Interesting things from the Remy study

- The authors of this paper did a pretty good job
- It was pretty easy to do
- Most people felt more confident in the results after
  * But did not necessarily understand what was going on better
  * Tradeoff between making it easy and teaching something
  * Know your audience; you don’t have to teach the basics
- Results came out similar, but not identical (Vegas consistently had something funny going on)
- Parameters / reasons behind them were not documented
- It was not easy to discover how to re-run individual experiments
  * Probably not all that hard to actually do, but hard to discover how
- One student suggested making VM images available
  * What would the downside be? — even more opaque
- The authors’ institution is not useful input into whether you should trust a paper
  * Established credibility of individual authors can be, though
  * But you should mostly be looking at the claims of the paper and how well they are supported, both in the body of the paper and in things it references
  * This is a scientific process, not a popularity contest

Definitions

- Repeatability: Do the same thing again, get the same results
  * Requires access to the code and same environment
- Reproducibility: Do it independently
  * Validate the idea, not just the code or environment
  * Does not necessarily require code
- Benefaction: Avoid needless replication of work
  * I don’t like this characterization
  * We do it for selfish reasons too
  * In systems, we build on code not just algorithms

Why I had you read the Collberg paper

- To get some important definitions
- To understand the state of reproducibility in our field
- To get a sense of what you are in for for the class project

- Main findings of the study
  - Code is hard to get
  - And often hard to build when you can get it

- Why are reproducibility and repeatability in CS so hard?
  - “All” you have to run software!
  - *Brainstorm on ideas*
    * Experiment environments hard to set up
    * Gathering dependences - esp. when those dependencies are themselves research artifacts
    * May not be documented
    * Experiments may not be scripted
    * May require special hardware
    * May require access to licensed software
    * Code not released
    * Visioning
  - Why people don’t release code
    * Afraid of having to support it
    * Afraid others will scoop them
    * Lose it
    * License agreement with industrial funder/partner
    * Control over use (don’t use it unoptimized or in inappropriate settings)
    * Afraid of having someone else show that they can do better and the obvious flaw with this reasoning
  - Cultural problem:
    * We are not rewarded for making it easy
    * Or for doing reproductions

- Lessons from Collberg paper
  - *How many of these are feasible?*
  - *Which are most important?*
  - Unless you have compelling reasons not to, plan to release the code
  - Students will leave, plan for it
  - Create permanent email addresses
  - Create project websites
  - Use a source code control system
  - Backup your code
  - Resolve licensing issues
  - Keep your promises
  - Plan for longevity
  - Avoid cool but unusual designs
- Don't rely on the permanence of external software
- Plan for repeatable releases.

• What we are going to for final project
  - Pick a paper
  - Pick one key graph
  - Get the code
  - Try to reproduce that graph
  - Improve the evaluation some way (measure variance, compare to a different system, improve representativeness or environment or workload, etc.)
  - Make a profile for CloudLab/Apt
  - Document how you did it
  - Start by picking two papers, picking the graph, and finding the code
  - You are probably not going to do both, I want a backup in case one turns out to be too hard

• For next time
  - Read Chapter 14, linear regression
  - Think of some papers, due as an assignment by Thursday
  - Don't forget about papers3