CIS 602-01: Computational Reproducibility

Numerical Reproducibility

Dr. David Koop
Tools

- We have seen specific tools that address particular topics:
  - Versioning and Sharing: Git, Github
  - Data Availability and Citation: DOIs, Dryad, DataONE, figshare
  - Virtual Machines and the Cloud: Xen, Parallels, EC2
  - Containers: Docker
  - Scientific Workflows: Pegasus, Kepler, VisTrails, Taverna
  - Provenance: PDIFF, Analogies, VisComplete

- More at ReproMatch

- Putting them together?
  - Packaging: CDE, ReproZip
  - Versioning: Sumatra, noWorkflow
Sumatra Reproducibility Issues

• Complexity:
  - dependence on small details
  - small changes have big effects

• Entropy:
  - computing environment
  - library versions change over time

• Human memory limitations:
  - forgetting
  - implicit knowledge not passed on

[Davison, 2012]
Sumatra Reproducibility Solutions

• Complexity: use/teach good software-engineering practices
  - loose coupling
  - testing

• Entropy: plan for reproducibility from the start
  - run in different environments
  - write tests
  - record dependencies

• Human memory limitations
  - Record everything

[Davison, 2012]
Running Sumatra

$ python main.py default.param
$ smt run --executable=python --main=main.py default.param
$ smt configure --executable=python --main=main.py
$ smt run default.param

• Configure sets default executable for iterative development
• Detect every change & prompt user (or record diff automatically)

$ smt run default.param
Code has changed, please commit your changes.
$ smt configure --on-changed=store-diff
$ smt run default.param

[Davison, 2012]
Steps in capturing experiments in Sumatra

- Create new record
- Has the code changed?
  - Yes
    - Code change policy
      - Error
      - Store diff
    - Yes
    - Get platform information
      - Run simulation/analysis
      - Record time taken
      - Find new files
      - Add tags
      - Save record
  - No
    - Find dependencies
      - Save record

[Source: Davison, 2012]
Record annotations and tags

- Important to include meaningful comments and tags for recall later

```bash
$ smt run --label=haggling --reason="determine whether the gourd is worth 3 or 4 shekels" romans.param
$ smt comment "apparently, it is worth NaN shekels."
$ smt tag "Figure 6"
```

[Davison, 2012]
Repeating past experiment

$ \text{smt repeat haggling}$

The new record exactly matches the original.

[Davison, 2012]
Reviewing Provenance

![Browser interface with ProjectGlass data](image)

<table>
<thead>
<tr>
<th>Repository</th>
<th>Label</th>
<th>Tag</th>
<th>Reason</th>
<th>Outcome</th>
<th>Duration</th>
<th>Date</th>
<th>Time</th>
<th>Executable name</th>
<th>Executable version</th>
<th>Main file</th>
<th>Version</th>
<th>Arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Glass</td>
<td>20121026-174545</td>
<td>Precise, default</td>
<td>Fixed to work with the new ntplib.histogram() function</td>
<td>3.85s</td>
<td>26/10/2012</td>
<td>17:45:45</td>
<td>Python</td>
<td>glass_sem_analysis.py</td>
<td>2.6.5</td>
<td></td>
<td>924a3</td>
<td></td>
</tr>
<tr>
<td>Glass</td>
<td>20121026-172642</td>
<td>error</td>
<td>Running on Ubuntu 12.04</td>
<td>TypeError: histogram() got an unexpected keyword argument 'new'</td>
<td>3.07s</td>
<td>26/10/2012</td>
<td>17:26:42</td>
<td>Python</td>
<td>glass_sem_analysis.py</td>
<td>2.6.5</td>
<td></td>
<td>432ff</td>
</tr>
<tr>
<td>Glass</td>
<td>20121025-173606</td>
<td>Lucid</td>
<td>No filtering, but more cleaning</td>
<td>2.82s</td>
<td>25/10/2012</td>
<td>17:36:06</td>
<td>Python</td>
<td>glass_sem_analysis.py</td>
<td>2.6.5</td>
<td></td>
<td>432ff</td>
<td></td>
</tr>
<tr>
<td>Glass</td>
<td>20121025-173350</td>
<td>Lucid</td>
<td>Trying a different colormap (&quot;hot&quot;)</td>
<td>&quot;Copper&quot; is nicer</td>
<td>4.08s</td>
<td>25/10/2012</td>
<td>17:33:50</td>
<td>Python</td>
<td>glass_sem_analysis.py</td>
<td>2.6.5</td>
<td></td>
<td>432ff</td>
</tr>
<tr>
<td>Glass</td>
<td>20121025-173036</td>
<td>Lucid</td>
<td>No filtering</td>
<td>2.83s</td>
<td>25/10/2012</td>
<td>17:30:36</td>
<td>Python</td>
<td>glass_sem_analysis.py</td>
<td>2.6.5</td>
<td></td>
<td>432ff</td>
<td></td>
</tr>
<tr>
<td>Glass</td>
<td>20121025-172833</td>
<td>Lucid, default</td>
<td>Parameters are now in a separate file</td>
<td>3.05s</td>
<td>25/10/2012</td>
<td>17:28:33</td>
<td>Python</td>
<td>glass_sem_analysis.py</td>
<td>2.6.5</td>
<td></td>
<td>432ff</td>
<td></td>
</tr>
<tr>
<td>Glass</td>
<td>20121025-170718</td>
<td>Lucid, default</td>
<td>Image file now specified on command line</td>
<td>3.73s</td>
<td>25/10/2012</td>
<td>17:07:18</td>
<td>Python</td>
<td>glass_sem_analysis.py</td>
<td>2.6.5</td>
<td></td>
<td>9d24b</td>
<td>MV_HVF_012.jpg</td>
</tr>
<tr>
<td>Glass</td>
<td>20121025-163949</td>
<td>Lucid, default</td>
<td>First run with Sumatra</td>
<td>3.76s</td>
<td>25/10/2012</td>
<td>16:39:49</td>
<td>Python</td>
<td>glass_sem_analysis.py</td>
<td>2.6.5</td>
<td></td>
<td>89af3</td>
<td></td>
</tr>
</tbody>
</table>

[Davison, 2012]
Reviewing Provenance

<table>
<thead>
<tr>
<th>Comparison of records</th>
<th>20121025-172833</th>
<th>20121026-174545</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reason</td>
<td>Parameters are now in a separate file</td>
<td>Fixed to work with the new numpy.histogram function</td>
</tr>
<tr>
<td>Outcome</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Timestamp</td>
<td>25/10/2012 17:28:33</td>
<td>26/10/2012 17:45:45</td>
</tr>
<tr>
<td>Duration</td>
<td>3.85s</td>
<td>3.85s</td>
</tr>
<tr>
<td>Executable</td>
<td>Python version 2.6.5 (/usr/bin/python)</td>
<td>Python version 2.7.3 (/usr/bin/python)</td>
</tr>
<tr>
<td>Launch mode</td>
<td>serial</td>
<td>serial</td>
</tr>
<tr>
<td>Repository</td>
<td>/home/bob/Projects/Glass</td>
<td>/home/bob/Projects/Glass</td>
</tr>
<tr>
<td>Main file</td>
<td>glass_sem_analysis.py</td>
<td>glass_sem_analysis.py</td>
</tr>
<tr>
<td>Version</td>
<td>432f7e3f845</td>
<td>924e393a3d24c</td>
</tr>
</tbody>
</table>

File name: 20121025/MV_HPV_012_172836_phasos.png

File name: 20121026/MV_HPV_012_174557_phasos.png

Digest: c9955f84ca3e19123d24ec4e8d197514d9e01e

Digest: 7f8ed0c6e6f97b8317af8ed9ab9f85a193ca2687

Dependencies:

<table>
<thead>
<tr>
<th>Name</th>
<th>Path</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>dateutil</td>
<td>/usr/lib/pymodules/python2.6/dateutil</td>
<td>1.4.1</td>
</tr>
<tr>
<td>glib</td>
<td>/usr/lib/pymodules/python2.6/glib</td>
<td>unknown</td>
</tr>
<tr>
<td>gobject</td>
<td>/usr/lib/pymodules/python2.6/gobject</td>
<td>unknown</td>
</tr>
<tr>
<td>matplotlib</td>
<td>/usr/lib/pymodules/python2.6/matplotlib</td>
<td>0.99.1.1</td>
</tr>
</tbody>
</table>

[Davison, 2012]
Sumatra Components

- Specific plugins for dependency finding, version control
noWorkflow

- Goal: Provenance analysis for **scripts**
- Python-specific tool, approach general
- Like Sumatra, tracks changes in files and scripts
- Like CDE/ReproZip, intercepts file access & environment info
- Also does profiling and tracing to track runtime information

- `python my_script.py → now run my_script.py`
noWorkflow

Provenance Capture

**Definition Provenance**
- AST Analysis and Disassembly

**Deployment Provenance**
- Python's modules (os, socket, platform, modulefinder)

**Execution Provenance**
- Profiling, Tracing and Reflection

Provenance Storage

- SQLite
- Content Database
- .noworkflow directory

Provenance Analysis

- Visualization of Trials
- Diff Analysis
- Querying
- IPython Notebook

[Murta et al., 2014]
noWorkflow Provenance Analysis

[Murta et al., 2014]
Assignment 2

- Keep your project on Github, keep images on Docker Hub
- Put a link to your Docker Hub images in your Github README.md
- Questions?
  - Running inside of Docker versus on your local machine
  - MacOS X stdout: start VisTrails with VisTrails.command
Projects

• Report
  - Reproducibility:
    • Summary of paper's main contributions
    • Reproducibility overview (what is available?, does it work?)
    • Experiments: what experiments are being reproduced and what is the ground truth
    • Results: what was reproducible, what wasn't, why
  - Research:
    • Like a standard research paper (see Lecture 2)
Projects

• Presentation (8-10 minutes):
  - Reproducibility:
    • **Short** summary of the paper
    • Summary about what is available, what works, etc.
    • Best: show reproducibility in action
    • Good: show side-by-side results
  - Research (focus on **reproducibility** aspects!):
    • **Short** motivation and introduction
    • Reproducibility requirements
    • Discuss Results
    • Best: show tool/reproducibility in action
Reproducibility Levels

- Data Availability
- Script / Workflow
- Lower-level code
- Environment
- Actual Hardware
What happens at the hardware level that may not be reproducible?
Hardware and Reproducibility

[Pentium Chip with FDIV Bug, Photo by Konstantin Lanzet, CC BY-SA 3.0]
To simulate the deformation of a plane sheet to a real car body part with such a deep drawing process, my colleagues and I at GNS use the Indeed program (www.gns-mbh.com/?id=indeed), a software system specially designed to simulate sheet metal forming based on the finite element method (FEM). As an example, consider the part depicted in Figure 2, which is supposed to be manufactured from a 0.6-millimeter-thick sheet of steel. The part is about 350-mm long and 400-mm wide. The finite elements used for the simulation have an edge length of 4 mm.

At GNS, we repeatedly performed the simulation with different degrees of parallelism. Specifically, we varied the number of processors between one and four, and executed the simulation with four processors twice. All other input data were kept constant. We then observed the program’s behavior and compared the output data. In this case, we chose the output data to be a variable that’s particularly important in practice—namely, the local change of the sheet thickness. This variable is a crucial criterion in the process of deciding whether the manufactured part will in practice be suitable for its intended application. The user will typically have two threshold values for the sheet thickness. If the computed result is below the smaller of these values, the part will be considered insufficient and the deformation process must be modified to change this behavior. If the computed values are above the larger of the two values, the workpiece will be accepted, and if the computation produces results between the two given values then additional investigations are required to determine how to proceed.

Even though the algorithms used don’t contain any stochastic elements, the results always differed from each other in all respects. In particular, it’s not uncommon for results to move from one side of one threshold value to the other as the simulation is repeated. Thus, the way in which the engineers must proceed in developing their tools and the details of the tool control can also vary, depending on how the algorithms are executed on the given hardware.

To illustrate the variation of the program’s behavior, let’s look at two specific aspects. The first is the way in which its automatic time-step control works. The program decomposes the period of time required for the process in reality into individual time steps, and the simulation is performed in these discrete time steps. The sizes of the first two time steps are determined by the user (we always choose the same values for all our experiments); the following step lengths are automatically chosen in a deterministic way by the program depending on the convergence behavior of the previous step.

Figure 3 shows a visualization of the corresponding data in a graph that illustrates the share of the already-simulated process time with respect to the total process time over the number of time steps. It’s evident that the behavior differs from program run to program run. In particular, the two curves corresponding to the two runs with four processors (that is, for two executions of the same program with absolutely identical input data in every respect) significantly differ from each other.

We’ve also compared the computed changes in sheet thickness. Here, I present only some selected values. Table 1 shows the computed extreme values of the sheet thickness change. Even in this small dataset, the significant differences are

Example: Sheet Metal Forming

- Simulate in software (Indeed)
- Uses a finite element method
- Manufacture from 0.6mm thick steel
- 350mm long, 400mm wide
- Care about local change in sheet thickness (threshold values)
Results

<table>
<thead>
<tr>
<th>Description of the simulation (no. of processors)</th>
<th>Minimal value of the sheet thickness change (%)</th>
<th>Maximal value of the sheet thickness change (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>−29.21</td>
<td>+9.54</td>
</tr>
<tr>
<td>2</td>
<td>−29.34</td>
<td>+9.14</td>
</tr>
<tr>
<td>3</td>
<td>−29.04</td>
<td>+9.02</td>
</tr>
<tr>
<td>4 (1st attempt)</td>
<td>−29.04</td>
<td>+8.98</td>
</tr>
<tr>
<td>4 (2nd attempt)</td>
<td>−29.09</td>
<td>+8.96</td>
</tr>
</tbody>
</table>

![Figure 4](image.png)

(a) Max = 9.54

(b) Max = 8.96
Why does the number of processors matter?
Investigation

• Indeed uses a Pardiso subroutine
• Part of the Intel Math Kernel Library (MKL)
• Paradise is a non iterative solver for sparse linear systems that is reliable and efficient
• Pardiso is nondeterministic?
Generally

• How do we deal with this?
• Do we expect all scientists to drill down and figure this out?
• What solutions are there that ensure reproducibility?
Scalar Product Example

- Computation of scalar product
  \[ s = \sum_{i=1}^{n} x_i y_i \]

- Now assume we have a $10^6$-dimension vector

- Parallelize! Divide among $p$ processors the $n$ summands; $l$-th processor computes the following partial sum:
  \[ s_l = \sum_{i=(l-1)n/p+1}^{ln/p} x_i y_i \]

- Then, compute the final sum:
  \[ s = \sum_{i=1}^{p} s_l. \]
Fixed number of processors

• Processors complete in different orders...
• Compute the sum as the solutions come in
• n=8, p=4
• \( x_1 = 10^{12}, \; x_2 = 0, \; x_3 = 10^{-8}, \; x_4 = 0, \; x_5 = -10^{12}, \; x_6 = 0, \; x_7 = 10^{-8}, \; x_8 = 0, \) and \( y_j = 1 (j = 1, 2, \ldots, 8) \)
• \( s_1 = 10^{12}, \; s_2 = s_4 = 0, \; s_3 = -10^{12} \)
• If we have order \( s_1, \; s_3, \; s_2, \; s_4 \)

\[ z_1 := s_1 + s_3 = 10^{12} - 10^{12} = 0 \]

\[ z_2 := z_1 + s_2 = 0 + 10^{-8} = 10^{-8} \]

\[ s = z_2 + s_4 = 10^{-8} + 10^{-8} = 2 \cdot 10^{-8}. \]

• Order \( s_1, \; s_2, \; s_3, \; s_4 \) gives

\[ s = z' + s = 0 + 10^{-8} = 10^{-8}, \quad \text{(because } 10^{12} + 10^{-8} = 10^{12}) \ldots \]
Solution?

• Wait for all partial sums to come in, and then add them in a deterministic manner
Varied number of processors

• What is we have only two processors?

\[ s_1 = 10^{12} + 10^{-8} \]

\[ s_2 = -10^{12} + 10^{-8}, \]

\[ s^* = s_1^* + s_2^* = 10^{12} - 10^{12} = 0 \]

• Solutions?

  - Use virtual processors that is fixed no matter what the environment
Other solutions

• Understanding rounding
• Use higher precision
• Interval arithmetic
• Fixed-point arithmetic
• Uncertainty quantification

"The 'exact' reproducibility, i.e. identical numerical results on different number of processors, is impossible in our view, and is not our goal"
Ahn Paper

• Data-race free: the presence of data races can result in execution-order-dependent final results;

• Determinate (external determinism): a system can be considered deterministic if the final results are the same regardless of the internal execution details;

• Internal determinism: even the internal execution steps (traces) are required to be the same;

• Functional determinism: this refers to the absence of mutable state updates, inherently giving rise to determinate outcomes;

• Synchronous parallelism: some regard synchronously parallel execution models as deterministic, as opposed to asynchronous models.