Hot Off the Press!

Solving Uncompromising Problems with Lexicase Selection

in *IEEE Transactions on Evolutionary Computation*

Thomas Helmuth, Lee Spector, and James Matheson
Hampshire College & University of Massachusetts, Amherst

GECCO

Madrid, Spain
July 11-15, 2015

Genetic and Evolutionary Computation Conference

2015

acm

Outline

- Lexicase selection
- Modal and uncompromising problems
- Four problems
- Experimental results
- Conclusions

Selection

- In genetic programming, selection is typically based on average performance across all test cases (sometimes weighted, e.g. with "implicit fitness sharing")
- In nature, selection is typically based on sequences of interactions with the environment

Lexicase Selection

- Emphasizes individual test cases and combinations of test cases; not aggregated fitness across test cases
- Random ordering of test cases for each selection event
- Can **DRAMATICALLY** enhance the power of genetic programming to solve problems

Lexicase Selection

To select single parent:

1. Shuffle test cases
2. First test case – keep best individuals
3. Repeat with next test case, etc.

Until one individual remains

The selected parent may be a specialist in the tests that happen to have come first, and may or may not be particularly good on average

Modal Problems

- Require successful programs to do something qualitatively different in different circumstances
- “Circumstances” vary across fitness cases
- How many modes? How are they detected? May not be obvious in advance
- Many software design problems (among others) are modal


STAPLES
SPL-110
12345678
%
OFF
ON/C
MRC
M-
M+
CE
7
8
9
÷
4
5
6
×
1
2
3
−
0
•
=
+

Uncompromising Problems

- Any acceptable solution must perform as well on each test case as it is possible to perform on that test case
- Not acceptable for a solution to perform sub-optimally on any one test case in exchange for good performance on others
- Many software design problems (among others) are uncompromising

Potential

- Not only for modal or uncompromising problems
- Other uses of selection in genetic programming
- Other forms of evolutionary computation with case-like assessment
- More to be done, e.g. for problems with continuous errors

Related Work

- Multi-objective evolution (generally assumes objectives, which may not be factored by input, are known in advance)
- Multi-modal problems (generally refers to problems with multiple global optima)
- Lexicographic ordering in selection (but here we order fitness cases, in random order)
- Ensemble methods (but here we seek a single program perhaps with some code used for multiple modes)

Experiments

- Problems
  - Finding discriminator terms in finite algebras
  - Designing digital multipliers
  - Symbolic regression of the factorial function
  - Automatic programming of "wc" (word count)
- Genetic programming systems
  - Koza-style tree-based GP
  - PushGP
- Selection
  - Lexicase
  - Tournament (various sizes)
  - Implicit Fitness Sharing (various tournament sizes)

Finite Algebras

$\mathbf{A}_1$ $*$	0	1	2
0	2	1	2
1	1	0	0
2	0	0	1

$\mathbf{A}_2$ $*$	0	1	2
0	2	0	2
1	1	0	2
2	1	2	1

$$t(x,y,z) = \begin{cases} x \text{ if } x \neq y \\ z \text{ if } x = y \end{cases}$$

Digital Multiplier

- 3 bits x 3 bits => 6 bits

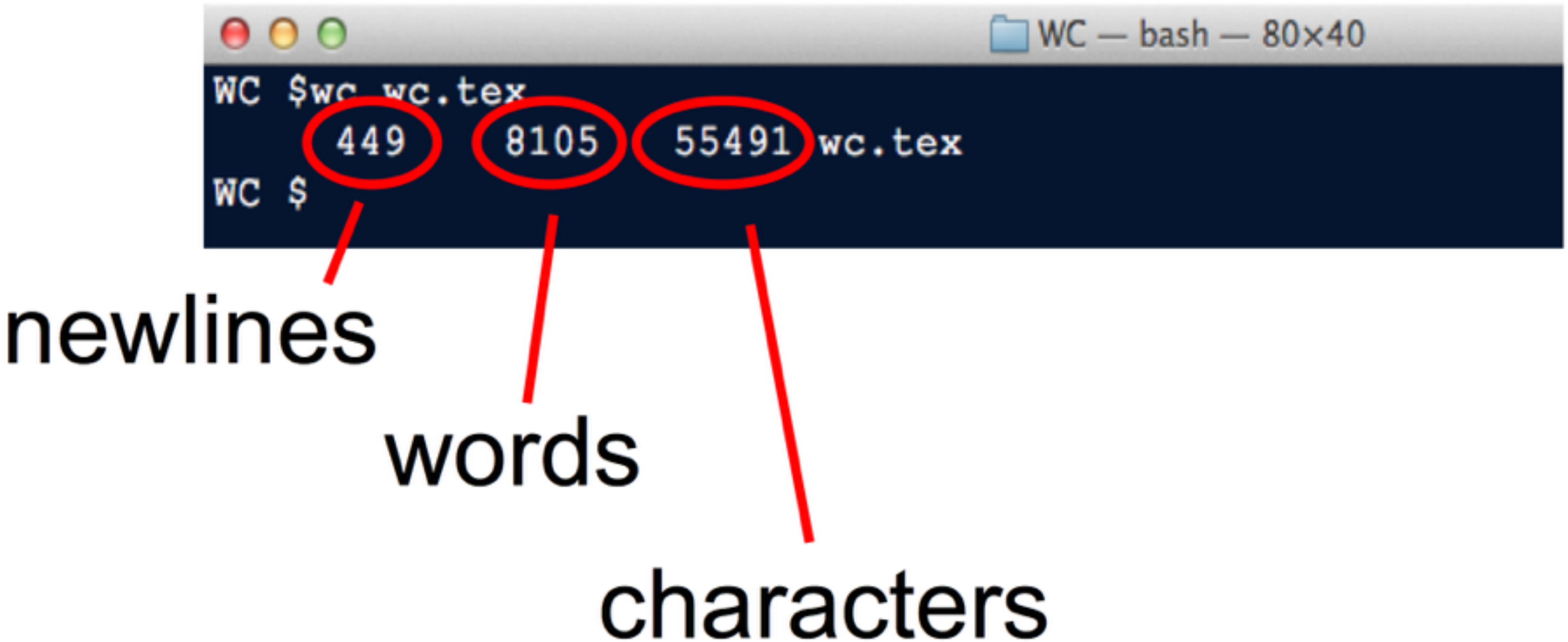
A LIST OF THE PUSH INSTRUCTIONS USED IN OUR DIGITAL MULTIPLIER EXPERIMENTS. FOR THE n-BIT DIGITAL MULTIPLIER PROBLEM, THERE ARE $2n$ INPUT INSTRUCTIONS AND $2n$ OUTPUT INSTRUCTIONS.

Boolean Stack	*and*, *or*, *xor*, *invertFirstThenAnd*, *dup*, *swap*, *rot*
Input/Output	$in1$, ..., $in2n$, $out1$, ..., $out2n$

Factorial

- Inputs 1!=1 to 10!=3628800
- Various forms of normalization for non-lexicase methods
- Instructions for integers, booleans, execution stack (for conditional branches and recursion)
- No high-level Push instructions that allow for trivial solutions

wc


WC — bash — 80×40
WC $wc wc.tex
449 8105 55491 wc.tex
WC $
newlines
words
characters

wc Test Cases

- 0 to 100 character files
- Random string (200 training, 500 test)
- Random string ending in newline (20 training, 50 test)
- Edge cases (22; empty string, multiple newlines, etc.)

Instructions

- General purpose
- I/O
- Control flow
- Tags for modularity
- String, integer, and boolean
- Random constants

Input	file_readchar, file_readline, file_-EOF, file_begin
Output	output_charcount, output_wordcount, output_linecount
Exec	exec_pop, exec_swap, exec_rot, exec_dup, exec_yank, exec_yankdup, exec_shove, exec_eq, exec_stack-depth, exec_when, exec_if, exec_-do*times, exec_do*count, exec_-do*range, exec_y, exec_k, exec_s
Tag ERCs	tag_exec, tag_integer, tag_string, tagged
String	string_split, string_parse_to_chars, string_whitespace, string_contained, string_reverse, string_concat, string_take, string_pop, string_-eq, string_stackdepth, string_rot, string_yank, string_swap, string_-yankdup, string_flush, string_-length, string_shove, string_dup
Integer	integer_add, integer_swap, integer_-yank, integer_dup, integer_yankdup, integer_shove, integer_mult, inte-ger_div, integer_max, integer_sub, integer_mod, integer_rot, integer_-min, integer_inc, integer_dec
Boolean	boolean_swap, boolean_and, boolean_-not, boolean_or, boolean_frominte-ger, boolean_stackdepth, boolean_dup
ERC	Integer from [-100, 100] {"\n", "\t", "␣" } {x|x is a non-whitespace character}

Implicit Fitness Sharing

- Scale errors per case based on population-wide error
- Non-binary version

$$f_{NBIFS}(i) = \sum_{t \in T} \frac{f(i,t)}{\sum_{i' \in P} f(i',t)}$$

Push

- Designed for program evolution
- Data flows via stacks, not syntax
- One stack per type:
  integer, float, boolean, string, code, exec, vector, ...
- Rich data and control structures
- Minimal syntax:
  program → instruction | literal | (program*)
- Uniform variation, meta-evolution

Parameters

Problem	FA	DM	Fact	wc
System	Tree	Push	Push	Push
Runs Per Condition	100	100	100	200
Number of Test Cases	27	64	10	242
Population Size	1000	5000	1000	1000
Max Generations	100	4000	500	300
Max Program Size	1000	1000	500	1000
Max Initial Program Size	-	400	100	400
Expected Initial Program Size	50	-	-	-
Max Initial Program Depth	20	-	-	-
Expected Mutation Code Size	10	-	-	-
Max Mutation Code Depth	10	-	-	-
Max Instructions Executed	-	1000	1000	2000
Crossover Probability	50%	0%	0%	0%
Mutation Probability	50%	0%	0%	0%
ULTRA Probability	0%	100%	100%	100%
ULTRA Mutation Rate	-	0.01	0.05	0.01
ULTRA Alternation Rate	-	0.01	0.05	0.01
ULTRA Alignment Deviation	-	10	10	10

A1 Results

Parent Selection Method	Tournament Size	Success Rate	Difference in Success Rate with Lexicase	95% Confidence Interval of Difference in Success Rate
Lexicase	-	0.99	-	-
Tournament	2	0.01	0.98	[0.923, 0.999]
Tournament	3	0.01	0.98	[0.923, 0.999]
Tournament	4	0.05	0.94	[0.869, 0.975]
Tournament	5	0.02	0.97	[0.909, 0.993]
Tournament	6	0.04	0.95	[0.882, 0.981]
Tournament	7	0.03	0.96	[0.895, 0.987]
Tournament	8	0.06	0.93	[0.856, 0.968]
Tournament	9	0.07	0.92	[0.843, 0.961]
Tournament	10	0.04	0.95	[0.882, 0.981]
IFS	2	0.13	0.86	[0.771, 0.915]
IFS	3	0.43	0.56	[0.449, 0.649]
IFS	4	0.58	0.41	[0.302, 0.501]
IFS	5	0.55	0.44	[0.331, 0.532]
IFS	6	0.64	0.35	[0.246, 0.440]
IFS	7	0.57	0.42	[0.312, 0.512]
IFS	8	0.64	0.35	[0.246, 0.440]
IFS	9	0.71	0.28	[0.182, 0.367]
IFS	10	0.73	0.26	[0.164, 0.346]

A2 Results

Parent Selection Method	Tourna-ment Size	Success Rate	Difference in Success Rate with Lexicase	95% Confidence Interval of Difference in Success Rate
Lexicase	-	1.0	-	-
Tournament	2	0	1.0	[0.953, 1.0]
Tournament	3	0.06	0.94	[0.869, 0.974]
Tournament	4	0.12	0.88	[0.795, 0.930]
Tournament	5	0.14	0.86	[0.772, 0.914]
Tournament	6	0.16	0.84	[0.749, 0.898]
Tournament	7	0.17	0.83	[0.737, 0.890]
Tournament	8	0.10	0.90	[0.819, 0.946]
Tournament	9	0.26	0.74	[0.638, 0.813]
Tournament	10	0.18	0.82	[0.726, 0.882]
IFS	2	0.28	0.72	[0.616, 0.795]
IFS	3	0.61	0.39	[0.286, 0.479]
IFS	4	0.74	0.26	[0.167, 0.343]
IFS	5	0.83	0.17	[0.090, 0.243]
IFS	6	0.84	0.16	[0.082, 0.232]
IFS	7	0.83	0.17	[0.090, 0.243]
IFS	8	0.88	0.12	[0.050, 0.185]
IFS	9	0.79	0.21	[0.124, 0.288]
IFS	10	0.72	0.28	[0.185, 0.364]

Digital Multiplier Results

Parent Selection Method	**Tournament Size**	**Success Rate**	**Difference in Success Rate with Lexicase**	**95% Confidence Interval of Difference in Success Rate**
Lexicase	-	1.0	-	-
Tournament	2	0	1.0	[0.953, 1.0]
Tournament	4	0	1.0	[0.953, 1.0]
Tournament	6	0	1.0	[0.953, 1.0]
Tournament	7	0	1.0	[0.953, 1.0]
Tournament	8	0	1.0	[0.953, 1.0]

Factorial Results

Parent Selection Method	**Tournament Size**	**Success Rate**	**Difference in Success Rate with Lexicase**	**95% Confidence Interval of Difference in Success Rate**
Lexicase	-	0.51	-	-
Tournament	2	0	0.51	[0.401, 0.599]
Tournament	4	0	0.51	[0.401, 0.599]
Tournament	6	0	0.51	[0.401, 0.599]
Tournament	8	0	0.51	[0.401, 0.599]
Normalized	2	0	0.51	[0.401, 0.599]
Normalized	4	0	0.51	[0.401, 0.599]
Normalized	6	0	0.51	[0.401, 0.599]
Normalized	8	0.01	0.50	[0.390, 0.591]
IFS	2	0	0.51	[0.401, 0.599]
IFS	4	0	0.51	[0.401, 0.599]
IFS	6	0	0.51	[0.401, 0.599]
IFS	8	0	0.51	[0.401, 0.599]

wc Results

Parent Selection Method	Tournament Size	Success Rate	Difference in Success Rate with Lexicase	95% Confidence Interval of Difference in Success Rate
Lexicase	-	0.055	-	-
Tournament	3	0	0.055	[0.020, 0.088]
Tournament	5	0	0.055	[0.020, 0.088]
Tournament	7	0	0.055	[0.020, 0.088]
IFS	3	0	0.055	[0.020, 0.088]
IFS	5	0	0.055	[0.020, 0.088]
IFS	7	0	0.055	[0.020, 0.088]

Diversity

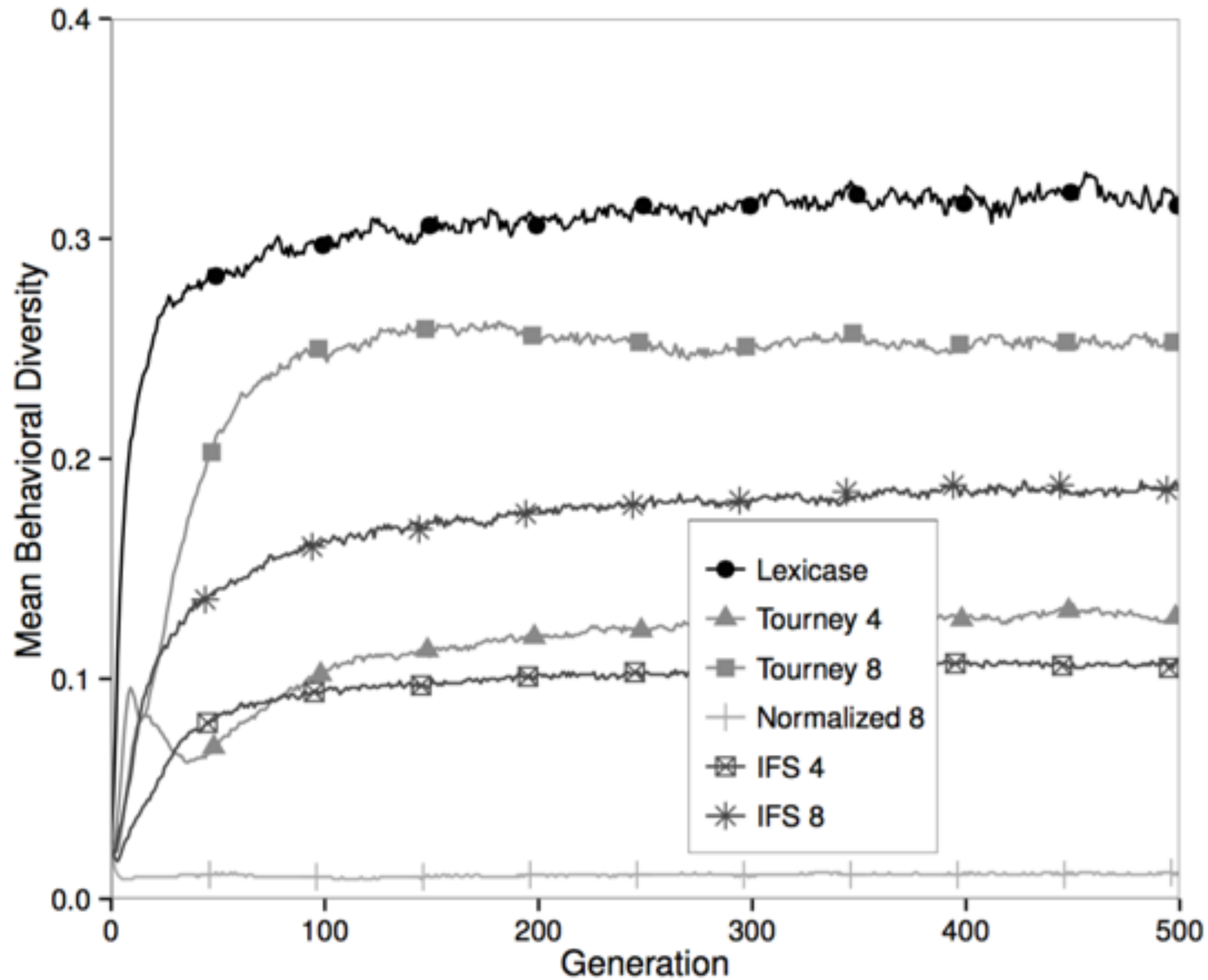


Fig. 4. Behavioral diversity for the factorial problem. The numbers beside runs indicate the tournament size used.

Cost

Problem	Parent Selection Method	Minimum mean time per generation (seconds)	Maximum mean time per generation (seconds)
$\mathbf{A}_1$	Lexicase	2.6	2.6
	Tournament	1.2	1.4
	IFS	1.2	1.3
$\mathbf{A}_2$	Lexicase	2.5	2.5
	Tournament	1.2	1.4
	IFS	1.0	1.2
DM	Lexicase	464	464
	Tournament	25	71
Fact	Lexicase	11.9	11.9
	Tournament	5.4	6.7
	Normalized	0.4	0.6
	IFS	3.0	4.7
wc	Lexicase	394	394
	Tournament	142	295
	IFS	136	229

Future

- Try lexicase selection on your problems and in your systems!
- Investigate how/when/why lexicase selection helps
- Improve performance where it helps less, e.g. for problems with continuous errors
- Decrease cost
- Look for Tom Helmuth's dissertation, to appear soon

Thanks

- Members of the Hampshire College Computational Intelligence Lab.
- This material is based upon work supported by the National Science Foundation under Grants No. 1017817, 1129139, and 1331283. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the National Science Foundation.