Search-based approaches to improving the energy consumption of Java programs

Sandy Brownlee

University of Stirling

sbr@cs.stir.ac.uk

Content

- Motivation: why target energy?
- Opacitor: a tool to measure energy on the JVM
- Case study 1: quicksort pivots
- Case study 2: multi-objective tuning of MLPs
- Case study 3: OO-GI on collections
- Summary

Collaborators

Nathan Burles (IBM)

Jerry Swan (NNAISENSE)

Ibttihal Israr (MSc student, Univ. Stirling)

Funded by DAASE Project (EPSRC EP/J017515/1)

Search-Based Energy Optimization of Some Ubiquitous Algorithms

Alexander Edward Ian Brownlee, Nathan Burles, and Jerry Swan

Abstract—**Reducing computational energy consumption is of growing importance, particularly at the extremes (i.e., mobile devices and datacentres). Despite the ubiquity of the Java virtual machine (JVM), very little work has been done to apply search-based software engineering (SBSE) to minimize the energy consumption of programs that run on it. We describe OPACITOR, a tool for measuring the energy consumption of JVM programs using a bytecode level model of energy cost. This has several advantages over time-based energy approximations or hardware measurements. It is**

other end of the scale, the repeated execution of specific subroutines by server farms offers the potential for considerable energy saving via the identification of energy-intensive 'hotspots' in the code. The electricity consumption by servers was estimated at between 1.1 and 1.5% of global electricity production in 2010 [6], with energy consumption reaching 50-100% of the purchase cost of the hardware over its lifetime [7]. These facts motivate

- Brownlee, A. E. I., Burles, N. and Swan, J. (2017). Search-based energy optimization of some ubiquitous algorithms. IEEE Transactions on Emerging Topics in Computational Intelligence, vol 1, issue 3, pp. 188-201. DOI:10.1109/TETCI.2017.2699193

Motivation

- Energy consumption matters!
- Data Centres
  - 191 TWh in 2018 (1% of global electricity demand)[1,2]
- Smart phones
  - CPU could reach 2.5x power consumption of screen or 3G hardware on an S3[3]

1 https://www.iea.org/reports/tracking-buildings/data-centres-and-data-transmission-networks
2 Masanet, E. R. et al. (2018), *"Global Data Center Energy Use: Distribution, Composition, and Near-Term Outlook"*
3 A. Carroll and G. Heiser (2013) "The systems hacker's guide to the Galaxy: Energy usage in a modern smartphone"

Energy Measurement Approaches

- Direct measurements
- CPU time as a proxy
- Model-based

Opacitor

- Modified version of OpenJDK
- Counts bytecodes as they are executed
- Applied to linear model based on model of Hao et al.[4]

```
/home/sbr/SharedLibs/hamcrest-core-1.3.jar:/home/sbr/eclipse/java-neon/eclipse/c
onfiguration/org.eclipse.osgi/213/0/.cp/:/home/sbr/eclipse/java-neon/eclipse/con
figuration/org.eclipse.osgi/212/0/.cp/ opacitor.junit.TestClass  test2.txt 0 200
00

Histogram of 9343254 executed bytecodes:

       absolute  relative  code    name
-------------------------------------------------------------------------
         566754     6.07%    db    fast_aload_0
         496107     5.31%    df    fast_iload
         437509     4.68%    b6    invokevirtual
         328505     3.52%    dd    fast_aaccess_0
         299949     3.21%    1b    iload_1
         263830     2.82%    19    aload
         259636     2.78%    dc    fast_iaccess_0
         244025     2.61%    1c    iload_2
         241322     2.58%    2b    aload_1
         232308     2.49%    a7    goto
         230730     2.47%    1d    iload_3
         218938     2.34%    36    istore
         212900     2.28%    b7    invokespecial
         201410     2.16%    84    iinc
         178189     1.91%    9a    ifne
```

4 S. Hao, D. Li, W. G. Halfond, and R. Govindan, “Estimating mobile application energy consumption using program analysis,” in *Proc. Int. Conf. Softw. Eng.* San Francisco, CA, USA: IEEE, 2013, pp. 92–101

Opacitor

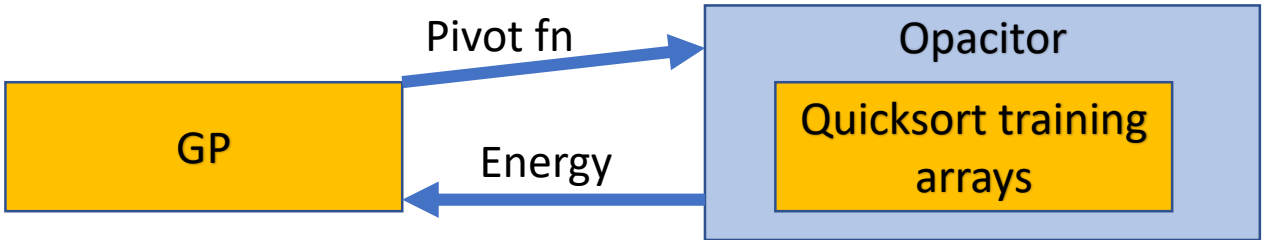
- Advantages:
  - Distinctions can be made between very similar programs
  - Unaffected by anything else executing
    - can be parallelised, or executed simultaneously with other programs
  - No need for warmup
  - Hao's model was within 10% of hardware measurements for a set of mobile applications from the Google Play store
- But:
  - In practice, energy use probably follows a conditional distribution rather than remaining constant per-bytecode
  - Trade-off between generality of a static model (Opacitor) and the device-specific labour required for dynamic measurement

Case studies: quicksort

- GP to generate quicksort pivot functions
- Average case $O(n \log n)$, worst case $O(n^2)$
- Highly dependent on pivot function
  - ideal is median
- Common heuristics:
  - Middle index
  - Random index
  - Sedgewick (median of first/middle/last elements)

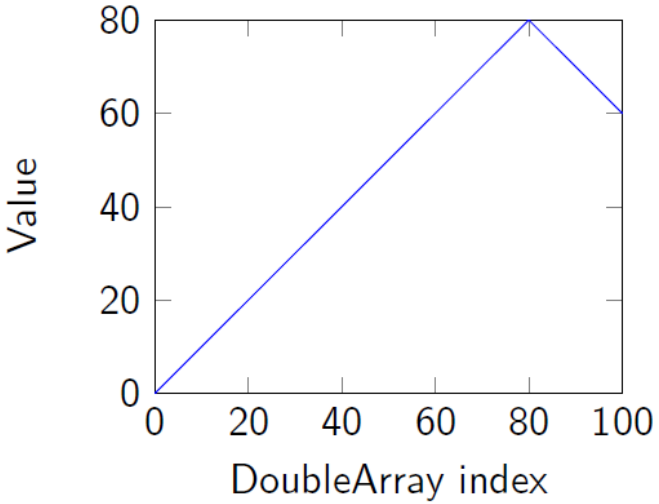
Case studies: quicksort

- Choose ***r*** samples at random from the array; median of these is the pivot
- ***r*** is determined by a function generated using GP
  - terminals l (input array length) and d (recursion depth)
  - operators + - * /
- GP population 100; 200 generations; initial tree depth 2; maximum tree depth of 4
- Objective: minimise energy for quicksort on training set of 1000 arrays; each array 100 elements with “pipeorgan” distribution

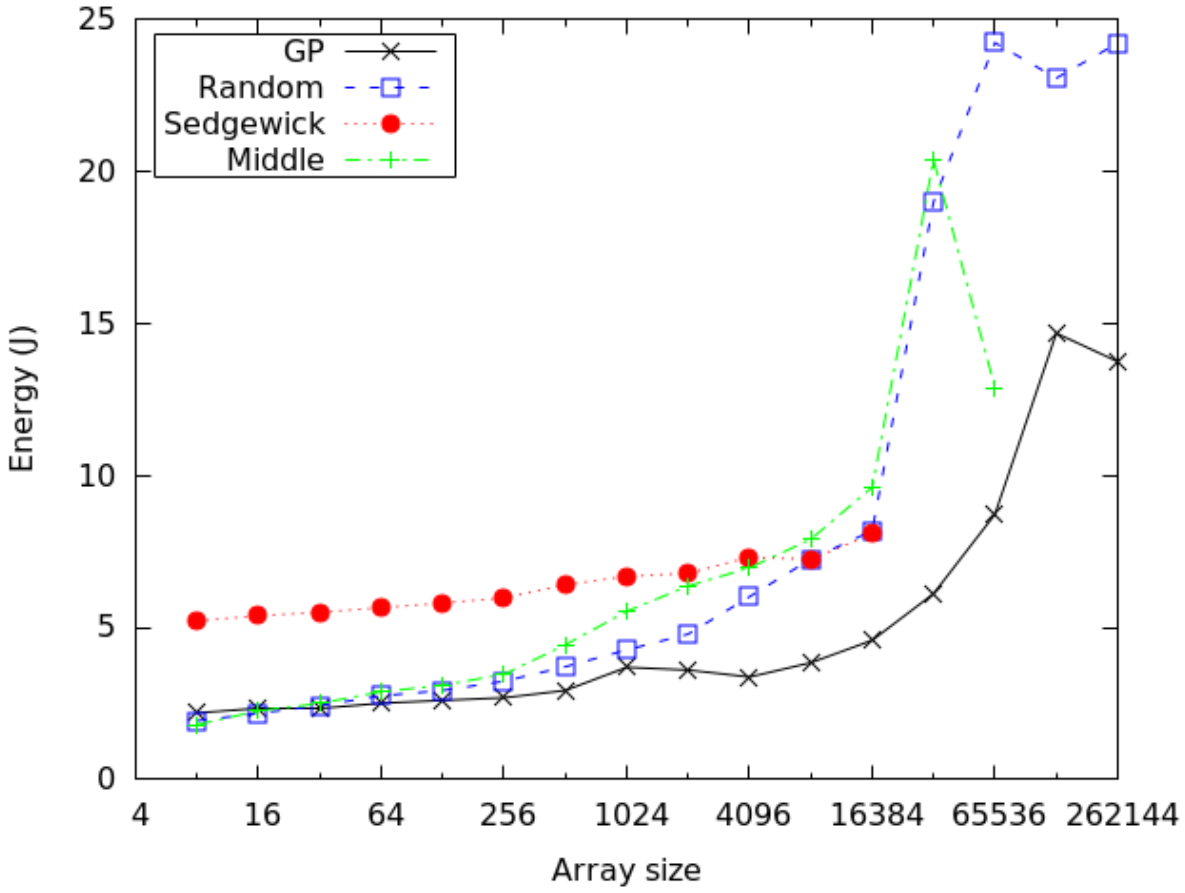


Experiment

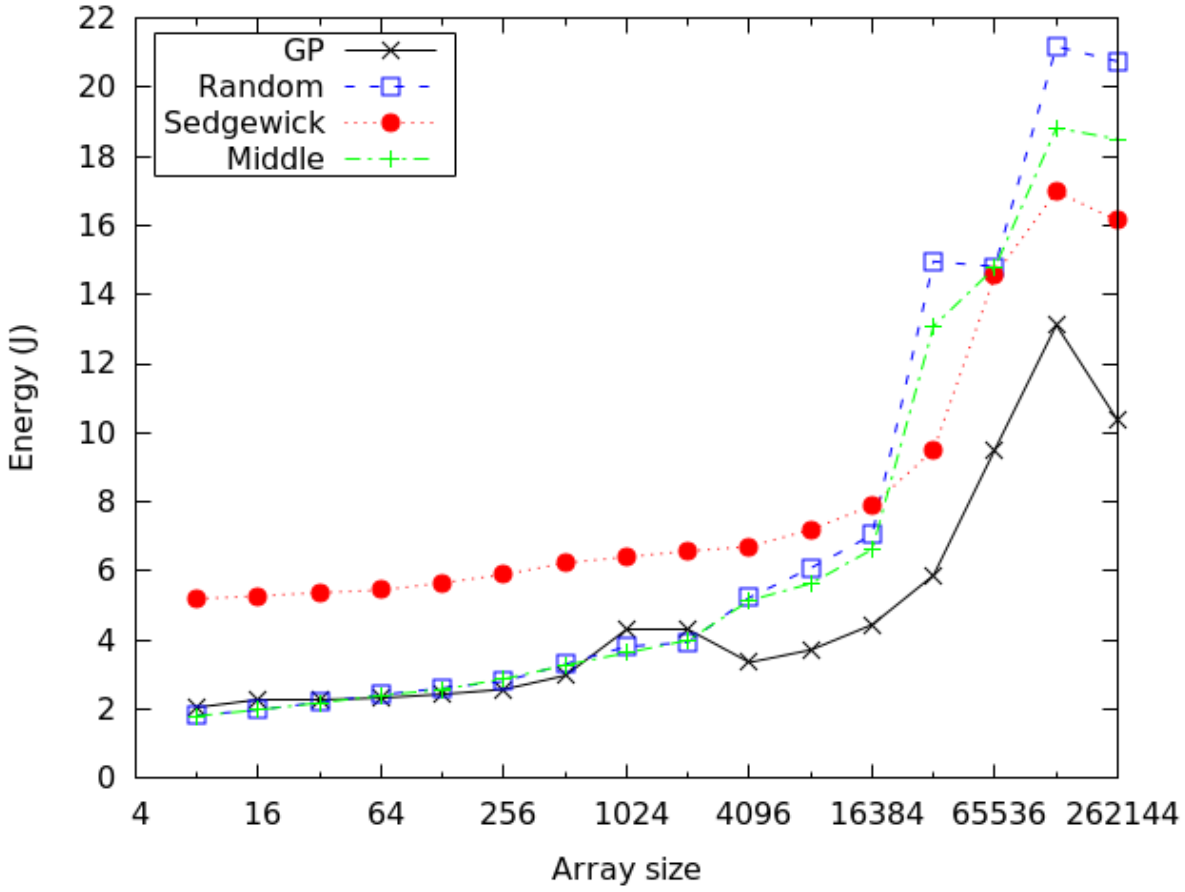
- Apply middle/random/Sedgewick/GP pivots to:
  - 1000 pipeorgan arrays of lengths from 8 to 262144
  - 1000 random arrays of lengths from 8 to 262144
  - (both repeated 100 times with different sets)



Results on pipeorgan arrays



Results on random arrays



Case Studies: MLP

- Tuning the hyperparameters of a multilayer perceptron
- Trade-off a functional software property **(error rate)**, against a nonfunctional property **(energy)**
- Target application is WEKA

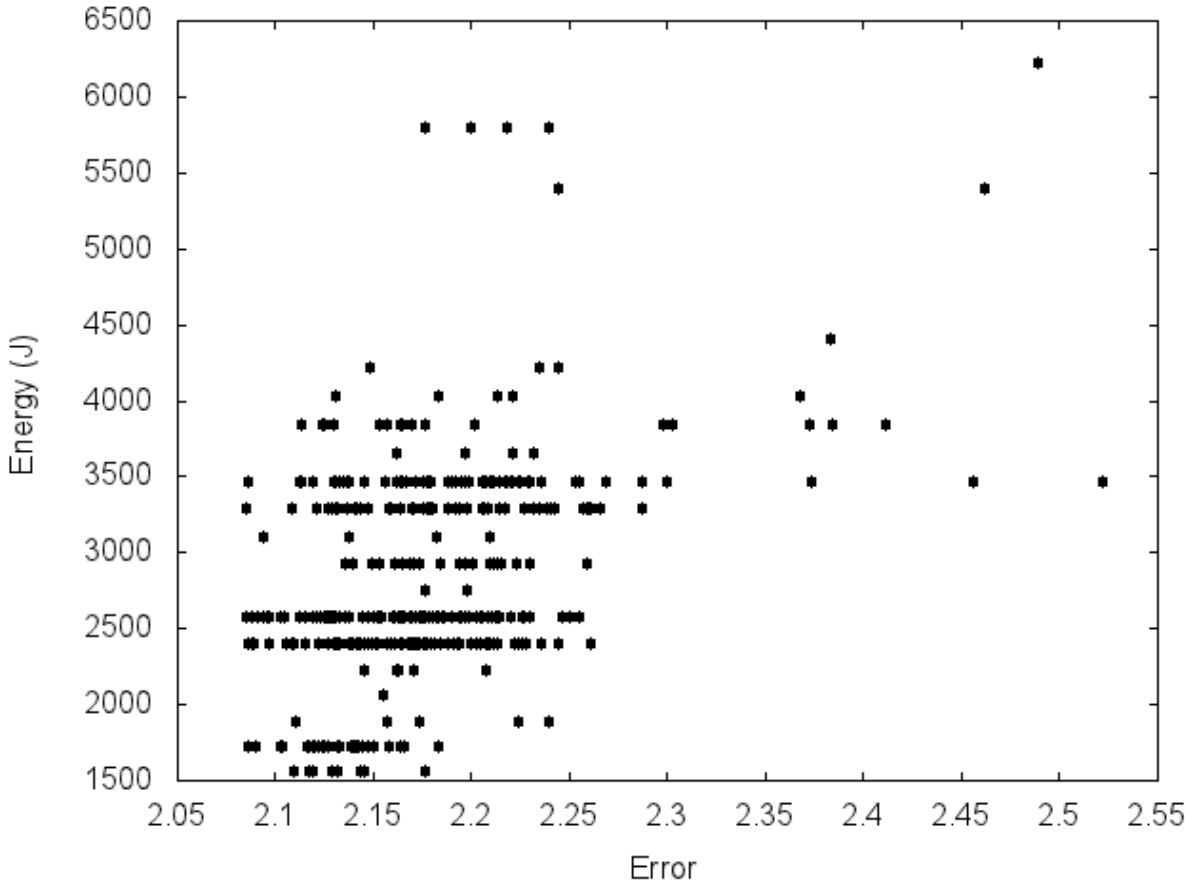
Case Studies: MLP

Label	Parameter	Type	Range
a	AF Parameter 0	Continuous	0,1
b	AF Parameter 1	Continuous	0,1
c	AF Parameter 2	Continuous	0,1
d	AF Parameter 3	Continuous	0,1
h	Neurons count in hidden layer	Integer	2,30
l	Backpropogation learning rate	Continuous	0,1
α	Backpropogation momentum	Continuous	0,1

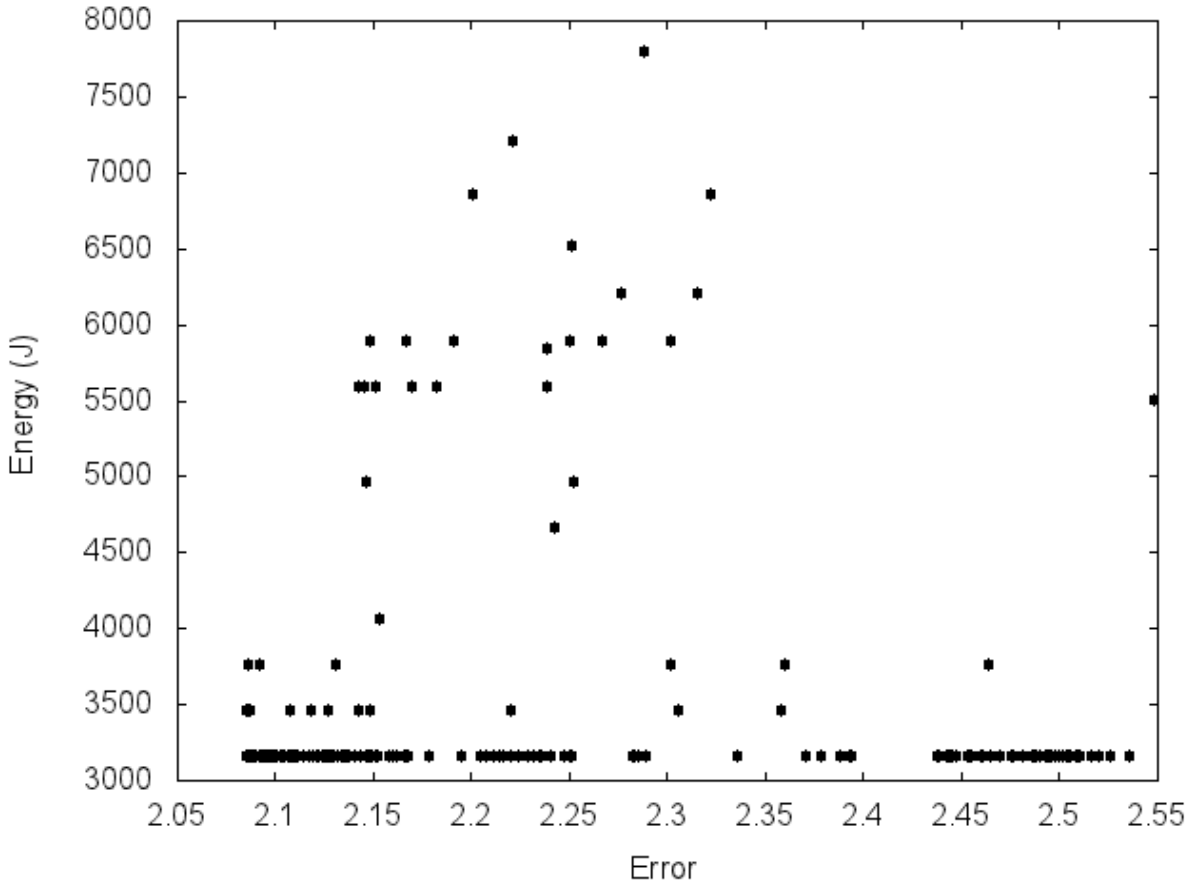
Experiment

- NSGA-II for find parameters; population 10; 400 evaluations; SBX crossover (rate 0.9, dist index 20.0); polynomial real mutation (rate 1/7, dist index 20.0); binary tournament selection
- Objectives (both average of 5 train/valid repeats):
  - Energy from Opacitor
  - Error on validation data
- Well known UCI data sets: Pima Indians, Glass, Ionosphere, Iris; split to 2/3 training, 1/3 validation
- Two runs, with energy for:
  - training only
  - training + 1000 validations

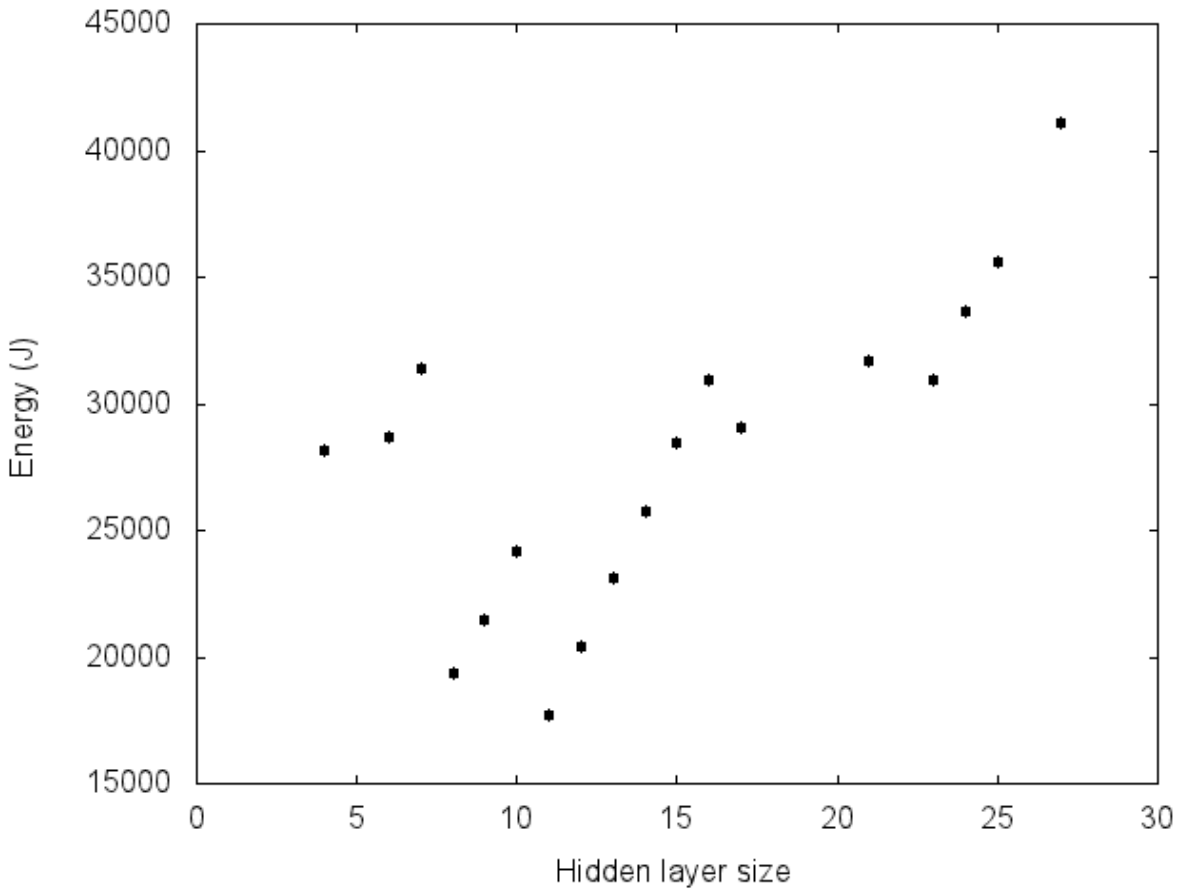
Results: training (Glass)



Results: training+validation (Glass)



Results: HLS, training (Glass)



Hurray!

- We minimised both energy and error rate
- Some further digging (more datasets; including number of epochs, minibatch size) suggests a similar picture
- However, adding more hidden layers seems to introduce some kind of trade-off as we might have expected…

MLP Experiment 2

- With the UCI breast cancer data set (same experiment, but with multiple hidden layers…)

Energy (joules)	Error	Hidden Layers	Layer sizes
6693	0.246	3	4,5,5
2092	0.263	3	2,2,2
11396	0.351	4	8,3,1,9
2115	0.368	4	2,2,2,2

Case Study: OO-GI

- Similar idea to Darwinian Data Structures, and SEEDS framework[5], developed independently
- Minimising just energy
- Exchanging collections drawn from three libraries:
  - Java 8 Collections
  - Google Guava 18
  - Apache Commons 4
- Targeting classes in Guava and Apache Commons

[5] I. Manotas, L. Pollock, and J. Clause, "SEEDS: A software engineer's energy-optimization decision support framework," in *Proc. Int. Conf. Softw. Eng*. Hyderabad, India: ACM, 2014, pp. 503–514.

Case Study: OO-GI

- Look for:
  - constructors (e.g. new HashMap<>());
  - Factory classes (e.g. Maps.newHashMap());
  - static creator methods (e.g. ImmutableList.of()).
- Find most specific abstract supertype (earliest of):
  - java.util.SortedMap
  - java.util.Map
  - java.util.SortedSet
  - java.util.Set
  - java.util.List
  - com.google.common.collect.Multimap
  - com.google.common.collect.Multiset
  - java.util.Collection

Case Study: OO-GI

- Targets:

Class for modification	Variation points	Search space
com.google.common.collect. ArrayListMultimap	3	2 738
com.google.common.collect. ImmutableMultimap	5	674 325
com.google.common.collect. LinkedListMultimap	6	7 513 072
org.apache.commons.collections4.map. PassiveExpiringMap	3	50 653
org.apache.commons.collections4.set. ListOrderedSet	4	38 416
org.apache.commons.collections4.bidimap. DualHashBidiMap	6	2 565 726 409

Experiment

- Energy was measured for each class over 10,000 repeat runs of a test program
  - This called all unit tests for the class, and exercised methods not covered by the unit tests using randomly generated data
- GA had population size 500; 100 generations; single-point crossover (rate 75%); one-point mutation (rate 50%); binary tournament selection; 5% elitism.
- 30 repeat runs for each target class

Results

Target class	Measure	GA		Original			Independent Exhaustive		
		J	evals	J	p	e	J	p	e
`ArrayListMultimap`	excl.	**260.70**	2455 (10.34%)	260.90	–	–	**260.70**	–	–
	incl.	**21.23** σ1.00		23.39 σ1.09	<.001	0.92	**21.23** σ1.00	–	–
`ImmutableMultimap`	excl.	**224.70**	10 035 (98.5%)	300.70	–	–	275.26	–	–
	incl.	**12.22** σ0.73		15.75 σ1.79	<.001	0.97	13.09 σ1.09	<.001	0.74
`LinkedListMultimap`	excl.	**370.28**	18 092 (99.76%)	370.29	–	–	370.29	–	–
	incl.	**19.82** σ1.25		32.95 σ2.79	<.001	1.00	26.11 σ0.82	<.001	1.00
`PassiveExpiringMap`	excl.	**108.34**	11 014 (78.26%)	193.60	–	–	158.94	–	–
	incl.	**11.54** σ1.52		17.06 σ0.65	<.001	1.00	14.74 σ1.37	<.001	0.94
`ListOrderedSet`	excl.	**70.83**	8470 (77.95%)	70.84	–	–	**70.83**	–	–
	incl.	**17.50** σ1.71		22.96 σ2.53	<.001	0.95	**17.50** σ1.73	–	–
`DualHashBidiMap` (best case)	excl.	**133.22**	23 646 (99.99%)	133.73	–	–	133.49	–	–
	incl.	**10.23** σ1.05		16.56 σ0.90	<.001	1.00	11.61 σ0.90	<.001	0.84
`DualHashBidiMap` (worst case)	excl.	**133.38**	23 646 (99.99%)	133.73	–	–	133.49	–	–
	incl.	**10.99** σ1.14		16.56 σ0.90	<.001	1.00	11.61 σ0.90	<.001	0.68

Results

- GA matched best found by exhaustive search (where performed), but in far fewer evaluations

Summary

- Opacitor: a tool to measure energy on the JVM
- Quicksort: a hyper-heuristic, tuning energy performance for given distributions of input data
- MLP: parameter tuning for energy; trading off functional properties with energy (and not dependent on a test suite!)
- OO-GI: exploiting the OO design to explore different implementations
- Caveats:
  - Model not perfect
  - Quicksort/OO-GI assumes adequate tests are available!

- sbr@cs.stir.ac.uk www.cs.stir.ac.uk/~sbr