Word Count

as a Traditional Programming Benchmark Problem for Genetic Programming

Tom Helmuth and Lee Spector

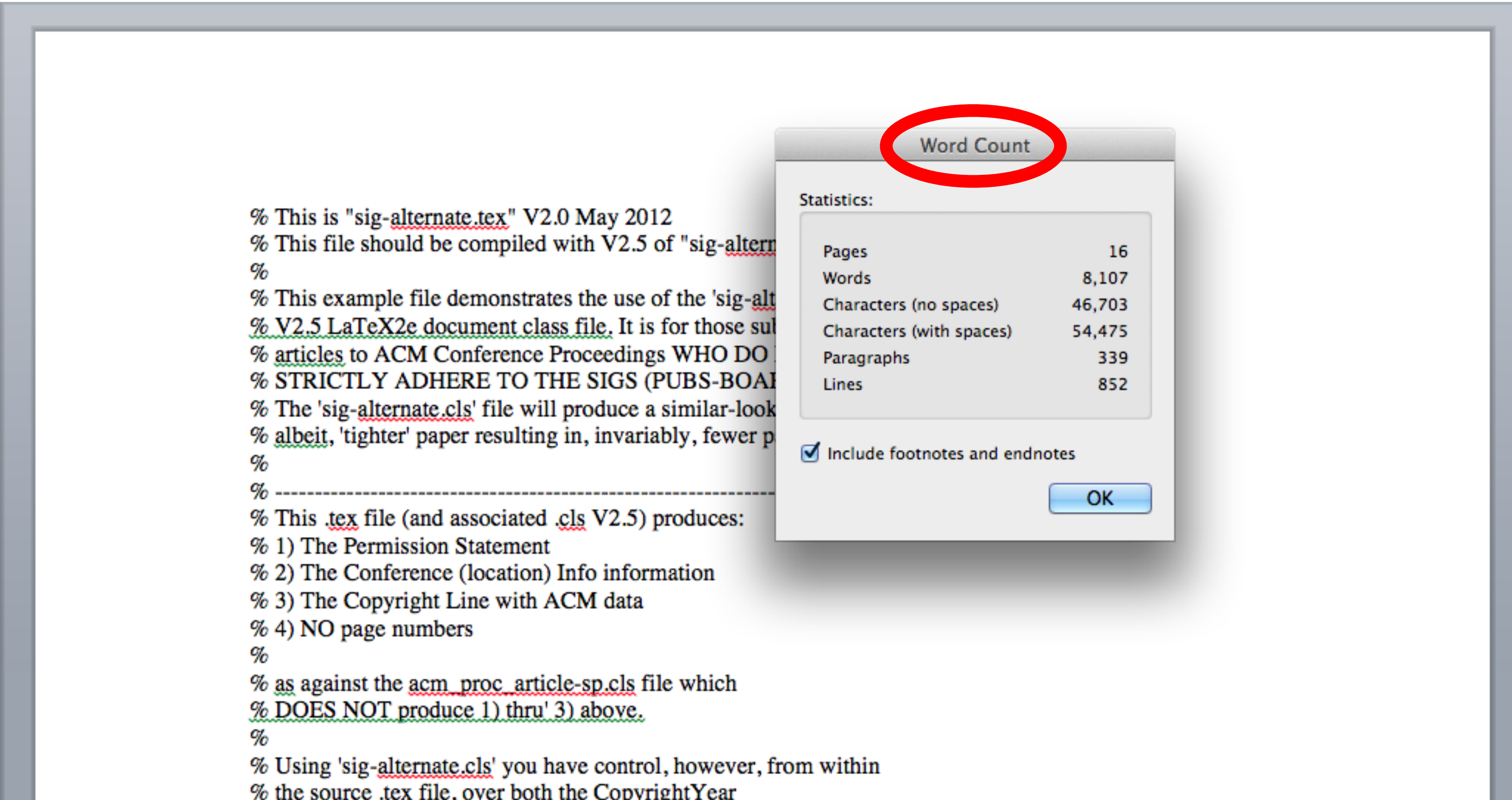
Traditional Programming Problems in GP

- Mimic human programming
- Large instruction set
  - multiple data types
  - control flow
  - I/O
- Based on tests
  - input/output example behavior

Traditional Programming Problems in GP

- Need benchmark problems!
  - interest shown in community survey[1]
  - but, none recommended in survey paper
- Word count problem

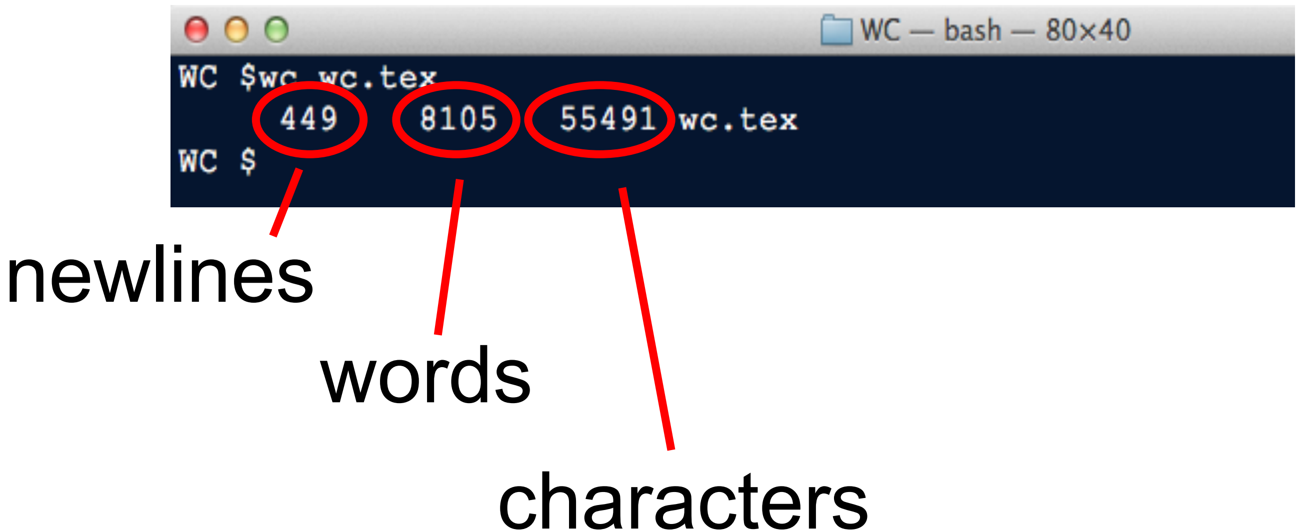
[1] D. R. White, J. Mcdermott, M. Castelli, L. Manzoni, B. W. Goldman, G. Kronberger, W. Jaskowski, U.-M. O'Reilly, and S. Luke. Better GP benchmarks: community survey results and proposals. Genetic Programming and Evolvable Machines, 14(1):3-29, Mar. 2013.


Word Count
Statistics:
Pages 16
Words 8,107
Characters (no spaces) 46,703
Characters (with spaces) 54,475
Paragraphs 339
Lines 852
Include footnotes and endnotes
OK
% This is "sig-alternate.tex" V2.0 May 2012
% This file should be compiled with V2.5 of "sig-altern
%
% This example file demonstrates the use of the 'sig-alt
% V2.5 LaTeX2e document class file. It is for those sul
% articles to ACM Conference Proceedings WHO DO
% STRICTLY ADHERE TO THE SIGS (PUBS-BOAI
% The 'sig-alternate.cls' file will produce a similar-look
% albeit, 'tighter' paper resulting in, invariably, fewer p
%
% This .tex file (and associated .cls V2.5) produces:
% 1) The Permission Statement
% 2) The Conference (location) Info information
% 3) The Copyright Line with ACM data
% 4) NO page numbers
%
% as against the acm_proc_article-sp.cls file which
% DOES NOT produce 1) thru' 3) above.
%
% Using 'sig-alternate.cls' you have control, however, from within
% the source .tex file, over both the CopyrightYear

Unix Command wc

```
WC $wc wc.tex
     449    8105   55491 wc.tex
WC $
```

Unix Command wc



Why wc Makes An Interesting Traditional Programming Problem

- Requires multiple data types
- Imitates real program
- Difficult but reasonably fast
- Open source, easy to implement
- Generalization to unseen test cases

Generate wc Problem Instance: Test Cases

- 0 to 100 character files
- Random string
  - 200 training set -- 500 test set
- Random string ending in newline
  - 20 training set -- 50 test set
- Edge cases
  - 22 training set
  - examples: “”, “A”, “\n”, “\n” repeated for 100 chars

Example Experiment

- Compare parent selection techniques
  - lexicase selection
  - tournament selection
  - implicit fitness sharing selection

Lexicase Parent Selection

- Emphasizes individual test cases
  - not aggregated fitness across test cases
- Uses random ordering of test cases for each selection event
- Unlike in Pareto selection, some test cases provide more selection pressure than others

Lexicase – Pseudocode

To select single parent:

1. Shuffle test cases
2. First test case – keep best individuals
3. Repeat with next test case, etc.
   a. Until one individual remains

Push and PushGP

- **Push** - Stack-based language for GP
- Arguments and results from typed stacks
- Executing code also on stack

- **PushGP** - Mostly typical GP using Push

http://pushlanguage.org

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", "␣" } $\{x \mid x$ is a non-whitespace character$\}$

PushGP Parameters

Runs Per Condition	200
Fitness Evaluations Budget	72,600,000
Population Size	1000
Max Generations	300
Max Program Size	1000
Max Initial Program Size	400
Max Node Evaluations	2000
Genetic Operator	ULTRA (100%)
ULTRA Mutation Rate	0.01
ULTRA Alternation Rate	0.01
ULTRA Alignment Deviation	10

Performance Metrics for Traditional Programming Problems

- When comparing sets of runs, don’t use mean best fitness
  - don’t care about incremental improvements of GP
- Care about perfect solutions
  - must pass training and unseen test sets
- Compare success rates

Success Rates

- Fisher’s exact test for significance
- Confidence intervals on difference

Results

Selection	Tournament Size	Successes (200 runs)
Lexicase	-	11
Tournament	3	0
	5	0
	7	0
Implicit Fitness Sharing	3	0
	5	0
	7	0

Results

- 95% confidence interval: **[0.020, 0.088]**
- Small but meaningful differences

Conclusions

- More traditional programming in GP!
  - problems/benchmarks
  - wc problem good starting point
  - applications
- Lexicase selection

Acknowledgments: This material is based upon work supported by the National Science Foundation under Grants No. 1017817 and 1129139. 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.