The art of Programming
Evolutionary Algorithms

Juan-Julián Merelo.
@jjmerelo

U. Granada (Spain)
Get out and smell the air
New is always better

Programming paradigms are changing on a daily basis

GPGPU     Cloud computing

NoSQL     Map/reduce

Internet of Things
Form should fit function
And we should adapt to the new
We should be able to write publishable papers, artifacts painlessly through efficient, maintainable, scientific programming.
Mind your environment
Open source your code and data
Aw, maaaaan!

- Open source first, then program
  - Scientific code should be born free.
- Science must be reproductible.
- Easier for others to compare with your approach
  - Increased H
    - Scientist heaven!
- Manifest *hidden* assumptions.

- If you don't share, you don't care!
Minimize bugs via test-driven programming
Tests before code

- What do you want your code to do?
  - Mutate a bit string, for instance.

- Write the test
  - Is the result from mutation different from the original?
    - Of course!
    - But will it be even if you change an upstream function? Or the representation?
  - Does it change all bits in the same proportion (including first and last ➞ corner cases)?
Control the source of your power
Source control systems save the day

- Source code management systems allow:
  - Checkpoints
  - Stygmergic interaction
  - Individual responsibility over code changes
  - Branches

- Distributed are *in*: git, mercurial, bazaar
- Centralized are *out*: subversion, cvs.

- Instant backup!
Code complete

1) Check out code/Update code

2) Make changes

3) Commit changes (and push to central repository)
Go with the Joneses

Use GitHub: http://github.com
Pushing is not the end of the story

- Tests must be run, compilations made, checks and balances checked and balanced.
- Use Travis or Jenkins
  - If it's good enough for software developers, it's good enough for scientists!
- All this is free if you open source your code
  - Back to #2
Be language agnostic
Language shapes thought

- Don't believe the hype:
  - Compiled languages are faster... NOT
  - There is no free lunch.

- Avoid programming in C in every language you use

- Consider scripting languages: Python, Perl, Lua, Ruby, Clojure, Javascript... interpreted languages are faster.
Language agnoticism at its best

Evolving Regular Expressions for GeneChip Probe Performance Prediction

http://www.springerlink.com/content/j3x8r1

The regular expressions are coded in AWK scripts:

Although this may seem complex, gawk (Unix’ free interpreted pattern scanning and processing
Programming speed > program speed
Scientists, not software engineers

- Our deadlines are for papers – not for software releases (but we have those, too).
- What should be optimized is speed-to-publish.
- Makes no sense to spend 90% time programming – 5% writing the paper.
- Scripting languages rock
  - and minimize time-to-publish.
Perl faster than Java?

Algorithm::Evolutionary, a flexible Perl module for evolutionary computation

http://www.springerlink.com/content/8h025g83j0q68270/

- Class-by-class, Perl library much more compact
  - Less code to write.
    - More time to write the paper, perform experiments....

- In pure EC code, Algorithm::Evolutionary was faster than ECJ.
Don't assume:

measure
Performance matters

- Basic measure: CPU time as measured by `time`

```bash
jmerelo@penny:~/proyectos/CPAN/Algorithm-Evolutionary/benchmarks$ time perl onemax.pl
0; time: 0.003274
1; time: 0.005438
[...]
498; time: 1.006539
499; time: 1.00884
```
Performance Profile Index
For run_experiment_instances.pl

Profile of run_experiment_instances.pl for 166 (of 407) sources, executing 171726297 statements and 13126070 subroutine calls in 114 source files and 85 string evals.

Top 15 Subroutines

<table>
<thead>
<tr>
<th>Calls</th>
<th>P</th>
<th>F</th>
<th>Exclusive Time</th>
<th>Inclusive Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>2773830</td>
<td>4</td>
<td>2</td>
<td>35.7s</td>
<td>35.7s</td>
</tr>
<tr>
<td>396600</td>
<td>1</td>
<td>1</td>
<td>26.9s</td>
<td>26.9s</td>
</tr>
<tr>
<td>198300</td>
<td>1</td>
<td>1</td>
<td>17.1s</td>
<td>43.9s</td>
</tr>
<tr>
<td>405600</td>
<td>2</td>
<td>1</td>
<td>11.6s</td>
<td>12.2s</td>
</tr>
<tr>
<td>396600</td>
<td>1</td>
<td>1</td>
<td>10.8s</td>
<td>31.2s</td>
</tr>
<tr>
<td>396600</td>
<td>2</td>
<td>1</td>
<td>8.87s</td>
<td>9.38s</td>
</tr>
<tr>
<td>899000</td>
<td>2</td>
<td>1</td>
<td>6.83s</td>
<td>40.7s</td>
</tr>
<tr>
<td>690000</td>
<td>1</td>
<td>1</td>
<td>6.27s</td>
<td>33.9s</td>
</tr>
<tr>
<td>159300</td>
<td>1</td>
<td>1</td>
<td>6.18s</td>
<td>27.6s</td>
</tr>
<tr>
<td>396600</td>
<td>1</td>
<td>1</td>
<td>5.28s</td>
<td>17.2s</td>
</tr>
<tr>
<td>38</td>
<td>2</td>
<td>1</td>
<td>4.88s</td>
<td>165s</td>
</tr>
<tr>
<td>661</td>
<td>1</td>
<td>1</td>
<td>4.47s</td>
<td>96.1s</td>
</tr>
<tr>
<td>584</td>
<td>3</td>
<td>1</td>
<td>4.06s</td>
<td>18.4s</td>
</tr>
<tr>
<td>198300</td>
<td>1</td>
<td>1</td>
<td>3.56s</td>
<td>13.6s</td>
</tr>
<tr>
<td>1718600</td>
<td>4</td>
<td>3</td>
<td>2.48s</td>
<td>2.48s</td>
</tr>
</tbody>
</table>

Subroutine:
- Algorithm::MasterMind::check_combination
- Algorithm::Evolutionary::Wheel::first
- Algorithm::Evolutionary::Wheel::spin
- Algorithm::Evolutionary::Op::String_Mutation::apply
- Algorithm::Evolutionary::Op::Base::check
- Algorithm::MasterMind::distance_taxicab
- Algorithm::MasterMind::matches
- Algorithm::MasterMind::check_rule
- Algorithm::Evolutionary::Individual::String::clone
- Algorithm::MasterMind::EvoRank::issue_next
- Algorithm::Evolutionary::Op::Canonical_GA::apply
- Algorithm::MasterMind::partitions
- Algorithm::Evolutionary::Op::QuadXCover::apply
- Algorithm::Evolutionary::Individual::Base::Fitness

See all 2644 subroutines

You can view a treemap of subroutine exclusive time, grouped by package. NYTProf also generates call-graph files in Graphviz format: inter-package calls, all inter-subroutine calls (probably too complex to render easily).

You can hover over some table cells and headings to view extra information. Some table column headings can be clicked on to sort the table by that column.
There's always a better algorithm/data structure
And differences are huge

- Sort algorithms are an example
  - Plus, do you need to sort the population?

- Cache fitness evaluations
  - Cache them permanently in a database?
    - Measure how much fitness evaluation takes

- Thousand ways of computing fitness
  - How do you compute the MAXONES?
    - \$fitness_of\{\$chromosome\} = ($copy_of =~ tr/1/0/);

- Algorithms and data structures interact.
Case Study: EAs as software programs

Time analysis of standard evolutionary algorithms as software programs

http://dx.doi.org/10.1109/ISDA.2011.6121667

Programs implementing EAs are analyzed; huge improvements can be achieved by changing random number generators or memory usage patterns

Implementation matters!
Learn the tricks of the trade
Two trades

- Evolutionary algorithms
  - Become one with your algorithm.
    - It does not work, but for a different reason that what you think it does

- Programming languages.
  - What function is better implemented?
  - Is there yet another library to do sorting?
  - Where should you go if there's a problem?

- Even a third trade: programming itself.

This is the Zen!
Case study: sort

- Sorting is routinely used in evolutionary algorithms
  - Roulette wheel, rank-based algorithms

- Faster sorts (in Perl):
  http://raleigh.pm.org/sorting.html
  - Sorting implies comparing
  - *Orcish* Manoeuver, *Schwartzian* transform
  - Sort::Key, fastest ever
    http://search.cpan.org/dist/Sort-Key/
Make experiment processing easy.
Avoid drowning in data

- Every experiment produces megabytes of data
  - Timestamps, vectors, arrays, hashes.
  - Difficult to understand after some time.
- Use serialization languages for storing data
  - YAML: Yet another markup language.
  - JSON: Javascript Object Notation.
  - XML: eXtensible Markup language.
  - [Name your own].
Case study: Mastermind

Entropy-Driven Evolutionary Approaches to the Mastermind Problem

Carlos Cotta et al., http://www.springerlink.com/content/d8414476w2044g2m/

- Output uses YAML.
- Includes:
  - Experiment parameters.
  - Per-run and per-generation data.
  - Final population and run time.

Open source! (Follow #2!)
When everything fails, visualize...
backup your data
Better safe than unpublished

- Get an old computer, and backup everything there.
  - If you do open science, you get that for free!

- In some cases, create virtual machines to reproduce one paper's environment
  - Do you think gcc 3.2.3 will compile your old code?

- Use rsync, bacula or simply cp.

- It's not if your hard disk will fail, it's when.

Cloud solutions are OK but backup that too.
Keep stuff together
Where did I left my keys?

- Paper: program + data + graphics + experiment logs + text + revisions + referee reports + presentations.
- Experiments have to be rerun, graphics replotted, papers rewritten.
- Use logs to know which parameters produced which data that produced which graph.
- And put them all in the same directory tree, or use sensible naming conventions.
Consider *literate* programming

- *Literate programming* means keeping program and document describing it and results in the same place.
- **SW**eave and **Knitr** integrate LaTeX and R in the same document.
  - Check availability for your favorite platform.
- Not the most popular way of writing papers.
- But check also [http://www.executablepapers.com/](http://www.executablepapers.com/)
Keep a balance between fashions and efficiency
Nurture your code
A moment of joy, a lifetime of grief

- Run tests periodically, or when there is a major upgrade of interpreter, upstream library or OS.
  - Can be automated.
    - See #6.

- Maintain a roadmap of releases
  - Remember this is free software, engage the community.

- Your research is intended for the whole wide world.
Publish, don't perish!
(no cats were harmed preparing this presentation)

Check me out at:

http://twitter.com/jjmerelo
http://goo.gl/OFou1

Any (more) questions?

See you in Evostar 2014, Baeza:
http://evostar.org!