

Doing Some Interesting Experiments with ECJ and MASON Toolkit

Presenter: Khaled

Computational Optimization and Innovation (COIN) Laboratory
Michigan State University

talukde1@msu.edu

April 28, 2017

Disclaimer

- ▶ Introduction to some new toolkits for EC and other scientific experiments.
- ▶ Not about any of my current work.
- ▶ Does not give any specific idea, but might give you some interesting insights into different orthogonal topics.

Agenda: organized as follows –

- ▶ A brief introduction to ECJ toolkit.
- ▶ An example research with ECJ toolkit: Meta-EA
- ▶ A brief introduction to MASON framework.
- ▶ An example research with MASON framework: HiTAB + Robotics
- ▶ Gluing ECJ with MASON.
- ▶ A simple experiment to demonstrate ECJ + MASON.
- ▶ A more sophisticated scientific experiments with ECJ + MASON.

ECJ: Evolutionary Computation using Java

- ▶ URL: <https://cs.gmu.edu/~eclab/projects/ecj/>
- ▶ Developed at the Evolutionary Computation Lab (ECLab) at George Mason University.
- ▶ First started as a Genetic Programming (GP) framework, and then extended.
- ▶ One of the most popular libraries cited/used by the EC community.
- ▶ Written in 100% Java.

ECJ: Evolutionary Computation using Java

ECJ 23
A Java-based Evolutionary Computation Research System

By Sean Luke, Liviu Panait, Gabriel Balan, Sean Paus, Zbigniew Skolicki, Rafal Kicinger, Elena Popovici, Keith Sullivan, Joseph Harrison, Jeff Bassett, Robert Hubley, Ankur Desai, Alexander Chircop, Jack Compton, William Haddon, Stephen Donnelly, Beenish Jamil, Joseph Zellbor, Eric Kangas, Faisal Abidi, Houston Mooers, James O'Beirne, Khaled Ahsan Talukder, Sam McKay, and James McDermott.

ECJ is a research EC system written in Java. It was designed to be highly flexible, with nearly all classes (and all of their settings) dynamically determined at runtime by a user-provided parameter file. All structures in the system are arranged to be easily modifiable. Even so, the system was designed with an eye toward efficiency.

ECJ is developed at George Mason University's [ECLab](#) Evolutionary Computation Laboratory. The software has nothing to do with its initials' [namesake](#), *Evolutionary Computation Journal*. ECJ's sister project is [MASON](#), a multi-agent simulation system which dovetails with ECJ nicely.

New Book!

Sean's got a free online text, [Essentials of Metaheuristics](#). Over 200 pages of goodness. Check it out!

Features

<p>General Features</p> <ul style="list-style-type: none"> • GUI with charting • Platform-independent checkpointing and logging • Hierarchical parameter files • Multithreading • Mersenne Twister Random Number Generators • Abstractions for implementing a variety of EC forms. <p>EC Features</p> <ul style="list-style-type: none"> • Asynchronous island models over TCP/IP • Master/slave evaluation over multiple processors, with support for generational, asynchronous steady-state, and coevolutionary distribution • Genetic Algorithms/Programming style Steady State and Generational evolution, with or 	<p>GP Tree Representations</p> <ul style="list-style-type: none"> • Set-based Strongly-Typed Genetic Programming • Ephemeral Random Constants • Automatically-Defined Functions and Automatically Defined Macros • Multiple tree forests • Six tree-creation algorithms • Extensive set of GP breeding operators • Grammatical Encoding • Push • Many pre-done GP application problem domains, including ant, regression, multiplexer, lawnmower, parity, two-box, edge <p>Vector (GA/ES) Representations</p>
--	--

Features

- ▶ A fairly complete GP research system.
- ▶ Set-based GP, Strongly Typed GP, ADF, ADM, Multi-tree (forests), Tree generation algorithms, all sorts of GP operators, GE, Push, lots of examples and benchmark problem sets.
- ▶ Other EC Algorithms: GA, ES, DE, PSO, EMO and CMAES etc.
- ▶ EC models: Steady State, Island Models, Co-evolution, Massively parallel evaluation etc.
- ▶ All kinds of EC related operators.
- ▶ Fairly good collection of benchmark problems.
- ▶ EMO: only NSGA-II and SPEA2, ZDT and DTLZ problems, no performance measure facility.

Why/Why not ECJ?

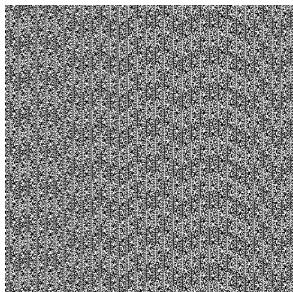
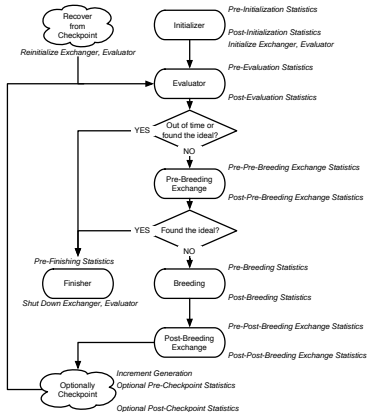


Figure: Pattern in
`java.util.Random`

- ▶ No particular reason.
- ▶ Java random number generator (`java.util.Random`) used to have a bug (fixed in 2009). ECJ has it's own correctly implemented RNG (Mersenne Twister), very efficient and fast.
- ▶ Covers almost all aspects in EC in general. There are other frameworks (i.e. *watchmaker*), but not as vast as ECJ.
- ▶ Strong emphasis on efficient code. ($O(N^2)$ vs. $O(\log N)$) – *therefore, sometimes it's quite hard to understand the code, steep learning curve.*
- ▶ Time consuming to implement a simple concept, but very easy to do complicated experiments.

ECJ Architecture



- ▶ Everything starts from a parameter file.
- ▶ Example run: `java ec.Evolve -file file.params`
- ▶ Objects are instantiated from parameter files.
- ▶ Parameters can be inherited.
- ▶ Parameters can form hierarchies.
- ▶ 300 page manual: <https://cs.gmu.edu/~ecclab/projects/ecj/docs/manual/manual.pdf>
- ▶ Show an example.

An Example Experiment: Meta-EA

- ▶ Meta-EAs: finding the best EA parameters for a particular algorithm to solve a specific problem.
- ▶ Digression: Given a specific FE budget, and a parallel computing resource, when it is worthwhile to run a Meta-EA? and how should we run it?
- ▶ Is it possible to smooth out the EA parameter space? Testing one meta-individual (EA parameter settings) multiple times or just once?
- ▶ Evolve the meta-individuals according to the mean fitness from multiple test.
- ▶ Look for the strategy that will most likely to give the best solution to a particular problem.
- ▶ Massively parallel evaluation implemented using ECJ to run over a 128 node cluster, **14.2 million evolutionary runs** for different EAs and benchmark problems.

Meta-EA: Sample Result

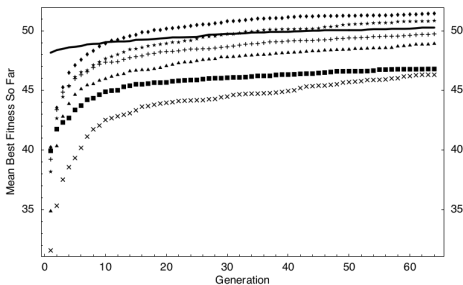


Figure 3: Meta-level mean best-fitness-so-far for the Lennard-Jones problem (DE, maximization).
 ♦ 1 Test ★ 2 Tests + 4 Tests ▲ 8 Tests
 × 16 Tests ■ Random Search — Best Discovered Setting

- Turns out that smoothing of the parameter space is not very useful if the budget is fixed.
- Specific example: 1, 2, 4, 8, 16 tests vs. 128, 64, 32, 16, 8 meta level pop. size.
- It's better to try 128 different parameter settings once than 8 parameter settings 8 times to get the “best solution discovering” parameter.

MASON: **M**ulti-**A**gent **S**imulator of **N**eighborhoods... or **N**etworks... or something...

- ▶ Fast discrete event simulator developed at GMU Autonomous Robotics Lab and Center for Social Complexity, Krasnow Institute of Advanced Studies.
- ▶ Initial objective was to do black-box simulation optimization with ECJ, but later came out as an independent project.
- ▶ More popular than ECJ, especially in the simulation science community.
- ▶ Also written in Java.

MASON: Multi-Agent Simulator of Neighborhoods... or Networks... or something...

MASON

Features Download Help Applet & Screenshots Projects Extensions Other Simulators

MASON is a fast discrete-event multiagent simulation library core in Java, designed to be the foundation for large custom-purpose Java simulations, and also to provide more than enough functionality for many lightweight simulation needs. MASON contains both a model library and an optional suite of visualization tools in 2D and 3D.

MASON is a joint effort between George Mason University's Evolutionary Computation Laboratory and the GMU Center for Social Complexity, and was designed by Sean Luke, Gabriel Catalan Balan, Keith Sullivan, and Liviu Panait, with help from Claudio Cioffi-Revilla, Sean Paus, Keith Sullivan, Daniel Kuebrich, Joey Harrison, and Ankur Desai.

MASON Stands for Multi-Agent Simulator Of Neighborhoods... or Networks... or something...

MASON Features

- 100% Java (1.3 or higher)
- Fast, portable, and fairly small
- Models are completely independent from visualization, which can be added, removed, or changed at any time
- Models may be checkpointed and recovered, and dynamically migrated across platforms
- Can produce results that are identical across platforms
- Models are self-contained and can run inside other Java frameworks and applications
- 2D and 3D visualization
- Can generate PNG snapshots, Quicktime movies, charts and graphs, and output data streams

Download MASON

Features

- ▶ Can represent continuous, discrete, or hexagonal 2D, 3D, or Network data, and any combination of it.
- ▶ Full-fledge GUI for visualization, probing and monitoring simulation activities.
- ▶ The Console gives access to model data, plays, stops, pauses, and steps the simulation, checkpoints and recovers models, and performs other tasks.
- ▶ The user can also view and change per-object model data by selecting objects in the visualization displays and bringing up their inspectors.
- ▶ Very fast and efficiently written (like ECJ)
- ▶ It also comes with a 300 page manual (<http://cs.gmu.edu/~eclab/projects/mason/manual.pdf>)

Examples

Click Thumbnail for Image



HeatBugs is a classic multiagent example popularized by the **Swarm** multiagent simulation toolkit.



Ants is an ant colony foraging simulation using two pheromones.



Solar System (Tutorial 6) is a simple (indeed simplistic) demo of planets orbiting the Sun.



Balls and Bands (Tutorial 5) simulates Hooke's Law with balls connected by rubber bands of different strengths.



Woims is another **Boids** example. [Quicktime Movie.](#)



Keepaway simulates a Keep-Away Soccer game. At present the robots have stupid "go to the ball and kick it" behaviors.



Conway's Game of Life, a classic cellular automata, here as a simple demo seeded with the b-heptomino.



The LightCycles Game is a discretized version of the old game popularized by the movie Tron.



Bouncing Particles is a tutorial



HeatBugs shown in wireframe 3D. [Quicktime Movie.](#)



Flockers is an implementation of Craig Reynolds' **Boids** algorithm. 2000 flockers shown.



HexaBugs is HeatBugs on a hex grid, inspired by the **RePast** simulation toolkit. (Shown zoomed in).



Mouse Traps and Ping Pong Balls When a ball hits a trap, it pops the ball back up, plus another ball. [Quicktime Movie.](#)



Woims in 3D shows the Woims in 3D space. [Quicktime Movie.](#)



L-system is a deterministic, bracketed context-free L-system generator with basic turtle graphics.



Cooperative Observation implements a k-means clustering method for cooperative observation of randomly-moving targets.



MAV is a basic platform for simulating simple Micro-air Vehicle behaviors.

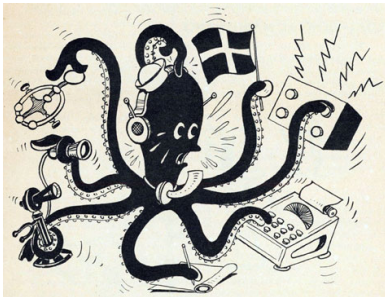


Virus is a simple platform for

GAN STATE
UNIVERSITY

COIN
Computational Optimization and
Innovation Laboratory
Michigan State University

Gluing ECJ with MASON



- ▶ Scientists make **models**.
- ▶ Complicated models (i.e. ABM) requires a large number of **multiple inter-dependent parameters** to be tuned manually – prohibitive.
- ▶ Can we automate this ?

Preliminaries

Model Parameter Types

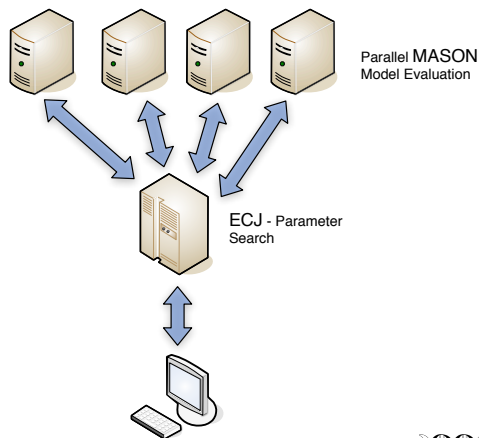
- ▶ Known Parameters – whose settings already known for a fact
- ▶ Inferable Parameters – whose settings should be set in a certain way (*according to a particular theory*)
- ▶ Insensitive Parameters – over whose settings the model is expected to be insensitive
- ▶ **Unknown Parameters – on which an experimenter has no idea/control over.** (*most interesting ?*)

Goals

- ▶ To produce a certain kind of output which is predicted by a theory, or
- ▶ to match and validate against known real-world results.

Parallel Evolutionary Optimization – A Bird's-Eye View

- ▶ Extremely expensive – need to optimize the model by running it many times.
- ▶ Ought to be done in parallel.



A Very Simple GP Experiment: Evolving PacMan Behavior

- ▶ MASON → **Behaviour** specifications
- ▶ ECJ → **Optimization** by **Evolution**
- ▶ MASON + ECJ → Evolving Optimized Behaviour
- ▶ **Target:** The game of PacMan

An Evolved Pac Behaviour



Show Demo

MICHIGAN STATE
UNIVERSITY



A More Sophisticated Example: Rebeland

- ▶ Built on top of MASON to simulate socio-economic instability.
- ▶ Developed by the Center for Social Complexity, Krasnow Institute of Advanced Study, GMU and Smithsonian Institution.
- ▶ Simulate a government policies according to canonical political science and theory from economics
- ▶ Experimenter can start the simulation with a set of initial parameters and check the validity of a theory by gathering the statistical information from the simulation agents.
- ▶ Includes historical and geoglogical data to model resources for the population.

RebeLand Model

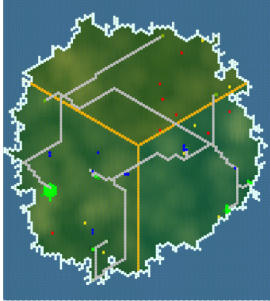


Fig. 1. Map of RebeLand Island showing its main natural and social features. *Legend:* Cities are shown in green, natural resources in yellow, and rebel and government forces in red and blue, respectively. Roads and provincial boundaries are in gray and yellow, respectively. Physical topography is shown on a green-tone scale and the island is surrounded by ocean. *Source:* Prepared by the authors.

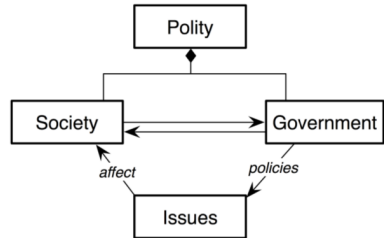
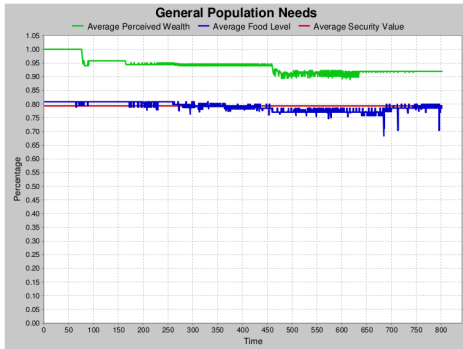
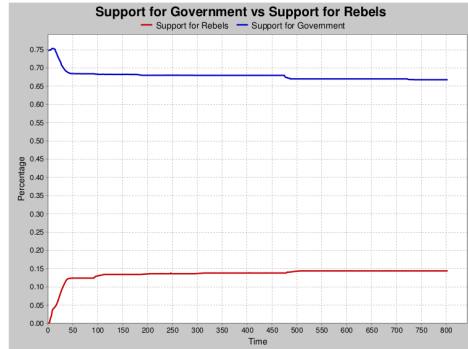


Fig. 2. High-level UML class diagram of a simple polity such as that represented in RebeLand. Government manages public issues through policies, as detailed in Figure 2. *Source:* Cioffi-Revilla 2009.

RebeLand Simulation Snapshot 1

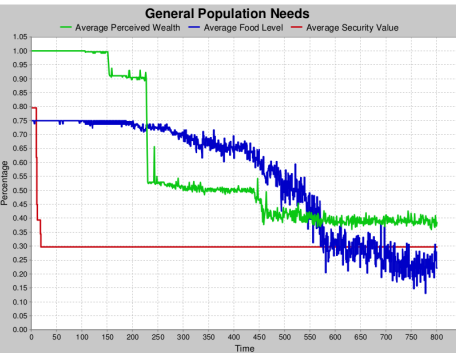


(a) General population needs under a stable scenario.

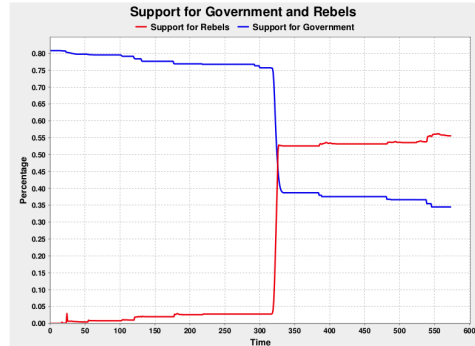


(b) Support for government and support for rebels under a stable scenario.

RebeLand Simulation Snapshot 2



(a) General population needs under an unstable scenario.



(b) Support for government and support for rebels under an unstable scenario.

Parameters/Knobs

Unknown Parameters to Set (*all scaled to 0...1*)

- ▶ State Corruption Rate
- ▶ State Tax Rate
- ▶ Maximum State Reserve
- ▶ Minimum State Reserve
- ▶ Minimum Spending on Populace
- ▶ Maximum Police Per Capita
- ▶ Initial Reserve Army Ratio
- ▶ Standing Army Size
- ▶ State Attack Interval

Details You Don't Care About

Optimization Algorithm

NSGA-II, 6000 evaluations.

Testing (i.e. *Generalization*)

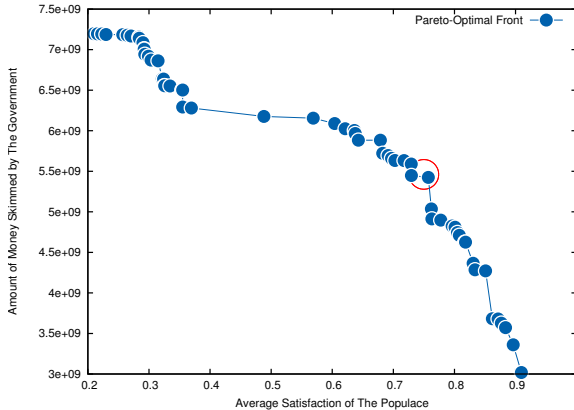
Individual *Parameter Settings* are tested by running RebeLand on MASON 8 times – mean results were considered.

Parallelization

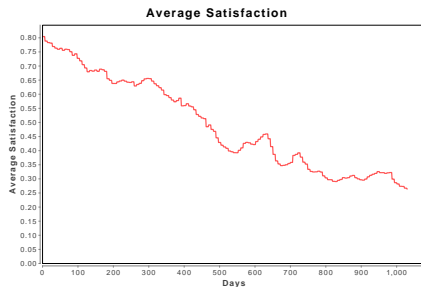
Master-Slave Evaluation, 30 Slave Units on Hydra

Total evaluation time: about 1 hour

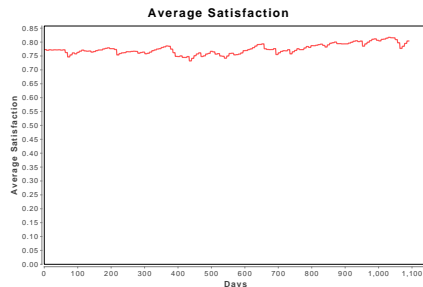
Results: The *Pareto Front*



Results: Population Satisfaction



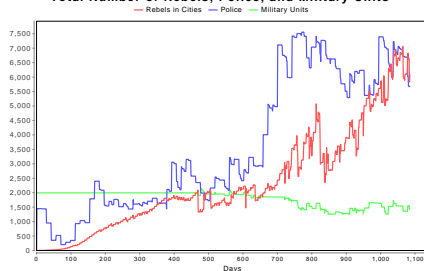
Before Optimization
(Original Parameters)



After
Optimization

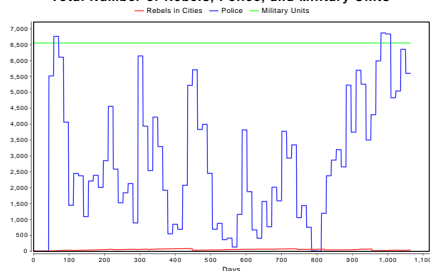
Results: The Overall Situation

Total Number of Rebels, Police, and Military Units



Before Optimization
(Original Parameters)

Total Number of Rebels, Police, and Military Units



After
Optimization

Results: Some (interesting) numbers

- ▶ State's corruption rate – **86%**
- ▶ State's tax rate – **77%**
- ▶ State's maximum reserve rate – 84%
- ▶ State's minimum reserve rate – 77%
- ▶ Minimum benefit share to the populace (i.e. public spending) – **98%**
- ▶ Maximum number of police force per capita – **31%**
- ▶ Initial reserve army ratio – 0.01%
- ▶ Standing army size – 0.06%
- ▶ State's attack frequency (to suppress rebels) – **78%**

Conclusions

Lesson ?

It's **fine** to run a **corrupt** government and keep its people **happy** – *only* if you have a **high** public spending, **large police force** and a very **frequent attack** on rebels.

Utopia ?

These results are “interesting” – i.e. *communist dictatorship* ?

Revelations ?

Does this reveal cracks in the *RebeLand* model semantics? Or a bug in the code?

Summary



- ▶ ECJ and MASON can be used to conduct many interesting experiments.
- ▶ Surely, symbiosis between MASON and ECJ – has a lot more to offer.
- ▶ Building a unified/standardized APIs to help plugging ECJ capabilities into MASON models – could just be a tip of the iceberg.

Questions?

Discussions?