

Evolution of Machine Language Iteration using Linearly structured GP

Motivation

- Provide direct synthesis of recursion from non-recursive instruction set;
 - Explicit support in the instruction set for: “do-until,” “while,” “for,” etc., is not provided;
 - Any recursion needs to appear as a natural consequence of the search for solutions.

Register Machine

- Consider a basic register machine in which the ‘machine’ is completely defined by three sets of attributes,
 - External State – ‘ m ’ registers;
 - Internal State – special purpose registers {PC, flag};
 - Sequence of ‘ n ’ instructions;
- Such a register machine will be simulated (but may also be directly implemented in terms of the host machine instruction set [1]).

External ‘register’ state

- $\mathbf{R} \equiv \langle R_0, \dots, R_{m-1} \rangle$
- Input, output and any intermediate results are expressed through this construct.

Internal State

- PC – program counter – address for next instruction;
- Flag – result of the last comparison operation,
 - {<, >, ==, \perp }
 - where \perp denotes the undefined initial state.

Instruction Set

- Any program consists of a sequence of ‘ n ’ legal instructions,
 - $\mathbf{I} \equiv \langle I_0, \dots, I_{n-1} \rangle$
 - Thus,

- $PC \in \{0, \dots, n - 1\}$
- if $PC > n - 1$ then program terminates.
- Contents of the instruction set is naturally problem specific
 - Figure 2 [2] denotes the basic instruction set, hereafter the virtual register machine- M or VRM- M ;
- From the instruction set of Figure 2 [2] it follows that the number of syntactically distinct instructions for register machine VRM- $M(m, n)$ is given by,
 - $S(n, m) = 3m^2 + 4m + 4(n + 1) + 1$
- The total number of syntactic programs, as opposed to just one instance, is therefore,
 - $S(n, m)^n$

Selection Operator

- As per Tree Structured GP, generational, tournament or steady state tournament selection are all applicable.

Search Operators

- Consider,
 - *2-point crossover* – given two parent instruction sequences, I_i, I_j , exchange code sub-sequences of ‘ l ’ instructions (all programs remain of fixed length, n).
 - a) Locate instruction, index p_i , from instruction sequence I_i .
 - b) Choose subsequence length, l , such that $p_i < p_i + 1 < n$.
 - c) Locate instruction, index p_j , in instruction sequence I_j .
 - d) Exchange l consecutive instructions between p_i and p_j in I_i and I_j respectively.

Note – all selections are stochastic.

Mutation

- Randomly select an instruction and REPLACE with a (uniformly) randomly generated instruction;
 - Randomly create *type*;
 - Randomly fill in operands as required by the instruction type.

Evaluation

- VRM- $M(m, n)$ takes an,
 - Initial instruction sequence, I ;
 - Initial register assignment, R ;
 - *A priori* computational limit, maximum of k instructions may be executed.
- And maps this to the,
 - Final register state, R' ;
 - Final PC value;
- Or,
 - $\varepsilon : (I, R, k) \rightarrow (R', PC)$
- Let,
 - $PC = n$ if $PC < k$;

Task 1 – Evolving generic solution to multiplication [2]

- Arithmetic operators limited to addition alone.
- Let,
 - (i, j) denote a training case t ,
 - where $t = i \times j$
- Compose the training set from the finite set of natural numbers, N_{10} ;
 - Or $T \equiv \{(i, j) \mid \forall i, \forall j \in N_{10}\}$
- Raw fitness of training case t takes the form,
 - $F_{(t)}(I) \equiv |R'_0 - ij| + |PC - n|$
 - Where ' PC ' is the above adjusted PC value.
- Total Fitness
 - $F(I) \equiv \sum_{t \in T} F_{(t)}(I)$

Experiment

- $R \equiv \langle 0, i, j, 0, \dots, 0_{m-1} \rangle$

Registers (m)	4	Instructions (n)	8
P(xover)	25%	Max Inst. Exe (k)	100
Population size	1024	Num. Trials	2,000

Experiment I		Experiment II	
P(mutate)	0%	P(mutate)	0.1%
28 solutions	2.2×10^6 evaluations / solution	60 solutions	6.8×10^6 evaluations / solution

- $S(n, m)^n \approx 3.5 \times 10^{16}$
- Random walk of 10^9 VRM- $M(4, 8)$ programs produces,
 - 104 solutions, or 9.5×10^6 evaluations per solution.

Discussion

- GP solves problem by processing less programs than random walk.
- Not clear whether GP anymore efficient in total computational overhead.

Task 2 – Evolving generic solution to integer sequences [3]

- Wider set of (integer) problems,
 - Square $\equiv s = x^2$
 - Cube $\equiv s = x^3$
 - Factorial $\equiv s = \text{if } (x = 0) \text{ then } 1 \text{ else factorial}(x - 1)$
 - Fibonacci $\equiv s = \text{if } (x = 0 \text{ or } 1) \text{ then } 1 \text{ else Fibonacci}(x - 2) + \text{Fibonacci}(x - 1)$
- Wider set of alternative search algorithms
 - Exhaustive Iterative Hill Climbing (EIHC)
 - Methodology
 - Conduct an exhaustive search of all 1-point program changes and greedily accept the best as the next move.
 - Method
 - a) Random generation of ‘ n ’ instructions for program ‘ p ’;
 - b) Execute Program and compute fitness;
 - c) For all $i \in p$
 - i. for all Instructions \in Instruction Set
 - [1] Replace p_i with Instruction $\neq p_i$;

[2] Execute Program and compute fitness;

d) $p^* = \max \{ f(p) \}$ over steps (b) to (c)

e) IF $p^* \neq p$ THEN $p = p^*$ and return to (c);

f) ELSE IF $f(p) \neq f^*(p)$ THEN return to (a).

o Hybrid Search

▪ As per GP except one individual at each generation created by,

a) Select individual I from population;

b) Apply EIHC to create I' ;

c) Update the population $I \leftarrow I'$;

o Random Search

▪ As before,

• Randomly create ‘ n ’ instructions;

• Evaluate and record fitness;

Instruction Set

▪ Augment previous instruction set, defining VRM- $S(m, n)$, Figure 2 [3],

o Full set of arithmetic operators.

o Negation operator;

o ‘Stdout’

▪ explicitly forward register values for output;

▪ Append identified register content to output stream ‘stdout’;

▪ Number of syntactically distinct instructions for register machine VRM- $S(m, n)$ is given by,

$$o \quad S(n, m) = 6m^2 + 6m + 4(n + 1) + 1$$

▪ The total number of syntactic programs, as opposed to just one instance, is therefore,

$$o \quad S(n, m)^n \approx 10^{35} \text{ for } n, m \text{ of } 12.$$

Evaluation

o $\varepsilon : (I, R, k) \rightarrow (Stdout)$

▪ Fitness

o Let s denote the sequence function, defined over a sequence of P terms;

o Raw fitness,

- $F_s(\mathbf{I}) \equiv \sum_{i \in \{0, \dots, P-1\}} |v(i) - s(i)| \cdot \text{scale}(i)$
- where $v(i)$ is the i th term from stdout; $s(i)$ is the sequence value at index i ; and $\text{scale}(i)$ is a scaling function for each element of the sequence,
 - $\text{scale}(i) \rightarrow s_{\max} \text{ IF } s(i) = 0; \text{ ELSE } s_{\max} / s(i)$

Experiment

Registers (m)	12	Instructions (n)	12
Population size	256	P(xover)	25%
Stop Criteria:		10 solutions or 5×10^7 evaluations	

- Table 1 [3] summarize performance.
 - Hybrid algorithm produces best performance;
 - GP alone fails on Fibonacci sequence entirely;
 - Hill climber always finds some solutions;
 - ‘Random’ FAILS to find ANY solutions in all but simplest problem.
- All Solutions are exact.
- Comments on solutions found, Figure 3 [3],
 - Factorial and Fibonacci
 - Follow the recursive definition, retaining sub-results in registers R4 and R5.
 - Square and Cube
 - Expand binomial theorem and recursively compute answer;
 - Multiplication rarely employed.

Discussion

- Recursive solutions always employed;
- Directed search methods only located solutions;
- Instructions set does not directly support recursive operations.

Task 3 – Generic Solution to Binary Problem [4]

- General solution to the parity problem sought, independent of word length.
- Also investigate the significance of,
 - Crossover – substitute mutation of consecutive instructions;
 - Lexicographic fitness function – prioritised ranking of multiple objectives.

- Previous work on the parity problem,
 - Solution fixed to size of the terminal set (say ‘b’ bits)
 - Problem solution only good for parity $\leq b$.
 - ADF with tree based GP
 - Solution still fixed size, but k relatively large (11)
 - A lot of a priori information necessary for ADF definition.

Genetic Search

- Selection Operator
 - q – tournament selection.
 - Each parent chosen through a tournament between q individuals.
 - Lexicographic fitness
 - Multiple tiers, one per criterion.
 - IF (multiple individuals given same fitness under tier 1 (most significant criterion))
 - THEN (apply tier 2 criterion to resolve tie)
 - Tier #1 – correctness of program I on test case t .
 - $F_1^t(I) \equiv |R_2 - \text{answer}(t)|$
 - $F_1(I) \equiv \sum_{t \in T} F_1^t(I)$; i.e. lower values are best.
 - Tier #2 – program termination on test case t .
 - $F_2^t(I) \equiv \begin{cases} 1 & \text{if } I \text{ stops within } k \text{ instructions.} \\ 0 & \text{otherwise} \end{cases}$
 - $F_2(I) \equiv \sum_{t \in T} F_2^t(I)$; i.e. higher values are best.
- Search Operator
 - 2-point crossover
 - Single point mutation
 - Headless-Chicken Crossover (Macro mutation)
 - Most crossover results in children performing worse than or equivalent to the parents, so why bother?
 - Macro Mutation operator defined,
 - Generate 2 sequences of l random instructions;

- Over write instructions in parents I_i and I_j at $p_i, p_j < n - 1$;
- I.e. nothing is transferred from both parents to child.

Problem Definition

- There are an infinite set of parity problems which defined in terms of a natural number, b , in which the set of training cases, B , consists of 2^b exemplars.
- The ‘answer’ for training case t is the number of true assignments in t .
- Individuals receive,
 - t – a test case as a bit string of b bits.
 - b – number of bits (5 in this case).
- For each of the 2^b training cases input registers are initialised,
 - $R \equiv \langle t, b, 0, \dots, 0_{m-1} \rangle$

Instruction Set

- Branch instructions
 - Include conditional check;
 - Now perform a test against zero;
 - Bitwise instructions supported;
 - SHL, SHR – produce NOP if result negative or overflow.

Evaluation

- Parameters

Population Size	8192	Max. Instr. Exe. (k)	100
Registers (m)	10	Instructions (n)	20
GA-XO P(xover)	25%	GA-HC P(xover)	25%
P(SP mutation)	0.1%		

- Stop Criteria
 - Max Evaluation of 1×10^{10} or 10 solutions;
 - Solution implies terminating program and correct answer on all 2^b training patterns.
- Results – Table 1 [4]
 - Random search produced 1 solution;
 - Micro-mutation significantly better than alternative schemes.
- Solution Codes – Figure 2 [4].

- Type A
 - Counts number of bits set in input bit string;
 - Time Complexity \propto MSB set;
 - Specific example shown is specific to 'b' odd;
- Type B
 - Subtraction and AND successively removes LSB set.
 - Time Complexity \propto number of set bits;
 - Represent 34 of 40 solutions
- Type C
 - Appears to be a rare solution.
 - Not sure of the practicality as based on division.

Discussion

- Novel solution found, but not computationally efficient.
 - How might computational efficiency be encouraged?
- Search clearly needs to be directed as opposed to random in order to identify solutions.
- Does the lack of crossover rule out building blocks?

Reference

- [1] Nordin P., A Compiling Genetic Programming System that Directly Manipulates the Machine Code. Kinnear K. (ed), Advances in Genetic Programming, Chap. 14, MIT Press, pp 311-331, 1994.
- [2] Huelsbergen L., Toward Simulated Evolution of Machine-Language Iteration. Proceedings of the 1st Conference on Genetic Programming. Pp 315-320, 1996.
- [3] Huelsbergen L., Learning Recursive Sequences Via Evolution of Machine Language Programs. Proceedings of the 2nd Conference on Genetic Programming. Pp 186-194, 1997.
- [4] Huelsbergen L., Finding General Solutions to the Parity Problem by Evolving Machine-Language Representations. Proceedings of the 3rd Conference on Genetic Programming. Pp 158-166, 1998.

Note, [2] – [4] are available from,

<http://cm.bell-labs.com/cm/cs/who/lorenz/papers/bib.html>