The Automatic Generation of MutationOperators.pptx for Genetic Algorithms

[Workshop on Evolutionary Computation for the Automated Design of Algorithms 2012]

John Woodward – Nottingham (CHINA)

Jerry Swan - Stirling

In a Nutshell…

- We are *(semi)-automatically designing new mutation* operators to use within a Genetic Algorithm.
- The mutation operators are <u>*trained*</u> on a set of problem instances drawn from a particular probability distribution of problem instances.
- The mutation operators are <u>*tested*</u> on a new set of problem instances drawn from the *same* probability distribution of problem instances.
- We are not designing mutation operators by hand (as many have done in the past). "*We propose a new operator ….*"
- We are using machine learning to generate an optimization algorithm (we need independent training **(seen)** and test **(unseen)** sets from the same distribution)

Outline

- **Motivation** – why automatically design
- Problem Instances and **Problem Classes (NFL)**
- **Meta** and Base Learning - **Signatures** of GA and Automatic Design
- Register Machines (**Linear Genetic Programming**) to model mutation operators. Instruction set and 2 registers.
- **Two Common mutation operators** (one-point and uniform mutation)
- **Results** (highly statistically significant)
- **Response to reviewers'** comments
- Conclusions – *the algorithm is automatically tuned to fit the problem class (environment) to which it is exposed*

Motivation for Automated Design

- The cost of **manual** design is **increasing** exponentially in-line with inflation (10% China).
- The cost of **automatic** design in **decreasing** in-line with Moore’s law (and parallel computation).
- Engineers **design for X** (cost, efficiency, robustness, …), Evolution **adapts for X** (e.g. hot/cold climates)
- We should **design metaheuristics for X**
- It does not make sense to talk about the performance of a metaheuristics in the absence of a **problem instance/class**. Needs context.

Problem Instances and Classes

A **problem instance** is a single example of an optimization problem (in this paper either a real-valued function defined over 32 or 64 bits).

A **problem class** is a probability distribution over problem instances.

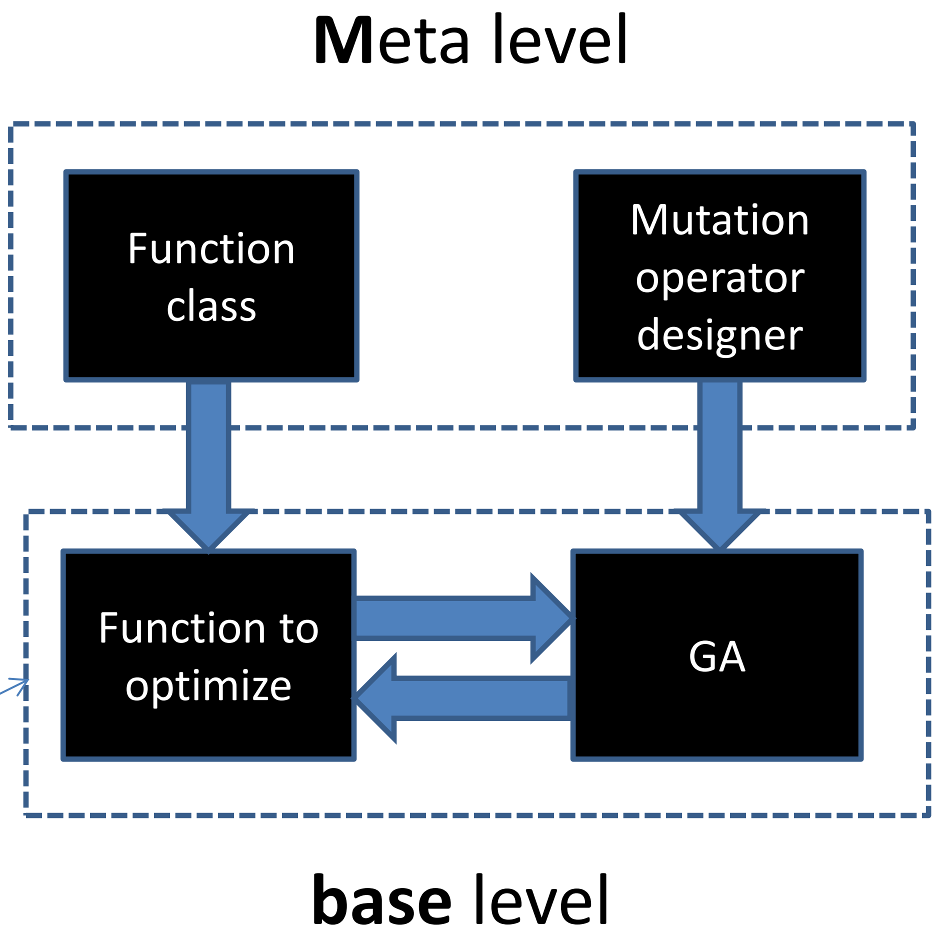
Often we do not have explicit access to the probability distribution but we can only sample it (except with synthetic problems).

Important Consequence of No Free Lunch (NFL) Theorems

- Loosely, NFL states under a uniform probability distribution over problem instances, **all metaheuristics perform equally well** (in fact identically). It formalizes a trade-off.
- This implies that under some other distributions (in fact ‘<u>almost all</u>’), **some algorithms will be superior.**
- **Automatic design can exploit** the fact an assumption of NFL is not valid (which is the case with most real world applications).

Meta and Base Learning

- At the **base** level we are learning about a **specific** function.
- At the **meta** level we are learning about the problem **class**.
- We are just doing **"generate and test"** at a higher level
- What is being passed with each **blue arrow**?
- **Conventional** GA



Compare Signatures (Input-Output)

Genetic Algorithm

- (B^n -> R) -> B^n

Input is a function mapping bit-strings of length n to a real-value.

Output is a (near optimal) bit-string

(i.e. the solution to the problem instance)

GA/mutation designer

- [(B^n -> R)] -> ((B^n -> R) -> B^n)

Input is a *list of* functions mapping bit-strings of length n to a real-value (i.e. sample problem instances from the problem class).

Output is a (near optimal) mutation operator for a GA

(i.e. the solution method to the problem class)

Register Machine with Indirection (USED AS MUTATION OPERATORS)

A program is a list of instructions and arguments.

A register is set of addressable memory (R0,..,R4).

Negative register addresses means indirection.

A program cannot affect IO registers directly

PROGRAM

Inc	0
Dec	1
Add	1,2,3
If	4,5,6
Inc	-1
Dec	-2

INPUT-OUTPUT REGISTERS

0	-1	+1	0	…	

WORKING REGISTERS

0	-1	+1	0	…	

Program counter pc	2

Arithmetic Instructions

These instructions perform arithmetic operations on the registers.

- **Add** Ri $\leftarrow$ Rj + Rk
- **Inc** Ri $\leftarrow$ Ri + 1
- **Dec** Ri $\leftarrow$ Ri − 1
- **Ivt** Ri $\leftarrow -1 * $ Ri
- **Clr** Ri $\leftarrow$ 0
- **Rnd** Ri $\leftarrow$ Random([−1, +1]) //mutation rate
- **Set** Ri $\leftarrow$ value

Control-Flow Instructions

These instructions control flow (NOT ARITHMETIC). They include branching and iterative imperatives.

Note that this set is *not Turing Complete*!

- **If** if(R0 > R1) pc = pc + R2
- **IfRand** if(arg1 < 100 * random[0,+1]) pc = pc + arg2//allows us to build mutation rates
- **Rpt** Repeat Rj times next Ri instruction
- **Stp** terminate

Human designed Register Machines

Line	UNIFORM	ONE POINT MUTATION
0	**Rpt, 33, 18**	**Rpt, 33, 18**
1	Nop	Nop
2	Nop	Nop
3	Nop	Nop
4	**Inc, 3**	**Inc, 3**
5	Nop	Nop
6	Nop	Nop
7	Nop	Nop
8	**IfRand, 3, 6**	**IfRand, 3, 6**
9	Nop	Nop
10	Nop	Nop
11	Nop	Nop
12	**Ivt,−3**	**Ivt,−3**
13	**Nop**	**Stp**
14	Nop	Nop
15	Nop	Nop
16	Nop	Nop

- **One point** mutation

Flips a single bit

- **Uniform mutation**

Flips all bits with a fixed probability.

Why insert NOP (No operation)?

Parameter settings for Register Machine

Parameter	Value
restart hill-climbing	100
hill-climbing iterations	5
mutation rate	3
program length	17
Input-output register size	33 or 65
working register size	5
seeded	uniform-mutation-RM
fitness	best in run, averaged over 20

Note that these parameters are not optimized.

Parameter settings for the GA

Parameter	**Value**
Population size	100
Iterations	1000
bit-string length	32 or 64
generational model	steady-state
selection method	fitness proportional
fitness	see next slide
mutation	register machine

<u>Note that these parameters are not optimized – except for the mutation operator.</u>

7 Problem Classes

1. We generate a Normally-distributed value t = −0.7 + 0.5 N (0, 1)) in the range [-1, +1].
2. We linearly interpolate the value t from the range [-1, +1] into an integer in the range [0, 2^num−bits −1], and convert this into a bit-string t'.
3. To calculate the fitness of an arbitrary bit-string x, the hamming distance between x and the target bit-string t' is calculated (giving a value in the range [0,numbits]). This value is then fed into one of the 7 functions.

7 Problem Classes

number	function
• 1	x
• 2	sin2(x/4 − 16)
• 3	(x − 4) ∗ (x − 12)
• 4	(x ∗ x − 10 ∗ cos(x))
• 5	sin(pi∗x/64−4) ∗ cos(pi∗x/64−12)
• 6	sin(pi∗cos(pi∗x/64 − 12)/4)
• 7	1/(1 + x /64)

Results – 32 bit problems

Problem classes Means and standard deviations	Uniform Mutation	One-point mutation	RM-mutation
p1 mean	30.82	30.96	31.11
p1 std-dev	0.17	0.14	0.16
p2 mean	951	959.7	984.9
p2 std-dev	9.3	10.7	10.8
p3 mean	506.7	512.2	528.9
p3 std-dev	7.5	6.2	6.4
p4 mean	945.8	954.9	978
p4 std-dev	8.1	8.1	7.2
p5 mean	0.262	0.26	0.298
p5 std-dev	0.009	0.013	0.012
p6 mean	0.432	0.434	0.462
p6 std-dev	0.006	0.006	0.004
p7 mean	0.889	0.89	0.901
p7 std-dev	0.002	0.003	0.002

Results – 64 bit problems

Problem classes Means and stand dev	Uniform Mutation	One-point mutation	RM-mutation
p1 mean	55.31	56.08	56.47
p1 std-dev	0.33	0.29	0.33
p2 mean	3064	3141	3168
p2 std-dev	33	35	33
p3 mean	2229	2294	2314
p3 std-dev	31	28	27
p4 mean	3065	3130	3193
p4 std-dev	36	24	28
p5 mean	0.839	0.846	0.861
p5 std-dev	0.012	0.01	0.012
p6 mean	0.643	0.643	0.663
p6 std-dev	0.004	0.004	0.003
p7 mean	0.752	0.7529	0.7684
p7 std-dev	0.0028	0.004	0.0031

p-values for 32 and 64-bit functions on the7 problem classes

	32 bit	32 bit	64 bit	64 bit
class	Uniform	One-point	Uniform	One-point
p1	1.98E-08	0.0005683	1.64E-19	1.02E-05
p2	1.21E-18	1.08E-12	1.63E-17	0.00353
p3	1.57E-17	1.65E-14	3.49E-16	0.00722
p4	4.74E-23	1.22E-16	2.35E-21	9.01E-13
p5	9.62E-17	1.67E-15	4.80E-09	4.23E-06
p6	2.54E-27	4.14E-24	3.31E-24	3.64E-28
p7	1.34E-24	3.00E-18	1.45E-28	5.14E-23

Example Operators

p1 32 bit	p1 64 bit	p2 32 bit	p2 64 bit	p3 32 bit	p3 64 bit	p4 32 bit	p4 64 bit	p5 32 bit	p5 64 bit	p6 32 bit	p6 64 bit	p7 32 bit	p7 64 bit
0 Rpt 33 18	0 Ivt -54	0 Set -10 16	0 Rpt 65 18	0 Rnd -8	0 Set 6 27	0 Inc -27	0 Rpt 65 18	0 Rpt 33 18	0 Rpt 65 18	0 Rpt 33 18	0 Rpt 65 18	0 Rpt 33 18	0 Rpt 65 18
1 Nop 0	1 Dec 38 14	1 Ivt 9	1 Inc 23	1 Clr 26	1 Nop 0	1 Rpt -2 -7	1 Rnd 36	1 Nop 0	1 Clr 7	1 Rnd -17	1 Rnd 1	1 Dec 26 27	1 Nop 0
2 IfRand 7 4	2 Nop 0	2 Nop 0	2 Nop 0	2 Nop 0	2 Nop 0	2 Rpt 26 -17	2 Nop 0	2 Nop 0	2 Nop 0	2 Nop 0	2 Nop 0	2 Rnd -31	2 Nop 0
3 Nop 0	3 Nop 0	3 Nop 0	3 Nop 0	3 Nop 0	3 If 40 39 -26	3 Nop 0	3 Nop 0	3 Set 32 4	3 Add 46 -38 0	3 Dec 5 29	3 Nop 0	3 Rpt -14 -23	3 Nop 0
4 Inc 3	4 Inc 3	4 Inc 3	4 Inc 3	4 Inc 3	4 Inc 3	4 Inc 3	4 Inc 3	4 Inc 3	4 Inc 3	4 Inc 3	4 Inc 3	4 Inc 3	4 Inc 3
5 Nop 0	5 Nop 0	5 If 8 -10 12	5 Nop 0	5 If -15 4 -11	5 Nop 0	5 Nop 0	5 Nop 0	5 Nop 0	5 Nop 0	5 IfRand 31 10	5 Add -37 28 0	5 Nop 0	5 Nop 0
6 Nop 0	6 Nop 0	6 Nop 0	6 Nop 0	6 IfRand -3 31	6 Nop 0	6 Add -3 -19 0	6 Nop 0	6 Dec -10 25	6 Rpt 23 48	6 Nop 0	6 Nop 0	6 Clr 6	6 Nop 0
7 Nop 0	7 Nop 0	7 Nop 0	7 Nop 0	7 Nop 0	7 Nop 0	7 Rnd -3	7 Rpt 41 -43	7 Ivt 25	7 Add 53 -42 0	7 Inc 5	7 Nop 0	7 Dec 32 0	7 Nop 0
8 IfRand 3 6	8 IfRand 1 6	8 If 24 -16 -27	8 If 3 -32 22	8 IfRand 8 -7	8 Add -35 -35 0	8 Rpt -30 -13	8 Rpt 11 57	8 Rnd -18	8 IfRand 1 6	8 If 18 26 27	8 Ivt -9	8 Inc 23	8 Add 9 48 0
9 Nop 0	9 Nop 0	9 Clr -8	9 Nop 0	9 IfRand -20 23	9 Nop 0	9 Nop 0	9 Nop 0	9 Nop 0	9 Clr -5	9 Nop 0	9 Nop 0	9 Rnd -28	9 If 62 26 31
10 Nop 0	10 Nop 0	10 If -17 2 -16	10 Ivt -13	10 Rnd -32	10 Dec 30 36	10 Rpt -21 -13	10 Clr -46	10 IfRand -27 -14	10 Add 47 9 0	10 Nop 0	10 Set 19 35	10 Rpt 0 18	10 Nop 0
11 Nop 0	11 Nop 0	11 Nop 0	11 Nop 0	11 Nop 0	11 Nop 0	11 Rpt 7 -23	11 Ivt 24	11 Rnd 1	11 Inc -42	11 Nop 0	11 Nop 0	11 Nop 0	11 Inc 56
12 Ivt -3	12 Ivt -3	12 Rnd -23	12 Ivt -3	12 Inc -8	12 Ivt -3	12 Ivt -3	12 Ivt -3	12 Ivt -3	12 Ivt -3	12 Ivt -3	12 Ivt -3	12 Inc -29	12 Ivt -3
13 Nop 0	13 Nop 0	13 Nop 0	13 Nop 0	13 Rnd 26	13 Nop 0	13 Dec 11 -32	13 Add 50 30 0	13 Inc 25	13 Set 17 45	13 Rpt 29 2	13 Nop 0	13 Nop 0	13 Nop 0
14 Dec -19 -1	14 Inc -48	14 Nop 0	14 Nop 0	14 Nop 0	14 Nop 0	14 Nop 0	14 Dec -38 56	14 Nop 0	14 Nop 0	14 If -14 -32 -25	14 Nop 0	14 Nop 0	14 Nop 0
15 Rpt -12 22	15 Nop 0	15 Nop 0	15 Nop 0	15 If 13 2 -25	15 Nop 0	15 Nop 0	15 Set -8 26	15 Nop 0	15 Nop 0	15 Nop 0	15 Nop 0	15 Nop 0	15 Nop 0
16 Nop 0	16 Nop 0	16 Nop 0	16 Nop 0	16 Nop 0	16 Nop 0	16 Nop 0	16 Nop 0	16 Nop 0	16 Nop 0	16 Nop 0	16 Nop 0	16 Nop 0	16 Nop 0

Reviews comments

1. Did we test the new mutation operators against standard operators (one-point and uniform mutation) on different problem classes?

- NO – the mutation operator is designed (evolved) specifically for that class of problem.

2. Are we taking the training stage into account?

- NO, we are just comparing mutation operators in the testing phase – Anyway how could we meaningfully compare “brain power” (manual design) against “processor power” (evolution).

Summary and Conclusions

1. Automatic design is ‘better’ than manual design.

2. Signatures of Automatic Design are **more general** than GA.

3. think about frameworks (**families of algorithms**) rather than algorithms, and **problem classes** rather than problem instances.

4. We are not claiming Register Machines are the best way.

5. Shown how two common mutation operators (one-point and uniform mutation) can be expressed in this RM framework.

6. Results are **statistically significant**

7. the algorithm is automatically **tuned to fit the problem class** (environment) to which it is exposed

8. We do not know how these mutation operators work. Difficult to interpret.

References

- C. Giraud-Carrier and F. Provost. Toward a Justification of Meta-learning: Is the No Free Lunch Theorem a Show-stopper? In Proceedings of the ICML-2005 Workshop on Meta-learning, pages 12–19, 2005.
- Jonathan E. Rowe and Michael D. Vose. Unbiased black box search algorithms. In Proceedings of the 13th annual conference on Genetic and evolutionary computation, GECCO ’11, pages 2035–2042, NewYork, NY, USA, 2011. ACM.
- J.R. Woodward and J. Swan. Automatically designing selection heuristics. In Proceedings of the 13th annual conference companion on Genetic and evolutionary computation, pages 583–590. ACM, 2011.
- Edmund K. Burke, Mathew R. Hyde, Graham Kendall, Gabriela Ochoa, Ender Ozcan, and John R. Woodward. Exploring hyper-heuristic methodologies with genetic programming.

…and Finally

- Thank you
- Any questions or comments
- I hope to see you next year at this workshop.