Testing large, untested code bases

C. Titus Brown

t@idyll.org

first:

Multiprocessing is next door.

I like live demos.

This is not a talk on Web testing.

Testing large, untested code bases

C. Titus Brown
t@idyll.org

(thanks to organizers!)

Outline

The software package “under test”: science background.

The software package under test: code and authors.

Introduction: who am I?

Asst. Prof at Michigan State U., CSE/bio

Test-obsessed.

Author of twill & figleaf.

Biology, bioinformatics, and software engineering.

pygr – Python graph database

Big problems in biology:

- lots of sequence data
- very few good tools for handling in good way (scalable, enabling, and extensible)
- most bioinformaticians obsessed with parsing and “lists” of data

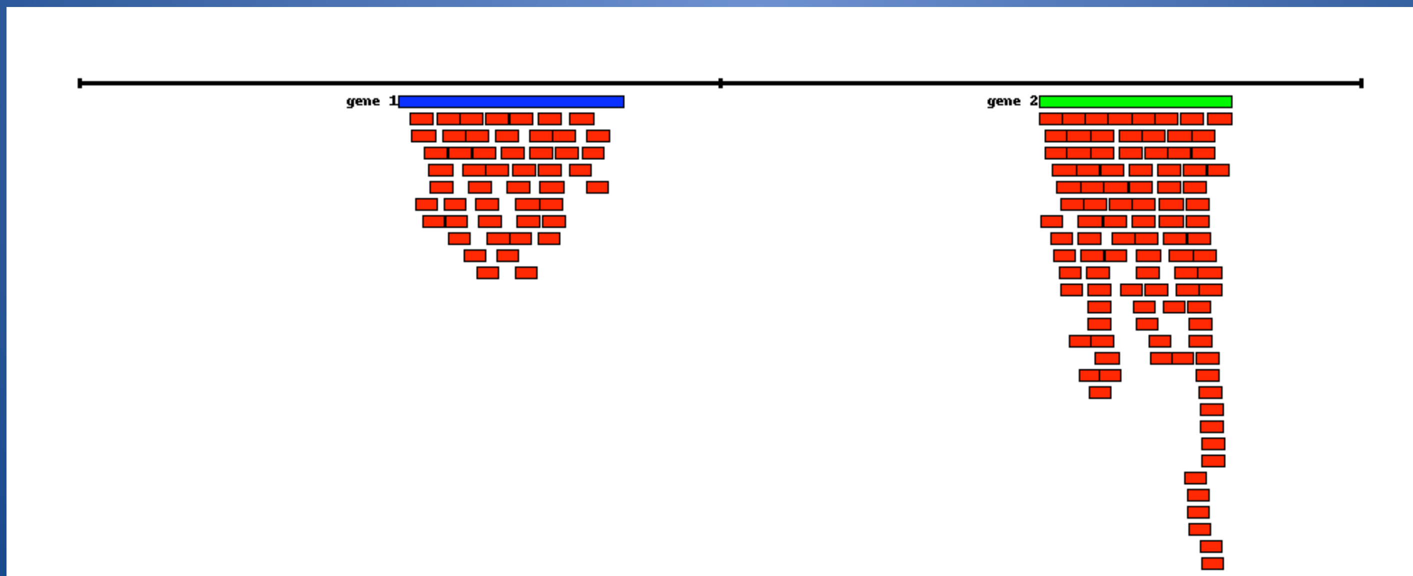
pygr offers a solution to these issues.

RNAseq

Read from entire RNA molecule

Map reads onto genome sequences & count.

Ratio =~ relative expression level

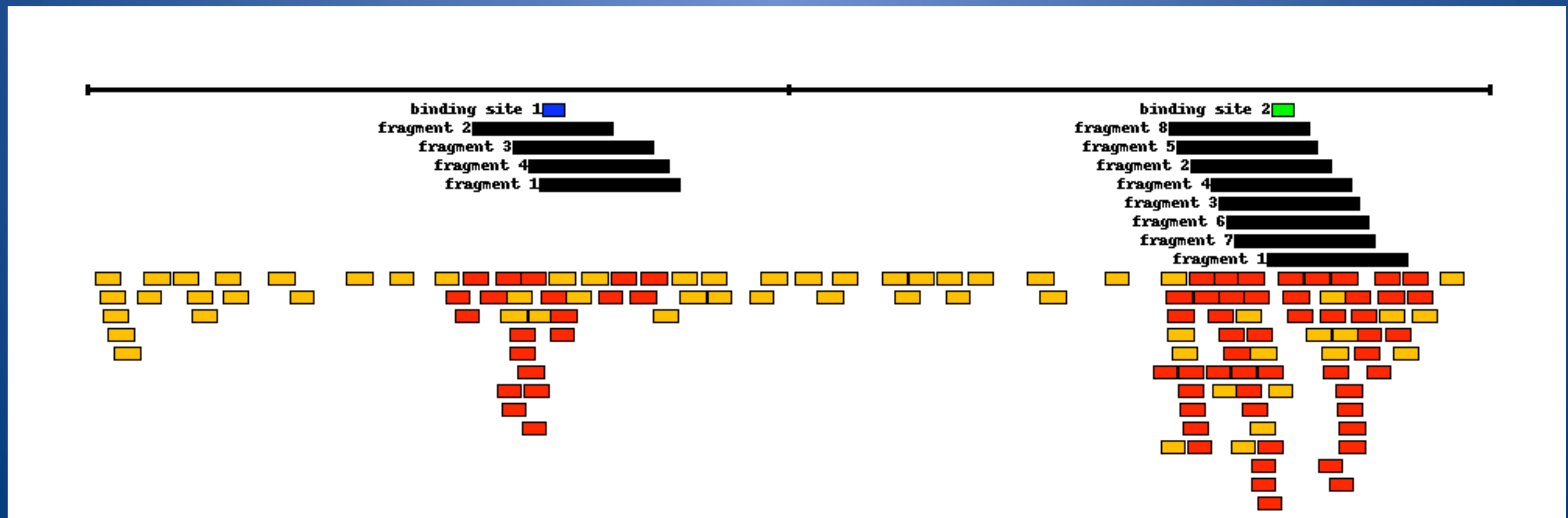


ChIP-seq

Select pieces of DNA by protein binding

Sequence them & map reads to genome

“Pile up” reads & look for binding sites.

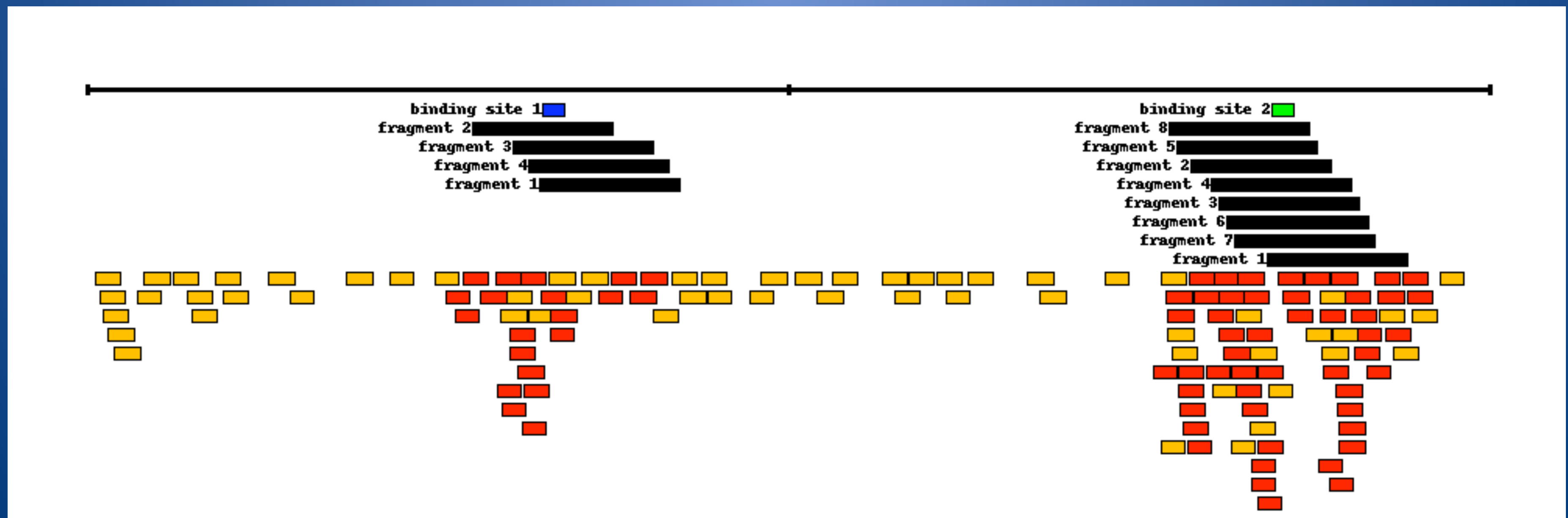


…now scale.

Do for 1-10 gb genome.

With mutations & indels.

With 10-100 gb of 30-base substrings.



pygr again

- Provides data abstraction with pygr.Data
  Pygr.Data.Bio.human.chr1
  - Could load from local or remote
  - Automatically links annotations (gene/etc.) and alignment information

- Retrieve annotations/alignment by simple reference,
  matches = alignments[chr1[500000:600000]]

  Scales to complete 17-genome alignment from UCSC.

Introducing the code

~8k of Python, ~2k of Pyrex (-> C, for speed).

Little app/UI code, almost all ***library*** and ***framework*** (complex).

Grew by accretion and personal use at UCLA, OSSed with hope that others would buy in.

(I've bought in ;)

Lots of technical debt.

Introducing the authors

Then: Chris Lee primarily; 2-3 others.

Now ~5-6 regular developers, including former GSoC student Jenny Qian.

Myself, Istvan Albert, and Marek Szuba.

(*Code ownership* is an issue, but more in perception than in actuality.)

Growing pygr (code + devs)

- Lots of unfamiliar code.
- Written at relatively high level throughout.
- Written by a very smart person; idiosyncratic?
- Functional, in use: backwards compat.
- Mix of developers: experienced Python devs, new students, between.
- Crosses domains! (Like most scientific code, and many apps.)
- Performance is *critical*.

Actually talking about testing.

(but wanted to provide rationale)

(Automated) testing is a partial solution to many of these problems.

What you already know

Unit tests and functional tests rock; use ‘em.

(Statement) code coverage is of limited utility, because it doesn’t measure branch coverage.

Wrong. WRONG. *WRONG.*

"(Statement) code coverage is of limited utility, because it doesn't measure branch coverage. "

There are plenty of reasons why code coverage is of limited use.

It is invaluable when aimed at:

- new test efforts on *legacy* code
- understanding code bases

“Software forensics”

Understanding big code bases is *hard.*

Code coverage can be used as a lever:

“Give me but a place to stand and with a lever I will move the entire world.”

- Archimedes

“Software forensics”

A mental track I’ve been following for some time:

See:

- figleaf sections (PyCon 2007)
- peekaboo (PyCon 2008)

figleaf is basically my bid to make a *more programmable coverage utility*, for use and abuse by me and others.

Grokking code thru coverage.

Start with minimum useful statement.

Examine code that's *actually executed.*

Add additional statement.

Examine executed code.

Repeat.

Grokking code thru coverage.

Lets you pull apart highly complex and interlinked code.

(High coupling & nonlinearity in system: c.f. Perrow, "normal accidents")

Code that connects abstraction blobs is hard to analyze any way other than *dynamically*, especially when *you didn't write it.*

Incredibly good tool for exploratory testing.

A couple of practical obstacles.

- Substantial portions of pygr are written in Pyrex, a Python-like language that is compiled into C with a Python interface (for speed). This code tends to be algorithmic in nature.
- Difficult to track changes to complex code from multiple developers, esp w/o complete test suite.
- Rapidly changing tests suites are hard to keep track of:
  - What tests should be run on which installs;
  - Running tests on multiple platforms;

Vignettes

- Including Pyrex code coverage.
- Keeping track of tests.
- Test “diff”
- “Coverage-driven testing”
- Code review...

Pyrex code coverage.

- Executable lines must be recorded at compile time.

  => Must hack Pyrex compiler.

- Must also hack figleaf to display, integrate.

Pyrex code coverage.

Pyrex code coverage.

- No real change in overall code coverage.
- …but surprise!
- Can also use for C code now; see GSoC project for core Python.

Keeping track of tests.

What tests are run? Are any missing?

Virtually impossible to keep track of with current software (!?). (See TIP list.)

So: record once, check in; examine each time you run the tests.

Now you have to break *two* things.

Keeping track of tests.

Q: how to handle dependencies?

Different “test lists”?

More structured file, e.g. YAML?

Test “tags” (e.g. nose, py.test)?

Test “diff”

Coverage driven testing.

Simple concept:

Each new test should “attack” an uncovered line of code.

Immediate gratification of new code coverage (like “dot” addiction); finds simple bugs with embarrassing ease; **you now understand that code.**

Code review of existing modules.

Style (PEP 8, PEP 257).

Documenting for framework programmers.

Design decisions? Coupling?

Spreading ownership.

Code review of existing modules.

git (or any DVCS) is awesome for this.

Reformatting vs “search/replace” vs real functionality changes.

Recent code review concluded:

- one stupid bug found & fixed;
- one bug introduced and found up on re-review;
- one bug introduced and found by performance tests.

Conclusions

- Code coverage is an invaluable lever for prying into other people's code bases.
- We're still missing tools to make this easy, although I'm trying to grow figleaf into this role.
- Code testability is an important issue:
  - Example seqdb fixture code!
  - pygr.Data used to be untestable; new code is shorter, cleaner, easier to understand.

What next?

- Tracing code coverage through repositories.
  (commit with code; track between diffs; automate?)
- Branch coverage!! (GSoC project?)
- Simplify continuous integration
  - Buildbot is too complex
  - Too difficult to extend
  - See testing BoF…

What else at PyCon?

- figleaf sprinting?
- Testing BoF, Saturday evening.
- Twill sprint on Monday.
- Tuesday?

Note: testing-in-python list.