N-gram GP: Early results and half-baked ideas

Dagstuhl Seminar 08051
Theory of Evolutionary Algorithms
31 Jan 2008

Nic McPhee (with Riccardo Poli)
University of Minnesota, Morris, and University of Essex

Introduction/motivation

Which of these sentences was written by me, which by Darwin, and which by Shakespeare?

```
    “I would have sent to Rome that's worthy death?”

    “The naturalist looking at species as he might succeed from a
fork low down in the separation of the species of any species in
a more or less from their sap this is unimportant for the
instincts already possessed by certain plants so that natural
selection of mere piles of superimposed strata and watch the sea
separating an island even if we believe that pure water can
effect little or no offspring.”

    “The troubling aspects of a building block semantics in a
given tree in the context and false.”
```

None was actually written by a human. All were generated by computing the frequency of 3-grams (triplets of adjacent words) taken from three texts (*Coriolanus*, *Origin of species*, and “Semantic building blocks in genetic programming” to appear in EuroGP 2008), and then using those frequencies to generate sequences of words. The generated sentences capture the style of the sampled texts (it's easy to see which sentence came from which source) and make local sense (at the level of phrases). They become gibberish at the macro level, however; the long-range correlations fall apart.

In this talk I present N-gram GP, a system for evolving linear GP programs using an EDA style system to update the probabilities of different 3-grams (triplets) of instructions. I then pick apart some of the evolved programs in an effort to better understand the properties of this approach and identify ways that it might be extended.

For technical details see “A linear estimation-of-distribution GP system”.

Outline of N-gram GP

We use a simple 2 register linear GP system with seven instructions:

```
    ID  Instruction
    0   R1 = x
    1   R2 = x
```

```
2   R1 = R1 + R2
3   R2 = R1 + R2
4   R1 = R1 * R2
5   R2 = R1 * R2
6   Swap R1 R2
```

R1 is initialized to x, R2 is initialized to 0. The output is read from R1.

In these experiments we only used triples, so we're really doing 3-gram GP. We have two probability matrices: L and M. L is the probability distribution for lengths. M(A, B, C) is (roughly) the probability of generating C as the next instruction assuming that the previous two instructions are AB.

To generate individuals we first choose a length from L, and then use M to choose that number of instructions. We use the appropriate projections of M to generate distributions for the first two instructions. Point mutation is applied to the resulting program.

Our basic execution loop:

1. Initialize L and M uniformly
2. Generate a population of individuals
3. Perform truncation selection
4. Update the probabilities in L, M based on the winners
5. Cook until done

Results on symbolic regression

We have a target function and a set of test points (21 evenly spaced points x in [-1, 1]). The goal is to evolve a program that matches the target function on those test points.

The system works quite well on certain types of target functions; details of results are in our paper.

Despite having only triplets to work with, the system can learn long sequences of instructions. E.g., can evolve distributions that generate a sequence of 9 instructions with a probability of more than 60%, which is more than 500,000 times more likely than choosing it at random from a uniform distribution of sequences of 7 instructions.

With the target x+x^2+...+x^7 we saw lots of 53 pairs (where 5 and 3 are instruction ideas from the table above) such as 1535353535352. The pair 53 maps the state of the registers from (a, b) -> (a, a(b+1). So this sequence of instructions passes through the states (x, 0) ->(1) (x, x) ->(53) (x, x + x^2) ->(53) (x, x + x^2 + x^3) -> ... -> (53) (x, x+x^2+...x^6) ->(52) (x+x^2+...+x^7, x^2+...+x^7).

This works, but is brittle because it depends on the final 52 pair at the end to ensure that R1 has the result. We need 53's everywhere except at the one, key point at the end.

This is reflected in the probability matrix M. Pr{3 | 35} = 66%, where Pr{2 | 35}=1.3% and Pr{5

| 53} > 99.999%. The system has “learned” that the sequence 53 should *always* be followed by a 5, where the sequence 35 should *usually* be followed by a 3, but occasionally by a 2 instead.

Possible “gene duplication” extension

This is entirely speculative; we haven't tried it but it seems plausible.

It's common to see rows in M that have a small number of non-zero values; the system is then having to probabilistically choose from a small set of options there (like what to follow 35 with above). This suggests a conflict that is having to be settled probabilistically, which is problematic in cases that that above, where you need the dice to roll “just right” at a specific, key point.

A possible approach to this is occasionally identifying such cases, and then creating a new pseudo-instruction that is a duplicate of a key instruction (either A or B; I lean towards B, but with no particular support for that). You then divide up all the probabilities associated with B between B and B', renormalize, and continue.

One question (of many) is whether we should deliberately bias the system when we do this. If we just copy and halve the probabilities, then the system will (at first) behave exactly the same as before. There’s a symmetry to break, however. Should we help out by breaking it right away? We could, for example, set ABC to 0.75, and ABD to 0.25, while AB’C is 0.25 and AB’D is 0.75. This breaks that symmetry and “forces” the system to deal with it, but I have no sense at the moment whether that's a good thing.

Dagstuhl photo URLs

I've been asked to my photos from the seminar so I've posted these links to the seminar wiki:

A few of my “cleaned up” photos (old and new): `http://tinyurl.com/36hrz4`
Unedited dumps (this year’s event): `http://tinyurl.com/32lve4`

Acknowledgements

First, I'd like the thank the participants at the 2006 Theory of EA's Dagstuhl seminar for their extremely helpful comments. The work presented their has eventually turned into “Semantic building blocks in genetic programming”, which will appear in EuroGP 2008.

I'd also like to thank this year's Dagstuhl participants for their wonderful ideas, feedback, and suggestions.

Lastly, I'd like to thank Riccardo and the folks at the University of Essex for providing such a productive research environment for my 2007-2008 sabbatical year.