Evolving Regular Expressions for GeneChip Probe Performance Prediction

W. B. Langdon and A. P. Harrison

Departments of Mathematical, Biological Sciences and
Computing and Electronic Systems, University of Essex, UK

Technical Report CES-483
ISSN: 1744-8050
27 April 2008

Abstract

Commercial GeneChips provide highly redundant but noisy data. Rapid identification and subsequent rejection of bad data effectively increases the quality of the remaining data at little cost whilst serving as a basis for better understanding the bio-physics of short surface mounted DNA sequences.

Affymetrix High Density Oligonuclotide Arrays (HDONA) simultaneously measure expression of thousands of genes using millions of probes. Regular expressions can be evolved from a Backus-Naur form (BNF) context-free grammar using tree based strongly typed genetic programming written in `gawk`. Fitness is given by `egrep`. The quality of individual HG-U133A probes is indicated by its correlation across 6685 human tissue samples from NCBI's GEO database with other measurements for the same gene. Low concordance indicates a poor probe. The evolved data mined motif is better at predicting poor DNA sequences than an existing human generated RE, suggesting runs of Cytosine and Guanine and mixtures should all be avoided. Section 4.6 gives more RE GP gawk implementation details.

1 Introduction

Typically Affymetrix GeneChips (e.g. HG-U133A) measure gene expression at at least eleven points along the gene. Individual measurements are given by short (25 base) DNA sequences, known as probes. These are complementary to corresponding locations in genes. Being complementary, the gene product (messenger RNA) preferentially binds to the probe. Cf. Figure 2. Half a million probes are placed on a glass slide in a square grid pattern. A fluorescent dye is used to measure how much mRNA is bound to each probe.

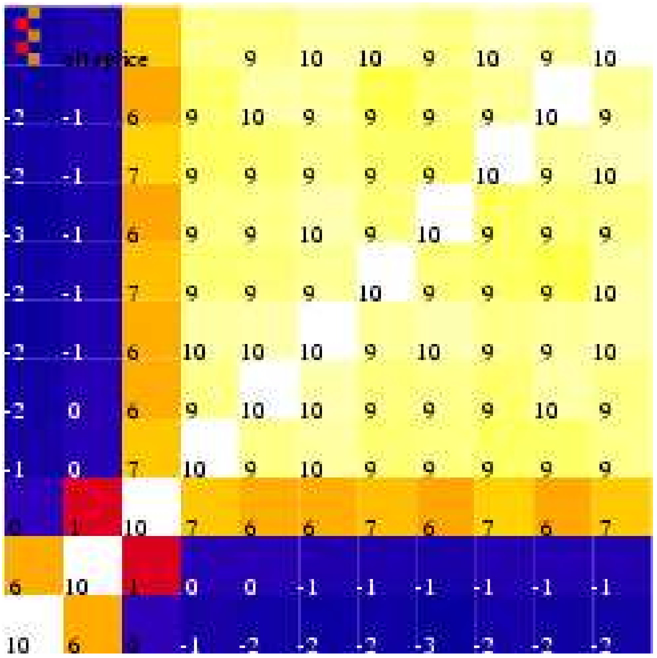

Figure 1: Correlation coefficients between 11 probes for gene "S100 calcium binding protein A11" S100A11. Nine of the perfect match are correlated but probes PM_1 and PM_2 (note negative values, in blue) are not.

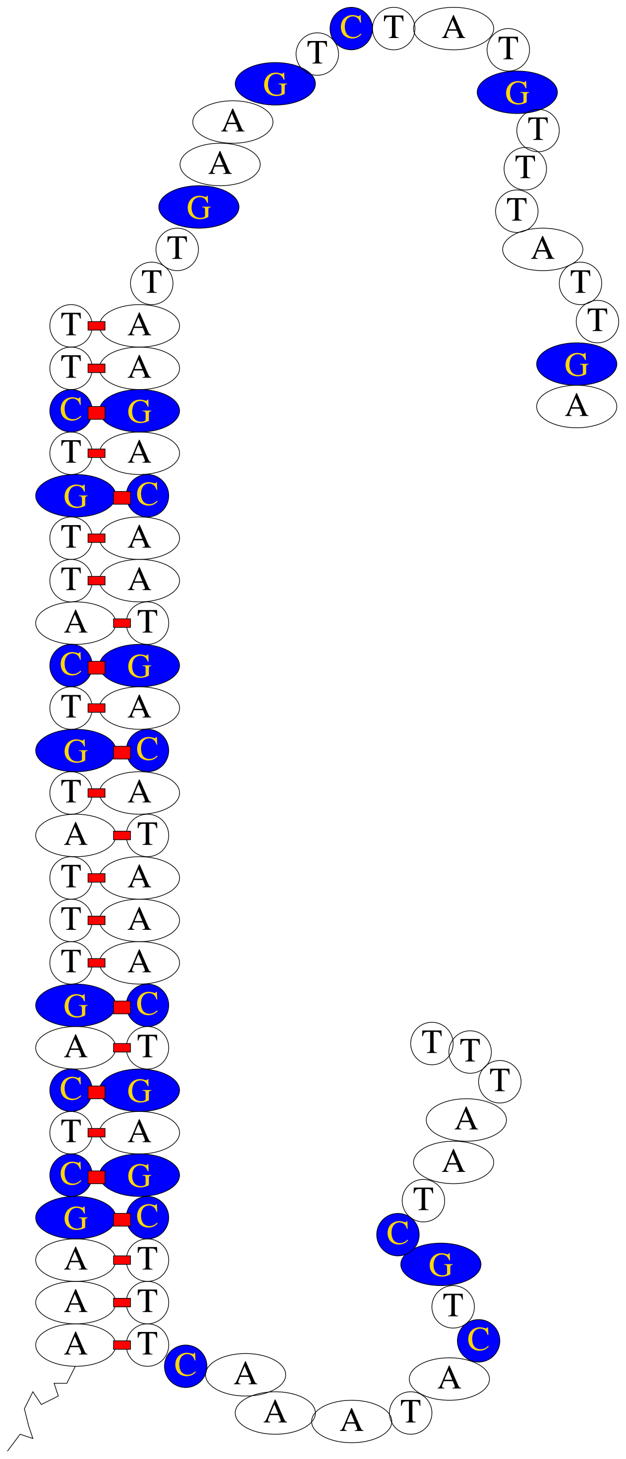
C T A T G T T T A T T G A
T G A A G T T
T T C T G T T A C T G T A T T T G A C T C G A A A
A A G A C A A T G A C A T A A A C T G A G C T T T
C A A A T A C T G C T A A T T T

Figure 2: Schematic of an Affymetrix probe (209649_at PM_5, left) bound with complementary target sequence (right). DNA double helix represented as straight vertical ladder. Note complementary T–A and C–G base bindings are shown by red rectangles. The 25 bases of the probe are tethered to the glass slide by a flexible linker (black lower left). Firmly bound target sequences can be detected by treatment with a florescent dye, whose location is detected with a laser and an optical microscope. The florescent intensity is approximately proportional to the amount of bound target and so gives some indication of target gene activity.

While nothing is simple in Biology, to a first approximation the amount of mRNA produced by a gene should be the same no matter which part of the mRNA molecule is bound to a probe. Affymetrix groups probes into probesets. Each probeset targets a gene. Excluding controls, the HG-U133A has 22 215 probesets. For simplicity we concentrate upon the 21 765 HG-U133A probesets with exactly 11 probes. Figure 1 shows for an example probeset its 110 correlations as a "heatmap" (yellow/lighter corresponds to greater consistency between pairs of probes).

There are several biological reasons which might lead to probes on the same gene giving consistently unrelated readings. (Alternative splicing, alternative polyadenylation and 3'-5' degradation, come to mind [Langdon *et al.*, 2007a].) However these do not explain all of the many cases of poor correlation. In [Langdon, 2008] we found some technological reasons. In particular [Langdon, 2008] showed that probes containing a large ratio of Guanine (G) to Adenosine (A) bases are likely to perform badly. Subsequently we have found that runs of Gs (which will tend to have a high G/A ratio) also tend to indicate problem probes [Upton *et al.*,]. This has lead us to ask if there are other *sequences* which might indicate dodgy probes.

The next section will describe the preparation of datasets containing the correlation coefficients and probe sequences. Section 4 describes the evolutionary algorithm use to create search strings for `egrep`. Section 5 describes how well the GP does. Section 6 uses the evolved motif to suggest potential Physical explanations for poor probes. Finally, in Section 7 we conclude that several regular expressions, e.g. `G(G|C){4}`, in additions to `GGGG`, can be used together to locate poor probes.

2 Earlier Evolutionary Algorithm work using Grammars

Existing research on using grammars to constrain the evolution of programs can be broadly divided in two: Grammatical evolution [O'Neill and Ryan, 2001] based largely in Ireland and work in the far east by Wigham, Wong and McKay. (See, for example, [Whigham, 1996; Whigham and Crapper, 1999], [Wong and Leung, 1996] and [McKay *et al.*, 2006].)

In grammatical evolution the genotype can be thought of as a variable length linear vector of bytes rather than a tree. Programs are generated by recursively following a BNF grammar. The choice of which option in each rule is to be used is determined by the next byte in the vector. The 8-bit value is converted to a substitution option simply by reducing the integer by modulus the number of options in the rule. If recursively expanding the BNF requires processing more rules than there are bytes in the genotype, grammatical evolution simply starts again at the start of the byte vector. If it requires less, the additional bytes are simply ignored.

The grammar systems employed by Wigham Wong and Mckay are more sophisticated and the tree shaped genotype is more integrated with the grammar.

There is quite a body of work on using evolution to induce formal grammars. E.g. [Nikolaev and Slavov, 1998] tackelled the Tomita regular expression benchmarks, whilst [Langdon, 1998] evolved a context free grammar. [Cetinkaya, 2007] used grammatical evolution to create regular expression for processing HTML.

Ross induced stochastic regular expressions from a number of grammars to classify proteins from their amino acid sequence [Ross, 2001]. Typically his grammars had eight alternatives. In Stockholm regular expressions have been evolved to search for similarities between proteins, again based on their amino acid sequences [Handstad *et al.*, 2007]. Whilst Brameier in Denmark used amino acids sequences to predict the location of proteins by applying a multi-classifier [Langdon and Buxton, 2001] linear GP based approach [Brameier *et al.*, 2007] (although this can be done without a grammar [Langdon and Banzhaf, 2005]). A similar technique has also been applied to study microRNAs [Brameier and Wiuf, 2007].

The next section describes our single stage strongly typed tree based GP being applied to an important DNA problem: explaining why some DNA sequences in wide spread commercial use make poor GeneChip probes.

3 Preparation of Training Data

3.1 Calculation of Correlation Coefficients for GEO GeneChips

For completeness this section repeats the description of the calculation of the correlation coefficients given in [Langdon, 2008]. As part of major Bioinformatics datamining exercise, all the human Affymetrix GeneChips in the USA National Center for Biotechnology Information's GEO [Barrett *et al.*, 2007] had been down loaded [Langdon *et al.*, 2007b]. This included CEL files from 6 685 HG-U133A GeneChips, each containing half a million data points. The measured mRNA was collected from a wide range of Human tissues and disease states.

Since it is impossible to accurately control the total amount of mRNA dropped on each GeneChip, it is necessary to normalise the data. We used quantile normalisation, which non-linearly rescales the data so that the mean and standard deviation (and all higher moments) are the same. We normalised each chip against a reference average chip [Langdon *et al.*, 2007b].

The correlation coefficient of each pair of probes within each probeset in the HG-U133A design was calculated. (A total of 5 310 652 correlation coefficients.) To avoid undue influence of outliers: if either probe was more than three standard deviations from its average value that tissue sample was exclude from the calculation of that probe pairs correlation. Also, each GeneChip was checked for spatial defects [Langdon *et al.*, 2007b] and the sample exclude if either probe was within 4 probes of a known spatial error. Even after excluding these data, all correlations used probe pairs from several thousand samples.

The placement of probes on GeneChips, like the HG-U133A, is designed to put similar probes next to each other. This is done to ease the photolithographic chemical process used in manufacture and leads to horizontal bands in average intensity. Probe variability tends to increase with average intensity (until the signal is so strong that the probe saturates, which reduces its ability to vary). Since signal variation (as opposed to noise) is required for correlation, correlation also tends to increase with average intensity leading to horizontal bands in correlation across the GeneChip. Figure 3 shows the expected bands in average correlation for each probe.

The distribution of probe-probe correlation coefficients for probes in the same probeset has two peaks: one near 0.8 and the other near 0, cf. Figure 4. As the peak at zero emphasises, there are many probes which are simply not well correlated with the other members of their probeset. Explaining why is a major research effort. The peak at 0.8 corresponds better behaved probes.

Part of the sample preparation process requires an enzyme to act upon purified mRNA starting at the 3' end. I.e. at the normal back end of mRNA. The enzyme works its way forward towards the 5' front of the mRNA. Unfortunately it has a tendency to fall off. Therefore the strength of signals tends to fall further away from the 3' end. This is a known problem and laboratories regularly check that the effect is not too extreme in each sample. Nonetheless, there is a small trend for correlation between probes in the same probeset to fall as the distance between where they measure along the mRNA gene transcript increases. (The 5' end is the negative direction in Figure 4).

3.2 Training Data Sets for Evolving DNA Motifs

Apart from the initial calculation of the correlation coefficients (where we use the same data as we used in [Langdon, 2008], cf. previous section) new training data sets were constructed. To exclude genes which are either never expressed (or are constantly expressed) we selected 4 118 probesets where ten or more non-overlapping probe pairs had correlations of 0.8 or more. For each probe we use the median value of all 10 of its correlations with other members of its probeset (excluding those it overlaps).

We had previously noted that probes which target overlapping parts of the gene (i.e. where the regions are within 25 bases of each other) tend to have high correlation. This happens even if the probes are not well correlated with the other measurements for the same gene. This is why we excluded correlation between overlapping probe pairs when deciding whether to include a probeset or not.

These were evenly split into three to provide independent training, test and validation data.

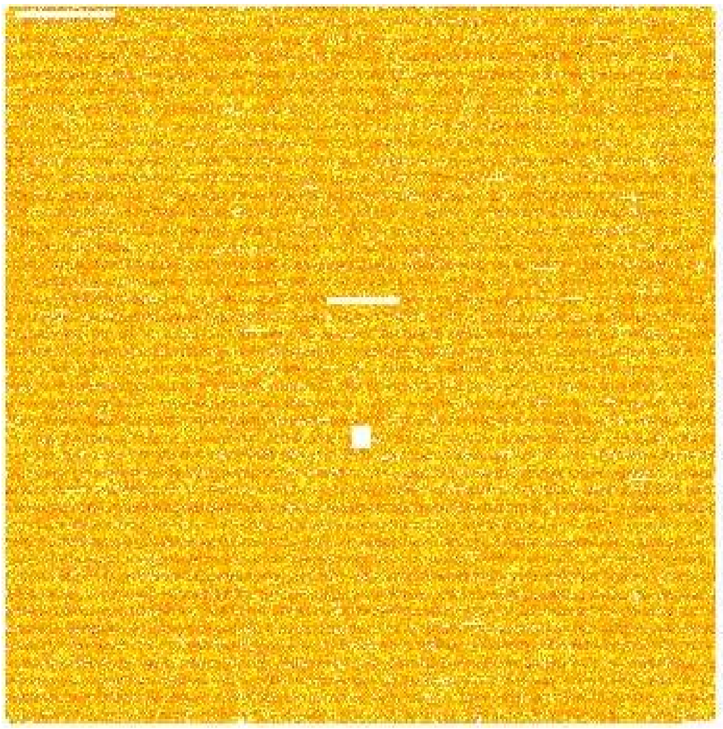

Figure 3: Mean of 5 310 652 correlations between probes in probesets across 6685 HG-U133A GeneChips. White regions contain no probesets. Smallest -0.63 (red). Median 0.16. Max 0.97 (yellow). 0.7% 3341 probes have a mean correlation greater than 0.8 with the other probes in their probeset.

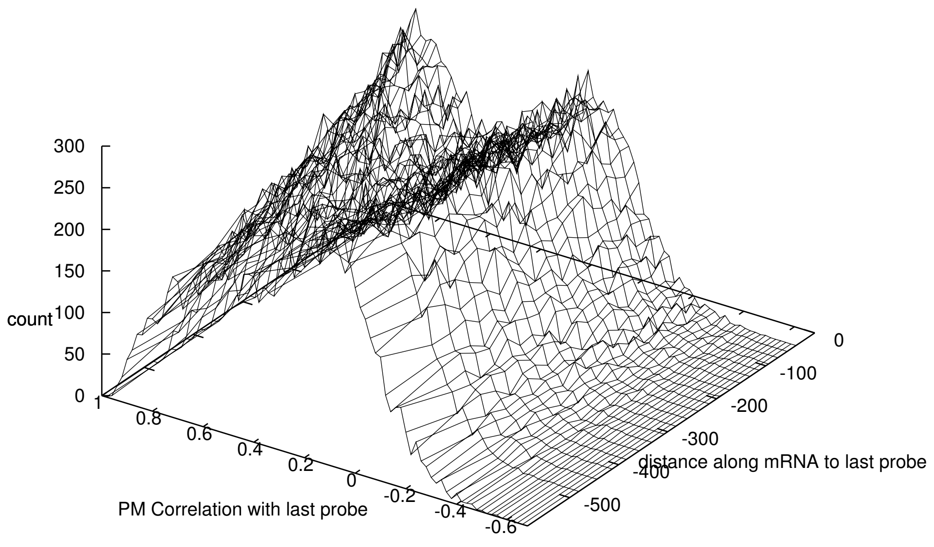

Figure 4: Correlation between PM probe and last PM probe in each GEO HG-U133A probesets. (Approx 10 000 data points per distance bin. Controls excluded.)

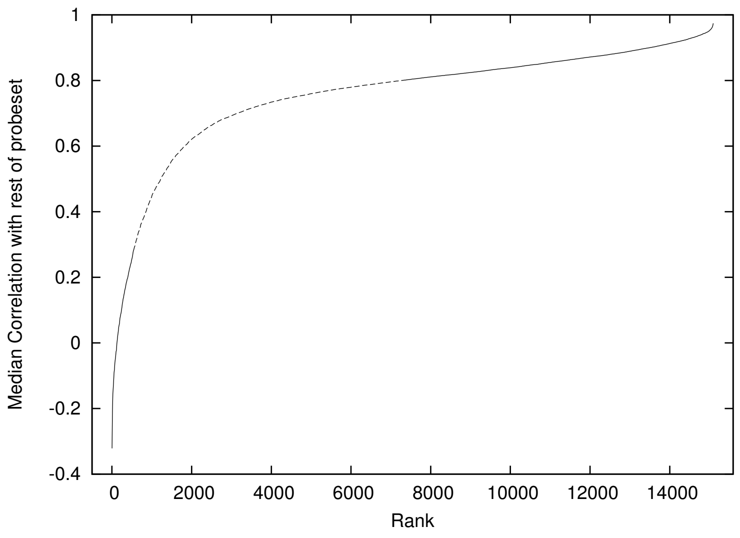

Figure 5: Training data. Probes with intermediate values (0.3 . . . 0.8) are not used.

Measurement of ultra low (pico molar) concentrations of long chain molecules, like mRNA, is noisy. Affymetrix provides various control signals, including multiple measurements to reduce noise. One controversial mechanism is adjacent to each measuring probe is a control "mismatch" probe. In previous work [Langdon, 2008] we found the "mismatch" probes were often poorly correlated with other measurements for the same gene. Since this is known, we ignored MM probes in this study.

3.3 GP Training Set

As Figure 5 shows correlation coefficients cover a wide range with many taking intermediate values. Since we are using correlation only as an indication of how well a probe is working we decided to exclude the middle values from training and instead use probe pairs that were highly correlated (≥ 0.8) or were very poorly correlated (≤ 0.3). Of the 15 092 available training examples, there are 7 832 probes highly correlated with the rest of their probeset but only 583 poorly correlated. To avoid unbalanced training sets, every generation all 583 negative examples are used and 583 positive examples are randomly chosen from the 7 832 positive examples available.

4 Evolving Regular Expression Motifs

The genetic programming system is a strongly typed tree GP system with subtree crossover [Poli *et al.*, 2008; Langdon, 1998]

4.1 BNF grammar of Regular Expression

The BNF grammar used (cf. Figure 6) is an extension of that given by Cameron `http://www.cs.sfu.ca/people/Faculty/cameron/Teaching/384/99-3/regexp-plg.html`. In particular matching the beginning of strings (`^`) and the `{n,m}` form of Kleen closure are also supported. The BNF has been customised for DNA strings. (I.e. `<char>` need only be `A C G` and `T`). It has also been simplified so that BNF rules either contain exactly two alternatives (which are themselves single symbols) or none. I.e. they are a list of further substitutions. Since various combinations of the start of string symbol, null strings and Kleen closure cause `egrep` to loop, care has been taken to ensure the new BNF does not permit null strings after `^`.

```
<start>  ::=  <RE>
<RE>  ::=  <union> | <simple-RE>
<union>  ::= <RE> "|" <simple-RE>
<simple-RE>  ::=  <concatenation> | <basic-RE>
<concatenation>  ::= <simple-RE> <basic-RE>
<basic-RE>  ::= <RE-kleen> | <elementary-RE>
<RE-kleen>::= <minmaxquantifier> | <kleen>
<kleen>::= <star> | <plus>
<star>  ::= <elementary-RE2> "*"
<plus>  ::= <elementary-RE2> "+"
<minmaxquantifier>   ::= <elementary-RE4> "{" <int> <optREint> "}"
<elementary-RE>  ::= <group> | <elementary-RE1>
<elementary-RE1>  ::= <xos> | <elementary-RE2>
<elementary-RE2>  ::= <any> | <elementary-RE3>
<elementary-RE3>::= <set> | <char>
<elementary-RE4>  ::= <group> | <elementary-RE2>
<group>  ::=  "(" <RE> ")"
<xos>  ::=  <sos> | "$"
<sos>  ::=  "^" <elementary-RE4>
<set>  ::=  <positive-set> | <negative-set>
<positive-set>  ::=  "[" <set-items> "]"
<negative-set>  ::=  "[^" <set-items> "]"
<set-items>  ::=  <set-item> | <set-items2>
<set-items2>  ::=  <set-item> <set-items>
<set-item>  ::=  <char>
<char>  ::=  <c00> | <c01>
<any>  ::=  "."
<c00>  ::=  T | C
<c01>  ::=  A | G

<optREint>  ::=  <2ndint> | $
<2ndint>  ::=  "," <int>
<int>  ::=  <d0>
#4 Bit Gray Code Encoder
<REdigit>  ::=  <d111> | <d0>
<d0>  ::=  <d00> | <d01>
<d00>  ::=  <d000> | <d001>
<d01>  ::=  <d010> | <d011>
<d000>  ::=  1
<d001>  ::=  3 | 2
<d010>  ::=  7 | 6
<d011>  ::=  4 | 5
<d111>  ::=  8 | 9
```

Figure 6: Grammar (based on Backus-Naur form) used to specify legal regular expressions for use as `egrep` search strings for testing DNA sequences.

4.1.1 Kleen closures

Brameier and Wiuf suggests that the traditional `*` and `+` form of Kleen closure are not suitable for bioinformatic applications [Brameier and Wiuf, 2007]. Instead they recommend the `{n,m}` form which explicitly defines both lower (`n`) and upper (`m`) limits on the number of times the preceeding symbol must occur. Accordingly we extended the BNF used to define regular expressions to include `{n,m}`. However both `{n,m}` and traditional Kleen closures are used by evolved solutions. To avoid `mutation.awk` seeing "Hamming cliffs", the integer quantifiers used in the `{n,m}` are Gray coded [Bäck, 1996]. Similarly the syntax groups together the chemically more similar Pyrimidines (`T` and `C`) and Purines (`A` and `G`).

Our system supports full positive integer values for `minmaxquantifier`, however even modest values can lead `egrep` to hang the computer. Therefore `n` and `m` are limited to 1-9. Finally `egrep` rejects `{n,m}` if `m<n`. This is handled by a semantic rule which removes `,m` from the phenotype when `m` is less than `n`.

4.2 Simplifying the BNF for use by Evolutionary Algorithms

For simplicity the BNF is written so that grammar rules are either simple substitution rules (e.g. `<minmaxquantifier>`) rules with two options (e.g. `<RE>`) or terminals (e.g. `"*"` and `T`). In BNF terms, a terminal is a symbol which cannot be substituted in the grammar. Therefore, unlike the BNF rules, it becomes part of the `egrep` regular expression. The simple substitution rules do not have any element of choice. They can be considered as "syntactic sugar". Once invoked, then the next level in the BNF must also be invoked. They, like terminals, cannot be chosen as crossover points or targets for mutation. Their principle use is to enable the rules with options to be kept simple.

The binary choice rules are the active parts of the syntax. As they are always binary, each sentence recognised by the BNF has an equivalent binary string. (A BNF sentence means an `egrep` regular expression in our case.) Each bit correspond to a BNF rule with two options needed when parsing the sentence. The bit indicates which option should be invoked. Note that the meaning of each bit depends upon where we have reached in the parsing the sentence (i.e. on the earlier bits). Given the recursive structure of this grammar, the bit string could (in principle) be of infinite length. In traditional GP terms, this list of choices is the evolving tree. The BNF grammar acts to put labels on the choices, which constrain crossover, but it is the choices (not the BNF grammar) which is the genetic material of the evolving individual. Note, unlike grammatical evolution, strongly typed crossover respects [Radcliff, 1993] the meaning of the BNF rules.

4.3 Creating Random BNF Sentences

The initial random population is created using ramped half-and-half [Koza, 1992]. It may help to think of this as applying the usual genetic programming ramped half-and-half algorithm to a binary tree (of choice nodes).

We start from `<start>` and recursively follow the BNF. However when we reach a rule with options we need to chose one. As in ramped half-and-half we keep track of how deep we are nested. If we have not reached the depth needed to terminate the recursion, we randomly chose one of the options. This is the equivalent of choosing a GP function. (As with other strongly typed GPs, if a chosen route through the syntax has no further choices to be made, we may be forced to terminate a recursive branch early.)

To terminate a recursion we chose the "simpler" option. Our BNF has been written so that the simpler option is always on the right. (This is flagged by `RE` in the rule name.) If there is no "simpler" choice, the choice is made randomly. This mechanism is also be used for mutating existing regular expressions.

Although this may seem complex, even `gawk` (an interpreted language) is fast enough to handle populations of a million individuals.

4.4 Crossing over BNF Sentences

Creating a new sentence from two high fitness sentences is essentially subtree crossover [Poli *et al.*, 2008] applied to the binary choice tree (cf. Section 4.1) with the addition of strong type constraints [Montana, 1995]. This is implemented by scanning the grammar used to create the first parent for all the rules with two options. One of these is randomly chosen. For example, suppose the first parent starts `<start> <RE> <union>`··· and suppose `<union>` is chosen as the crossover point. For a grammatically correct child to be produced all that is necessary is that the crossover point chosen in the second parent should also be `<union>`. (There are complications to do with depth and size limits, which we shall ignore for the time being.) Therefore the second parent is scanned to find all occurrences of `<union>`. One of them is randomly chosen to be the second crossover point. (If there are none, this crossover is aborted and another initial crossover point is chosen. If we keep failing, eventually another pair of parents is chosen.)

Crossover is based on normal GP subtree crossover, cf. [Poli *et al.*, 2008, Figure 2.5]. The new child is created by copying the start of the first parent, excluding the subtree at the first parent's crossover point. Then genetic material from the subtree at the second parent's crossover point is added. Finally the remainder of the first parent is appended to the child. This is implemented by crossing over the binary choice trees to create a binary choice tree for the new child. Apart from issues of tree size and depth, we are guaranteed that the new binary choice tree will represent a valid sentence in the BNF grammar.

The final step is to recursively trace through the BNF grammar, obeying the binary choices. Each time an BNF terminal (except the null symbol) is encountered it is appended to the new regular expression. In order to be able to breed following generations, the new individual's genotype must also be passed to the next generation. For convenience this is done by writing each BNF symbol as it is encountered to the file holding the next generation.

4.5 Evaluating the Fitness of BNF Sentences

Each generation a command file is generated which contains a `egrep -c -v '`*RE*`'` command for each individual in the population. (*RE* is the individual's regular expression.) The command is run on a file holding the DNA sequences of the 583 probes poorly correlated with the rest of their probeset. `egrep -c -v` counts the number of probes which do not match the evolved motif (*RE*). The same command is also run on a file holding the 583 positive probes selected for use in this generation. The fitness of the regular expression is based on the difference between the number of lines in the two files which match *RE*. Expressions which either match all probes or fail to match any are penalised by subtracting 583 from their score. (There is also a 583 penalty per file if `egrep` fails. This was invoked when developing the regular expression BNF syntax, cf. Section 4.1. However the grammar was tweaked (e.g. by limiting `<minmaxquantifier>` to numbers less than 10) so that `egrep` can now cope with all the strings it is given.)

4.6 Implementation

The system starts with an outer command shell script which is responsible for initialising the population and termination. It invokes an inner script `RE_gp1.bat` once per generation. The population is stored as a text file which contains, for each individual in the population, a line holding its genotype and an `egrep` command line containing the individual expressed as a regular expression. I.e. its phenotype.

4.6.1 `RE_gp1.bat`

Every generation `RE_gp1.bat` executes the population as a command script. It passes to it two files. One holds the negative test cases. The second holds the positive cases selected for this generation. I.e. the second changes each time `RE_gp1.bat` is invoked. `egrep`'s output is piped through `fitness.awk` and `sort` is used to select the best 200 of the current generation. `gawk` command lines and file `stats.awk` are invoked to provide immediate feedback on the run's progress.

`select.awk` extracts each of the best 200 individuals from the current population and puts them into a temporary file for use by `crossover.awk`, `mutation.awk` and `shrink.awk`. Shrink mutation is described in [Langdon, 1998]. (In this experiment only `crossover.awk` is used.)

`crossover.awk` creates the next population by crossing over pairs of individuals chosen uniformly at random from the current elite. `syntax.awk` is used to process the BNF grammar and contains several functions which are used by `crossover.awk`, `stats.awk`, `random_syntax.awk` etc.

4.6.2 `crossover.awk`

Function `crossover()` (cf. Section 4.4) is called with a randomly chosen pair of parents. If it fails (e.g.because it cannot find compatible crossover points) then another pair of parents is randomly chosen.

`crossover()` first makes a list of all the genetic rules in the first parent. I.e. those rules with two options. One of these is randomly chosen. To enforce *strong typing* `crossover()` next forms a list of all the places in the second parent where this rule is used. If there aren't any, `crossover()` goes back and randomly tries another crossover point in the first parent. Once a compatible pair of crossover points has been found crossover proceeds in the usual way for subtree crossover.

Genetic material from the start of the first parent up to its crossover point is copied by `treenode()` into globals `tree` and `IP`. In effect `tree` is a variable length bit string which defines the genetic material of the new individual. Then genetic material from the subtree starting at the second parent's crossover point is copied from the second parent by `traverse()` into globals `tree` and `IP`. Finally `treenode()` is used to copy genetic material from after the subtree to be replaced from the first parent. At this point `tree` and `IP` contain a complete specification, in terms of the branch points in the BNF grammar, for the new individual.

A modified version of the code used to recursively create random programs is now invoked. `SubInit_()` chases through the BNF grammar, following each substitution rule as it encounters it and storing them as a list in `prog` and `PC`. However, when `SubInit_()` finds a binary option rule it uses the next element in `tree` to decide which option to take.

The function `print_prog` transfers the current individual's regular expression (as stored in globals `prog` and `PC`) to the output stream (assumed to be connected to a file which will hold the next generation). The output is formated as an `egrep` command.

`gawk` is an interpreted language which does not have a notion of declaring variables (but note the use of additional function arguments to limit the scope of variables to within the function). Instead variables are created as they are used. Note the use of the function `element`, which checks that an element of an array exists before returning it. If such a check is not made, referencing a non-existent array element will cause it to be created with a default value. The default value is the empty string, which can be interpreted as false or zero, depending upon how it is used. The function `check` is used to try and verify that the syntax, read in by `syntax.awk`, has not been corrupted.

5 Results

At the end of the first run, with a population of 1000 (cf. Table 1 and Figure 7) GP evolved a probe performance predictor (see Figure 8) equivalent to `GGGG|CGCC|G(G|C){4}|CCC`. It is obvious that it includes the previous rule `GGGG` [Upton *et al.*,] but includes other possibilities. Therefore it finds more poor probes. Inevitably it will also incorrectly predict more high correlation probes are poor but the increase is more than offset by its better performance on the poor probes. See Figure 9. Thus it has a fitness of 856 v. 776 for `GGGG`.

Table 1: Strongly Typed Grammar based GP Parameters for GeneChip Correlation Prediction

Primitives:	The function and terminal sets are defined by the BNF grammar (cf. Figure 6. BNF rules with two options correspond to binary GP functions. The rest of the BNF grammar correspond to GP terminals.
Fitness:	true positives+true negatives. (I.e. proportional to the area under the ROC curve or Wilcox statistic [Langdon and Barrett, 2004].) Less large penalty if `egrep` fails or it matches all probes or none.
Selection:	(200,1000)
Initial pop:	Ramped half-and-half 3:7
Parameters:	100% subtree crossover. Max tree depth 17 (no tree size limit)
Termination:	50 generations

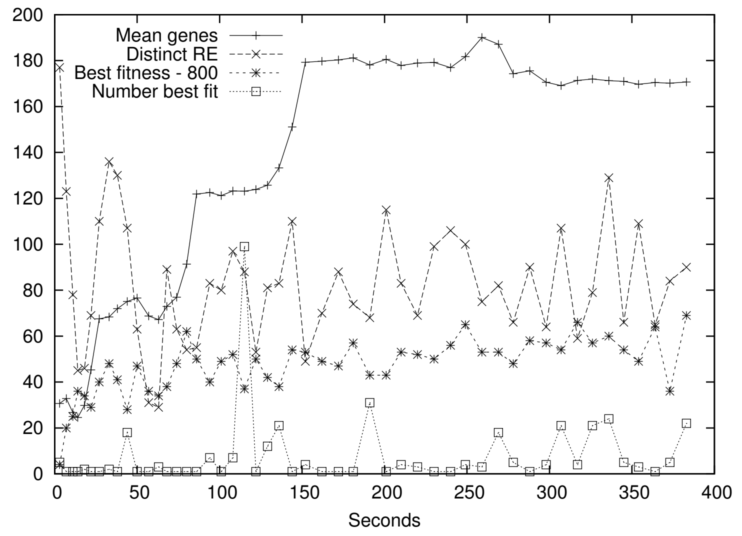

Figure 7: Evolution of breeding population (best 200 of 1000) of regular expressions to find poor GeneChip probes. Each generation the positive training cases are replaced leading to fluctuations in the measured best fitness (*). However diversity remains high and there are usually few individuals with the same highest fitness (□). In this run the number of distinct phenotypes (×) (i.e. `egrep` search strings) is almost identical to the number of distinct genotypes. Bloat is limited (+), apparently by the tree depth limit (17). However the system slows down by ($\approx \frac{1}{2}$) as evolution proceeds.

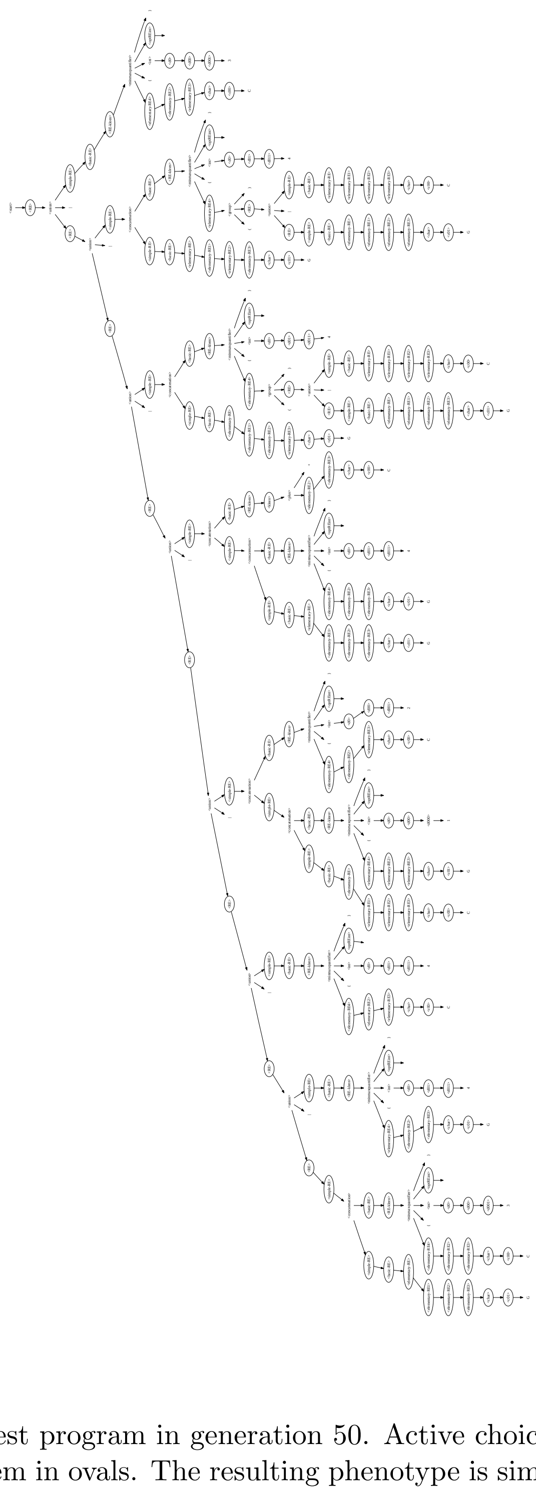

Figure 8: Geneotype of best program in generation 50. Active choice nodes in the BNF (cf. Figure 6) are emphasised by placing them in ovals. The resulting phenotype is simply the 58 (excluding empty symbols) end nodes, read in left to right order: `GC{3}|G{4}|C{4}|CG{1}C{2}|GG{4}C+|G(G|C){4}|G(G|C)4|C{3}`. It is equivalent to `GGGG|CGCC|G(G|C){4}|CCC`.

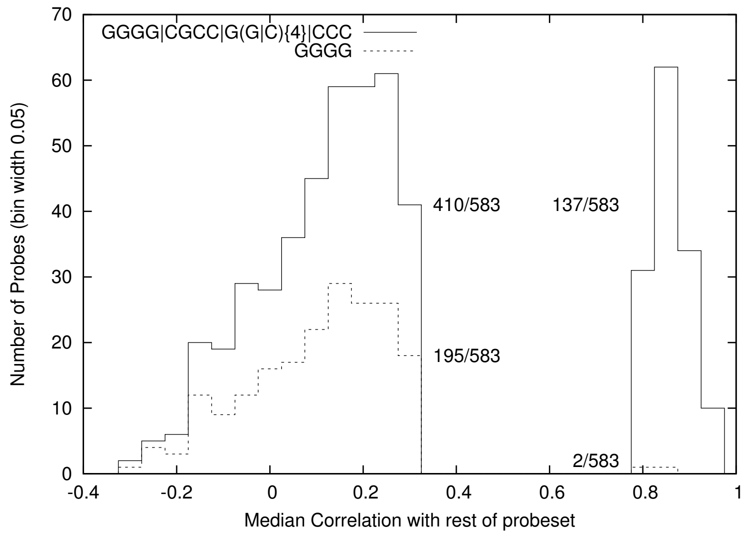

Figure 9: Performance of evolved motif on its training data versus human generated motif (dashed).

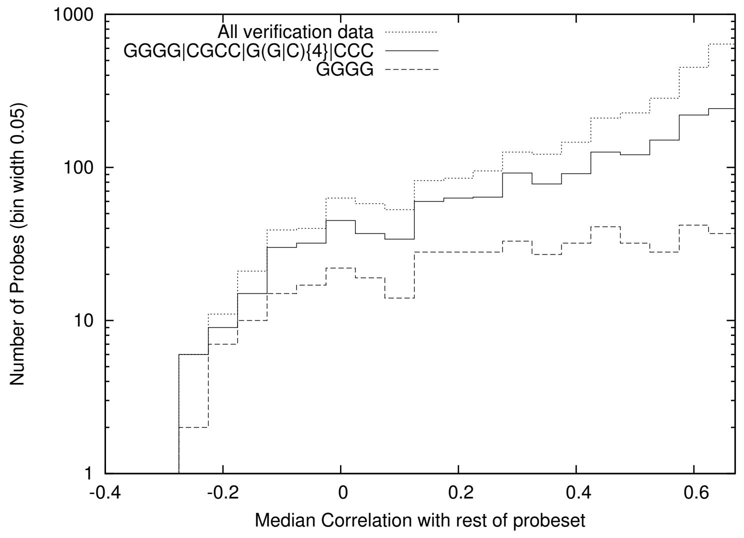

Figure 10: Performance of evolved (solid line) and human (dashed) generated motifs on verification data (dotted). The search string automatically created by GP finds more bad probes (left) but at the cost of flagging more good probes as suspect (right hand side).

Table 2: Performance of motifs. Number of probes matched.

`GGGG|CGCC|G(G|C){4}|CCC`

Whole training set			Test set		
Median Correlation	< 0.3	≥ 0.3	Median Correlation	< 0.3	≥ 0.3
	410	4448		$\underline{448}$	4436
-v	173	10061	-v	174	10045

`GGGG`

Whole training set			Test set		
Median Correlation	< 0.3	≥ 0.3	Median Correlation	< 0.3	≥ 0.3
	195	479		$\underline{209}$	434
-v	388	14030	-v	413	14047

The confusion matrix for the evolved regular expression on the whole of the training set (including the 6677 middling values which GP never saw) is given in Table 2 (left) and on the verification data Table 2 (right) (The corresponding matrices for `GGGG` are included below that of the evolved motif in Table 2. Unlike in many machine learning applications, there is no evidence of over fitting. Indeed the corresponding results for the test set (second matrix of each pair) are not significantly different (χ^2, 3 dof) from those on the whole training set. The evolved regular expression picks up significantly more (χ^2, 3 dof) (448 v. 209) poorly performing probes on the validation set than the human produced regular expression. Figure 10 shows the number of DNA probes matching the evolved motif against their average correlation with the rest of their probeset.

5.1 Speed

As is common in evolutionary strategies [Beyer, 2001], almost all the time is taken by the fitness evaluation. In our case, elapse time is dominated by the command script which runs `egrep -c`. Typically this takes 8.5mS per individual.

Speed depends upon the size and complexity of the evolved regular expressions. Earlier in the runs (i.e. generation 9) they are on average shorter (17.2 v. 52.8 characters) and so the time taken to process each is only 5.4mS. (Note `egrep -c` is run on two files of 583 lines. Each line containing 25 letters). The time taken by `gawk` to process the BNF grammar, perform crossover, generate the regular expressions, etc. is negligible.

6 Discussion

Theoretical and empirical studies of GeneChips confirm that the behaviour of DNA probes tethered to a glass surface can be quite different from DNA behaviour in bulk solution. This is a new and difficult area and there are not deep pure Physics experimental results. Therefore experimental studies have concentrated on data gathered during normal operation of the chips [Langdon *et al.*, 2008].

Our automatically generated motif, suggests that in addition to Gs, Cs are important. Indeed the fact that only three consecutive Cs is predictive (whereas four Gs are needed) suggests that Cs are more important than Gs. It is known in GeneChips DNA C–G RNA binds more strongly than DNA G–C RNA [Naef *et al.*, 2006]. (Both C and G bind more strongly than A and T.) We are tempted to suggest that a CCC sequence on a DNA probe can act as a nucleation site encouraging the probe to bind to GGG on

RNA. Indeed the evolved motif suggests that four Gs and mixtures of five Cs and Gs might also form nucleation sites.

The sequence CCC is too short to be specific to a particular gene. GeneChips are designed on the assumption that only DNA and RNA sequences which are complementary to the full length of the probe will be stable. However studies have shown that nonspecific targets can be bound to GeneChip probes for several hours even if held only by the nucleation site. This may be why probes with quite short runs of either Cs or Gs can be poorly correlated with others designed to measure the same gene.

7 Conclusions

Access to the raw results of thousands of GeneChips (each of which costs several hundreds of pounds) makes new forms of bioinformatic data mining possible.

Millions of correlations between probes in the same probeset, which should be measuring the same gene, show wide variation. Evolution of regular expressions confirms previous work [Langdon, 2008; Upton *et al.*,] that the DNA sequences from which the probes themselves are formed can indicate poor probe performance. Indeed several new motifs (e.g. CCC) which predict probe quality have been automatically found.

Linux code are available via FTP `ftp://cs.ucl.ac.uk/genetic/gp-code/RE_gp.tar`

References

[Bäck, 1996] Thomas Bäck. *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms.* Oxford University Press, New York, 1996.

[Barrett *et al.*, 2007] Tanya Barrett, Dennis B. Troup, Stephen E. Wilhite, Pierre Ledoux, Dmitry Rudnev, Carlos Evangelista, Irene F. Kim, Alexandra Soboleva, Maxim Tomashevsky, and Ron Edgar. NCBI GEO: mining tens of millions of expression profiles–database and tools update. *Nucleic Acids Research*, 35(Database issue), January 2007.

[Beyer, 2001] Hans-Georg Beyer. *The Theory of Evolution Strategies.* Natural Computing Series. Springer, 2001.

[Brameier and Wiuf, 2007] Markus Brameier and Carsten Wiuf. Ab initio identification of human microRNAs based on structure motifs. *BMC Bioinformatics*, 8:478, 18 December 2007. GPBiB

[Brameier *et al.*, 2007] Markus Brameier, Andrea Krings, and Robert M. MacCallum. Nucpred predicting nuclear localization of proteins. *Bioinformatics*, 23(9):1159–1160, 2007. GPBiB

[Cetinkaya, 2007] Ahmet Cetinkaya. Regular expression generation through grammatical evolution. In Tina Yu, editor, *Genetic and Evolutionary Computation Conference (GECCO2007) workshop program*, pages 2643–2646, London, United Kingdom, 7-11 July 2007. ACM Press. GPBiB

[Handstad *et al.*, 2007] Tony Handstad, Arne J H Hestnes, and Pal Saetrom. Motif kernel generated by genetic programming improves remote homology and fold detection. *BMC Bioinformatics*, 8(23), January 25 2007. GPBiB

[Koza, 1992] John R. Koza. *Genetic Programming: On the Programming of Computers by Natural Selection.* MIT press, 1992. GPBiB

[Langdon and Banzhaf, 2005] William B. Langdon and Wolfgang Banzhaf. Repeated sequences in linear genetic programming genomes. *Complex Systems*, 15(4):285–306, 2005. GPBiB

[Langdon and Barrett, 2004] W. B. Langdon and S. J. Barrett. Genetic programming in data mining for drug discovery. In Ashish Ghosh and Lakhmi C. Jain, editors, *Evolutionary Computing in Data Mining*, volume 163 of *Studies in Fuzziness and Soft Computing*, chapter 10, pages 211–235. Springer, 2004. GPBiB

[Langdon and Buxton, 2001] William B. Langdon and Bernard F. Buxton. Evolving receiver operating characteristics for data fusion. In Julian F. Miller, Marco Tomassini, Pier Luca Lanzi, Conor Ryan, Andrea G. B. Tettamanzi, and William B. Langdon, editors, *Genetic Programming, Proceedings of EuroGP'2001*, volume 2038 of *LNCS*, pages 87–96, Lake Como, Italy, 18-20 April 2001. Springer-Verlag. GPBiB

[Langdon *et al.*, 2007a] W. B. Langdon, R. da Silva Camargo, and A. P. Harrison. Spatial defects in 5896 HG-U133A GeneChips. In Joaquin Dopazo, Ana Conesa, Fatima Al Shahrour, and David Montener, editors, *Critical Assesment of Microarray Data*, Valencia, 13-14 December 2007. Presented at EMERALD Workshop.

[Langdon *et al.*, 2007b] W. B. Langdon, G. J. G. Upton, R. da Silva Camargo, and A. P. Harrison. A survey of spatial defects in Homo Sapiens Affymetrix GeneChips. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 2007. Submitted.

[Langdon *et al.*, 2008] William B. Langdon, Graham J. G. Upton, and Andrew P. Harrison. Probes containing runs of guanine provide insights into the biophysics and bioinformatics of Affymetrix GeneChips. *Briefings in Bioinformatics*, 2008. In preparation.

[Langdon, 1998] William B. Langdon. *Genetic Programming and Data Structures*, Kluwer, 1998. GPBiB

[Langdon, 2008] W. B. Langdon. Evolving GeneChip correlation predictors on parallel graphics hardware. In *WCCI*, Hong Kong, 1-6 June 2008. IEEE. Forthcoming. GPBiB

[McKay *et al.*, 2006] Robert Ian McKay, Tuan Hao Hoang, Daryl Leslie Essam, and Xuan Hoai Nguyen. Developmental evaluation in genetic programming: the preliminary results. In Pierre Collet, Marco Tomassini, Marc Ebner, Steven Gustafson, and Anikó Ekárt, editors, *Proceedings of the 9th European Conference on Genetic Programming*, volume 3905 of *Lecture Notes in Computer Science*, pages 280–289, Budapest, Hungary, 10 - 12 April 2006. Springer. GPBiB

[Montana, 1995] David J. Montana. Strongly typed genetic programming. *Evolutionary Computation*, 3(2):199–230, 1995. GPBiB

[Naef *et al.*, 2006] Felix Naef, Herman Wijnen, and Marcelo Magnasco. Reply to "comment on 'solving the riddle of the bright mismatches: Labeling and effective binding in oligonucleotide arrays' ". *Physical Review E*, 73(6):063902, June 2006.

[Nikolaev and Slavov, 1998] Nikolay I. Nikolaev and Vanio Slavov. Concepts of inductive genetic programming. In Wolfgang Banzhaf, Riccardo Poli, Marc Schoenauer, and Terence C. Fogarty, editors, *Proceedings of the First European Workshop on Genetic Programming*, volume 1391 of *LNCS*, pages 49–60, Paris, 14-15 April 1998. Springer-Verlag. GPBiB

[O'Neill and Ryan, 2001] Michael O'Neill and Conor Ryan. Grammatical evolution. *IEEE Transactions on Evolutionary Computation*, 5(4):349–358, August 2001. GPBiB

[Poli *et al.*, 2008] Riccardo Poli, William B. Langdon, and Nicholas Freitag McPhee. *A field guide to genetic programming.* Published via `http://lulu.com` and freely available at `http://www.gp-field-guide.org.uk`, 2008. (With contributions by J. R. Koza). GPBiB

[Radcliff, 1993] Nicholas J. Radcliff. Genetic set recombination. In L. Darrell Whitley, editor, *Foundations of Genetic Algorithms 2*, pages 203–219, Vail, Colorado, USA, 26-29 July 1992 1993. Morgan Kaufmann.

[Ross, 2001] Brian J. Ross. The evaluation of a stochastic regular motif language for protein sequences. In Lee Spector, Erik D. Goodman, Annie Wu, W. B. Langdon, Hans-Michael Voigt, Mitsuo Gen, Sandip Sen, Marco Dorigo, Shahram Pezeshk, Max H. Garzon, and Edmund Burke, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 120–128, San Francisco, California, USA, 7-11 July 2001. Morgan Kaufmann. GPBiB

[Upton *et al.*,] Graham J. Upton, William B. Langdon, and Andrew P. Harrison. Affymetrix probes containing runs of contiguous guanines are not gene-specific. *Nature Biotechnology.* Submitted to.

[Whigham and Crapper, 1999] Peter A. Whigham and Peter F. Crapper. Time series modelling using genetic programming: An application to rainfall-runoff models. In Lee Spector, William B. Langdon, Una-May O'Reilly, and Peter J. Angeline, editors, *Advances in Genetic Programming 3*, chapter 5, pages 89–104. MIT Press, 1999. GPBiB

[Whigham, 1996] P. A. Whigham. Search bias, language bias, and genetic programming. In John R. Koza, David E. Goldberg, David B. Fogel, and Rick L. Riolo, editors, *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 230–237, Stanford University, CA, USA, 28–31 July 1996. MIT Press. GPBiB

[Wong and Leung, 1996] Man Leung Wong and Kwong Sak Leung. Evolving recursive functions for the even-parity problem using genetic programming. In Peter J. Angeline and K. E. Kinnear, Jr., editors, *Advances in Genetic Programming 2*, chapter 11, pages 221–240. MIT Press, 1996. GPBiB