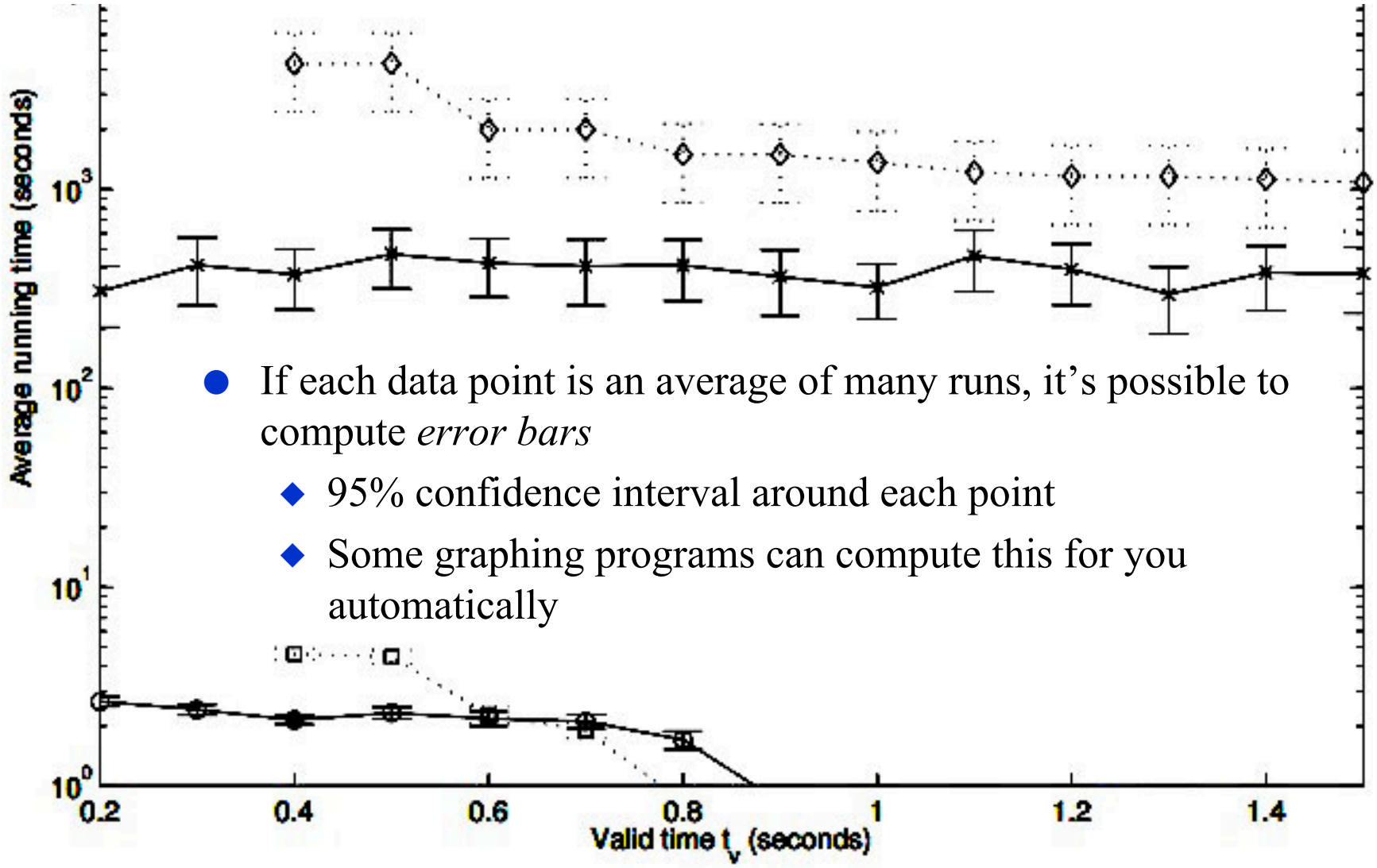


- Preferably each *data point* should be an average of at least 20 problems of similar difficulty
- What does *similar difficulty* mean?
  - Generally it means similar size
  - Sometimes there's more than one possible measure of size (see below)
- Below, each data point is an average over 100 problems
  - The curve is much smoother -- easier to see how the program is performing
  - These results have high statistical significance





Statistical significance?



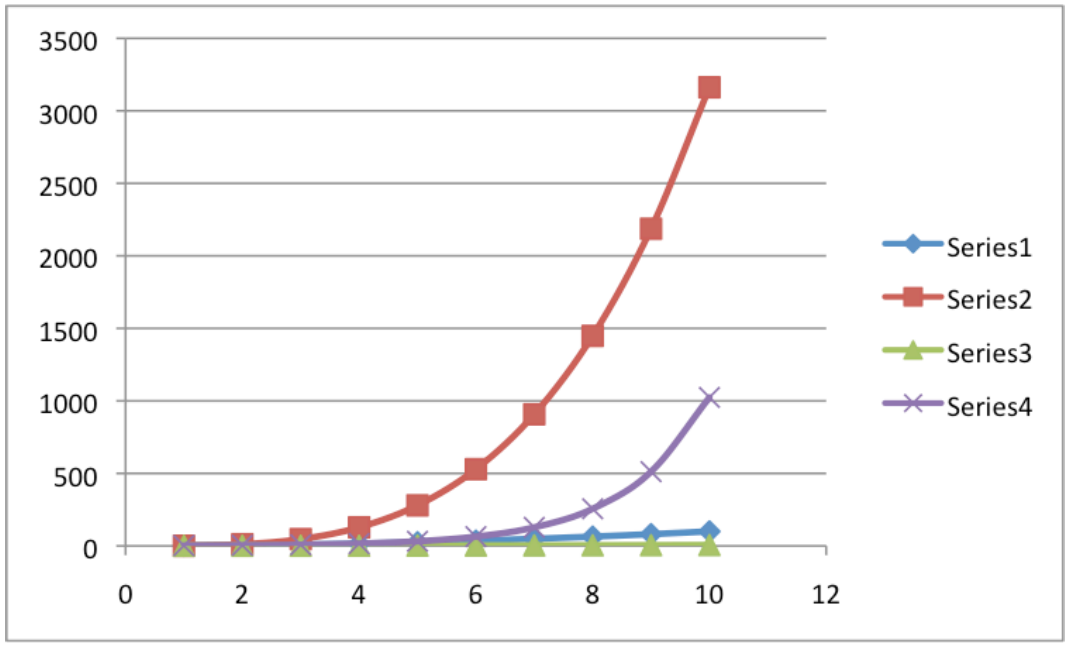
- If each data point is an average of many runs, it's possible to compute *error bars*
  - 95% confidence interval around each point
  - Some graphing programs can compute this for you automatically
- In other areas of AI (e.g., machine learning), people do this a lot
- In AI planning, almost nobody does it, and I won't require it

How to get a large set of problems

- In some cases, you can download the program that generated the problems, and use it to generate more problems
  - If you can't find the program on the web, ask me and I'll check to see whether I can find it
- In other cases you can build such a program yourself
  - But be careful *how* you do it
  - Important for the program to generate an unbiased random sample
  - On a biased sample, a program's performance can look much different than on an unbiased sample
- Many years ago, when one of my PhD students showed me his experimental results, his program appeared to be running in time $O(n^2)$
  - When I looked at how he was generating his problems, it turned out his program only generated very easy ones
  - When we fixed that, the running time turned out to be exponential

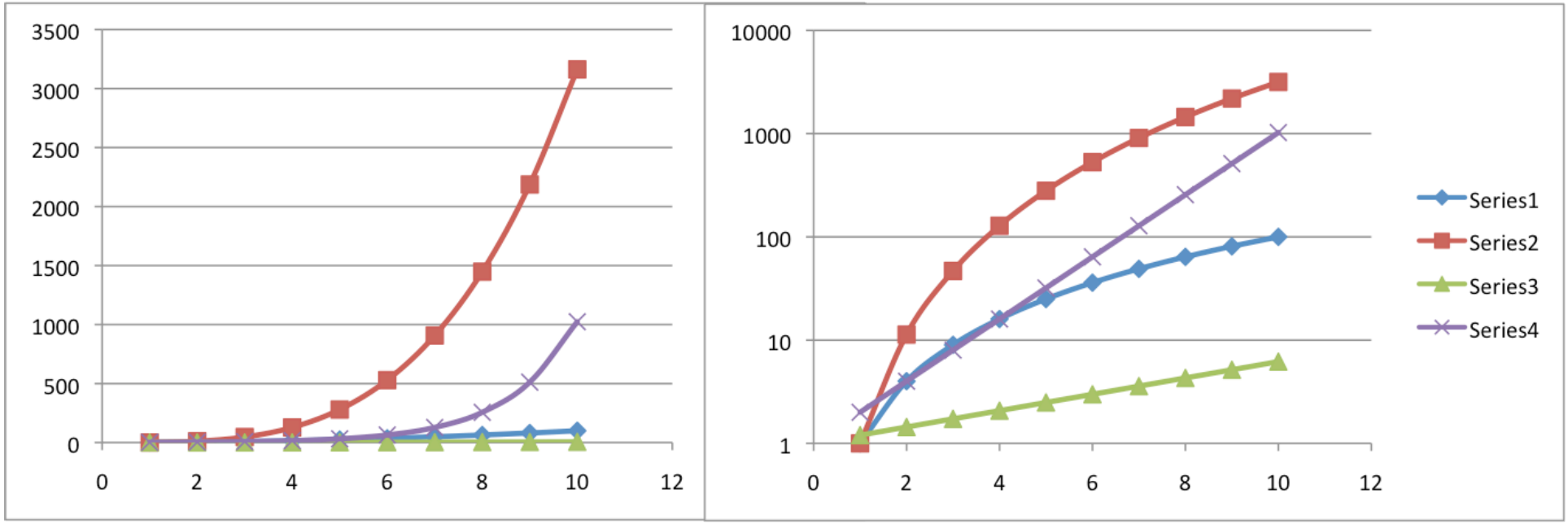
Polynomial versus exponential?

- Which of these is growing the fastest?



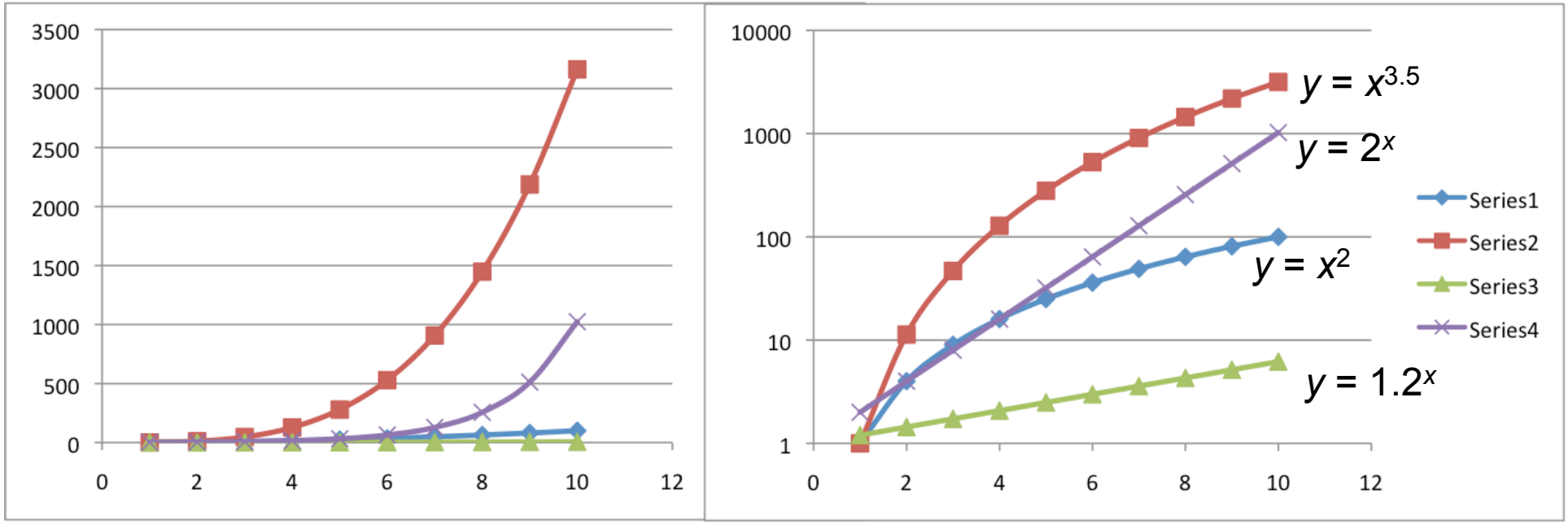
Polynomial versus exponential?

- Which of these is growing the fastest?



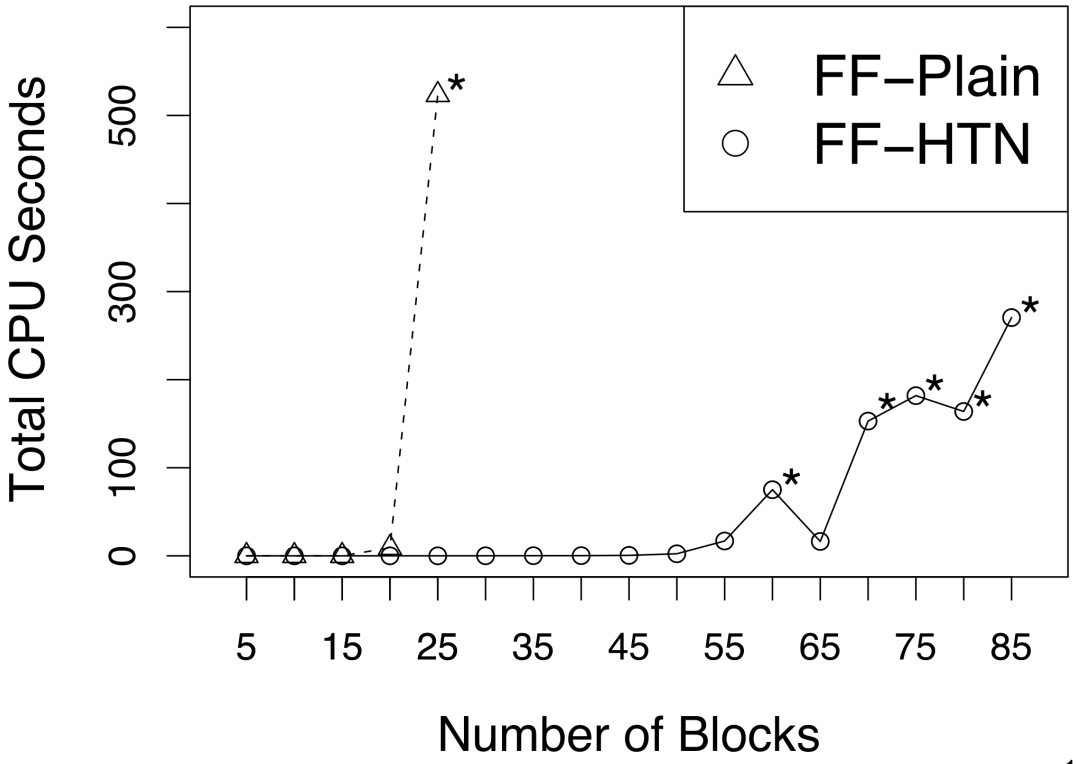
Polynomial versus exponential?

- Which of these is growing the fastest?

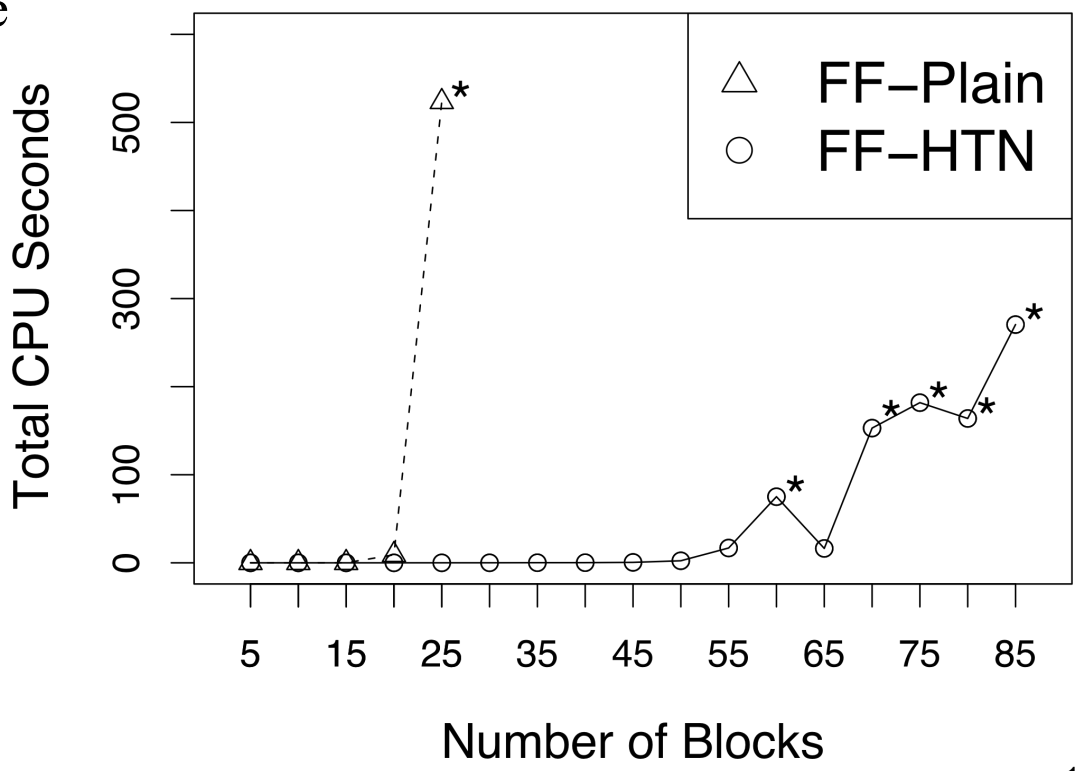


What if a program can't handle all the data?

- Sometimes a program can't solve all of the problems at each data point
  - It may run out of time or run out of memory
- In this case, you generally need to throw out that data point
  - ***Don't*** just take an average over the problems that the program can solve
- This biases the data – you're only reporting the average on the ***easy*** problems
  - That makes a program's performance look ***much*** better than it really is



- 2-hour time limit for each run
- If all 100 runs took less than 2 hours each, we took the average time per run
- If lots of runs failed to finish with 2 hours, we omitted those data points
  - No good way to get a good value for those data points
- If at least 97 of the runs finished within 2 hours
  - we counted others failure as 2 hours each when computing the average
  - we marked the data point with an asterisk, to tell the reader than in this case the program looks better than it actually is
- This gave us data points that were *close* to being correct
  - If we had thrown them away, the graph wouldn't have been as informative



What problem domains to use?

- Depends on what you're trying to show
- For a journal or conference paper
  - If you want to show that a program works well in lots of cases, then you generally want to compare its performance against other programs on several (at least 3?) domains that are significantly different from each other

- For the term project, results on a single domain are probably OK
  - Select a domain that illustrates a particular situation that you're trying to investigate

If you have difficulty

- If you're trying to evaluate a program's performance and you run into difficulty
  - e.g., there's some reason why it wouldn't be feasible to run your program on a large set of problems
- Come discuss it with me