NUANCE:
Naturalistic University of Alberta Nonlinear Correlation Explorer

Geoff Hollis & Chris Westbury

Department of Psychology, University of Alberta

P220 Biological Sciences Building

T6G 2E9 Edmonton, Alberta, Canada

[This manuscript is in press at 'Behavior Research Methods']

Correspondence on this submission should be addressed to Chris Westbury at the above address or:

Tel: 780-492-8518, Fax: 780-492-1768

E-mail: chrisw@ualberta.ca

Acknowledgements: This work was made possible by a National Engineering And Science Research Council grant from the Government of Canada to Chris Westbury. We gratefully acknowledge the many useful suggestions we received in response to an earlier version of this manuscript and software, from Jonathan Vaughan, Greg Francis, Ian Walker, an anonymous reviewer, and Gail Moroschan.

In this paper we describe the Naturalistic University of Alberta Nonlinear Correlation Explorer (NUANCE), a computer program for data exploration and analysis. NUANCE is specialized for finding non-linear relations between any number of predictors and a dependent value to be predicted. It searches the space of possible relations between the predictors and the dependent value by using natural selection to evolve equations that maximize the correlation between their output and the dependent value. This paper introduces the program, describes how to use it, and provides illustrative examples. NUANCE is written in Java, which runs on most computer platforms. We have contributed NUANCE to the archival web site of the Psychonomic Society (http://psychonomic.org/archive), from where it may be freely downloaded.

"*Facts are nothing without their nuance, sir.*"

- Norman Mailer, upon being told by the judge to stick to the facts during the 1969 trial of the Chicago 7

Genetic programming (GP) is a way of using natural selection to program computers (Koza, 1992). In distinction from genetic algorithms and neural networks, which use idiosyncratic and human-unfriendly binary representations of problems and their solutions, GP evolves ordinary computer code. The output it produces are strings of code in a human and machine comprehensible language. Westbury, Buchanan, Sanderson, Rhemtulla, and Phillips (2003) suggested that this makes GP particularly useful in solving the kinds of non-linear multivariate regression problems that are common in psychology and other scientific disciplines, and discussed some of the statistical and conceptual issues involved in using such tools in scientific work. In solving non-linear multivariate regression problems, it is it often highly desirable to be able to see exactly how multiple independent variables relate to a particular dependent variable. In this paper, we introduce a free new tool that is has been specially designed for this purpose: NUANCE (Naturalistic University of Alberta Nonlinear Correlation Explorer). Written in Java, NUANCE will run on most computer platforms. It can explore and suggest human-readable solutions for any problems in which one or more predictors may predict the value of a single dependent value. It is easy to use, fast, flexible, and powerful, incorporating a new rank-order based fitness function that is demonstrably superior to the averaged multi-test fitness (described below) introduced by Westbury et al (2003).

Our goals for this paper are twofold. First, we aim to describe the interface of NUANCE, so that anyone will be able to use it after reading this paper. Secondly, we aim to demonstrate that NUANCE works, and that the novel algorithm it introduces is superior to averaged multi-test fitness, which was itself introduced to address problems of particular relevance to curve fitting.

How does GP work?

The general principles of GP are simple. In this section we describe them in general terms; in later sections we will fill in the details as they apply to NUANCE.

To begin, one needs to define two elements: *a function set*, and *a fitness value*.

The *function set* consists of the set of computational operators that are available for use in any evolved solutions. Although these may be highly specialized for the specific problem under consideration, they may also consist simply of a set of plausibly relevant basic mathematical functions.

The *fitness value* is a way of quantifying the goodness of any evolved solution. These may be a very general measure, such as a squared error, a percent hit rate, or a correlation of an evolved function's output with known correct values. The function that calculates the goodness of any solution is called the *fitness function*, and the goodness measure is called the *fitness value*.

Once a *fitness function* and *function set* are defined, evolution of solutions may begin. In the first generation, many (at least hundreds, more often thousands) of potential solutions are randomly generated, by conjoining the operators in the function set in legal ways. Since these solutions are randomly generated, they are likely to be very poor at solving the problem. However, because the fitness function is defined, these thousands of

poor solutions can be rank ordered for their utility in solving the problem. The best functions are identified, by using some *selection criterion*, and retained. The rest of the population is discarded. The selected functions are randomly broken into subparts, of random size ranging from a single node to the whole tree. The subparts are then randomly conjoined to fill the population of the next generation.

In sum, there are three steps to GP: generate functions, select the best, and regenerate new functions by mixing up the selected best functions. As this brief description suggests, the process used in GP is closely analogous to selective breeding in biology. In selective breeding, the breeder decides which animal is suitable to breed, based on what characteristics are desired. In GP the breeder's choice is automated, instantiated in the fitness function and selection criterion. The animals to be bred are mathematical equations.

By repeated application of the three simple steps, GP can incrementally improve the average and best fitness of each succeeding generation (for formal analyses, see Koza, 1992; Holland, 1992). The process can be stopped when a 'perfect' solution is found, if such a solution exists, or it may be stopped because some other completion criterion is met: for example, because the slope of the fitness function is close to 0, or because some set number of generations has passed.

Multivariate estimation problems of the type handled by NUANCE are well suited for GP because they have a natural fitness function. In such problems fitness may be simply defined as the absolute value of the correlation of the estimate with the data to be estimated. Because correlations are well-defined and continuous across the bounded range of –1 to 1, it is possible to compare the output of any two estimator equations and

have the computer unambiguously decide which one is better: the one with the larger absolute correlation with the data to be estimated.

This brief overview should make clear that understanding any GP system is largely a matter of understanding three things: the function set, the fitness function, and the selection criteria. In the first section that follows, we consider NUANCE's function set. In the second section, we consider its fitness function and selection criteria, which are combined in NUANCE.

Understanding NUANCE: The function set

NUANCE uses a flexible approach to its function set. It has a set of functions built-in, which users may turn on or off. NUANCE also allows users to add their own custom functions.

Most of the built-in functions are simple well-known functions of one or two inputs: addition, subtraction, multiplication, division, natural logarithm, square, cube, square root, cube root, absolute value, sine, cosine, and tangent. All these functions are enabled by default.

NUANCE does not include a general power function, as it is too easy to generate overflow errors using large (absolute) powers. However, note that all low power/root functions can easily be constructed by NUANCE during evolution by concatenating the square, cube, square root, and cube root functions. Moreover, such power functions may be explicitly added to the user-defined functions.

NUANCE also includes four multi-input built-in functions. The first is an if/then/else statement. This statement takes three arguments. If the first argument is 1, it

returns the second argument, otherwise it returns the third argument. The remaining three functions are > (*greater than*), < (*less than*), and = (*equals*) comparators. They each take two arguments, returning 1 if the specified relationship holds, and 0 otherwise. The use of these functions allows NUANCE to evolve *radically non-linear solutions*, which have discontinuities. However, their use also causes huge increases in the size of evolved solution trees, thereby slowing down the evolutionary process and decreasing the probability that the solution will be humanly comprehensible. Moreover, it is also possible to use nested if/then statements to generate trivial solutions to any problem, by building 'lookup tables' that mirror the input file. These functions should therefore be used judiciously: Include them whenever there is reason to suspect that the solution may be very complex or discontinuous, and exclude them by turning them off if not. These functions are off by default.

It is our intention to define all the functions that most users will want inside NUANCE, and we will add more as they are deemed necessary. However, we have also provided a means for users to add their own functions. At the bottom of NUANCE's 'Function Set' Panel, there is a button called 'custom functions'. When clicked, users can add and delete their own, programmable functions. Users are required to provide a unique name for each function, and a description of what the function does. Descriptions must be written in prefix notation, with the operator in front of its arguments, just like all functions NUANCE produces. The input parameters for each function must be designated within the function description using special symbols (N1, N2, N3, …, Nn). When the function is used, subtrees attached to the custom function will replace these symbols.

For example, the function *(+ N1 2)* will add two to the subtrees attached to it. The function *(* N1 (* N1 N1))* will return the cube of any node it takes as an input value. It is allowable to use more than one parameter in a user-defined function. For example, the function *(+ (* N1 (* N1 N1)) (sin N2))* will add the sin of one input to the cube of another.

In general, simple combinations of primitive built-in functions that are useful to the problem need not be explicitly defined by the user, as they may be reasonably expected to be discovered by the evolutionary process itself. In addition, it is possible that the provision of very fit custom functions may harm the evolution process. If the fitness provided by a custom function is much greater than that provided by the basic operators, NUANCE may have a tendency to converge on a solution that is nothing more than the custom function, rather than exploring the solution space as it is intended to do. Custom functions may not necessarily be useful, and should be defined only if necessary.

Understanding NUANCE: The fitness function and selection criteria

In this section we discuss NUANCE's *fitness function*, which ranks evolved equations, and its *selection criterion,* by which a subset of ranked equations is selected for breeding to fill the next generation.

Any solution to a problem that generalizes from a small set of data to a general relation faces the danger of *over-fitting*, by finding solutions that are highly suited to the dataset on which they were evolved, but which generalize poorly to other datasets. We say that such solutions are *brittle*. Brittleness is defined as an evolved solution's inability to generalize to unseen problem cases that are part of the problem of interest, but that

were not used as a fitness set for the GP system. For instance, if one were to try and evolve a control behavior to direct the foraging of an artificial ant, an ant's behavior would be considered brittle if it could not forage with competence in an environment different from the one it is trained in. For NUANCE to have any utility, it must be able to create generalizable summaries of the datasets that it models. In other words, NUANCE must avoid overfitting its solutions or producing brittle solutions.

Kushchu (2002) and Moore & Garcia (1997) both suggest the same solution for avoiding brittle solutions: the use of randomized learning cases. If we change the problem that is used to assess fitness on each generation, it presents an environment that selects for generalization. For example, in a scenario of an evolved simulated ant, Kushchu (2002) did this by randomizing the starting location of the ants on each generation. He showed that this produced much more adaptive behavior to new environments than having the ants start their foraging from the same place each generation.

In NUANCE, we use an analogous strategy by using a new random subset of the input data to gauge the fitness of each generation. One problem with this method is that it runs the risk of culling out evolved functions that actually do generalize. This can happen if a subset were generated that happened by chance to be highly unrepresentative of the problem of interest. Poor fitness with that subset could wipe out many evolved functions that actually do generalize, thus setting back the evolutionary process.

Westbury et al. (2003) introduced a technique called averaged multitest fitness (AMTF) that lessened, though it did not necessarily remove, this problem of losing fit functions. AMTF builds cross-validation into its fitness function, in way that is analogous to the k-fold cross-validation technique (Stone, 1974) that is often used in training neural

networks. The algorithm takes the average correlation of an equation's output with the measure to be predicted over multiple random subsets. It uses this average as a gauge of an equation's fitness.

AMTF has the disadvantage of requiring multiple fitness tests for every evolved function. This makes it computationally costly, thereby slowing down the speed at which evolution can proceed.

We have recently developed a novel fitness method that is superior to AMTF for a number of reasons, which we call age-weighted fitness. This is the default method used within NUANCE, and is discussed in the 'Run-time parameters' section below.

For further discussion of the problem of over-fitting, see Example 3 below.

Using NUANCE

NUANCE is simple to run. In this section, we describe the steps.

NUANCE is written in Java. Java is freely available for most common computer operating systems. NUANCE requires Java 1.4.2 or later. If you need a copy of the Java Virtual Machine in which NUANCE runs, you can download one for most platforms from http://java.sun.com/j2se/downloads.html.

NUANCE can be downloaded from the Psychonomic Society archive at http://psychonomic.org/archive. NUANCE comes packaged as a Java Archive (JAR) file, and is run by double-clicking on the file icon. Java archives open more slowly than regular applications, so be patient.

After opening NUANCE, you will see a graphical interface with three tabs labeled 'NUANCE', 'OUTPUT', and 'FORMAT CONVERTER'. We will describe each of these three panels in turn.

The NUANCE panel

The NUANCE panel is the main panel for using NUANCE. On this panel you can control the main function of NUANCE, by a setting the evolution parameter values and the input and output files, choosing which functions to allow, and turning the evolutionary process on and off.

In the pane to the left, you will see check boxes for each of the built-in functions described above. You can turn these on or off by clicking on these buttons.

The middle pane on the right, entitled 'INPUT AND OUTPUT FILES' allows you to specify the input files and output directories for NUANCE. The input file must be a tab-delimited text file. By default it is a file named 'dataset.txt' that goes in the same directory as NUANCE. You may change this by clicking on the 'change' button beside the file path field, or by typing a new file path directly into that field. The first line of the input file must consist of the variable names, separated by tabs. Beneath this are the columns of data, also separated by tabs. The first column must contain the independent measure to be predicted. The remaining columns must contain predictors. NUANCE does not accept missing data, so missing values must be replaced with an estimate, or rows that contain missing values must be deleted.

You must also specify a directory where NUANCE will print its output. This defaults to a time-stamped directory, whose name reflects the time NUANCE was started. That directory will be written by default inside the 'logs' directory at the same level as NUANCE itself.

In each output directory, NUANCE will create sub-directories that are numbered for each run: 'run 1', 'run 2', and so on. Each of these output sub-directories contains two

files: "history.txt" and "log.txt". The "history.txt" file will contain a log of all the output NUANCE displays on screen. The "log.txt" file contains information about the whole run that might be of interest: average time per generation, average length and age of functions in that generation, the best function of that generation, and its age and fitness.

When a specified set of runs is completed, each output directory also contains four other files, named 'settings.txt', 'functions.txt', 'best_estimates.txt', and 'graphing_data.txt'.

The 'settings.txt' file simply records the parameter settings that were used for those runs.

The file named 'functions.txt' contains a list of all functions, including custom functions, used during the runs.

The 'best_estimates.txt' file contains the best evolved function across all runs, in both prefix and infix notation, as well as a list of the output of that function for each data point in the input file. It is possible that some cases may be undefined. This can happen if a particular case that was never selected as part of the testing subset of a given function is undefined for that function. Undefined values are replaced with ‘NaN’ [‘Not a number’].

The 'graphing-data.txt' file contains a different representation of the same best function. It includes one 10 x 10 table for each pair of predictors. Each table spans across the range of the two predictors in 10 equal steps. Each of the 100 cells in the 10x10 table contains the normalized value of the output function over the ten steps of the two predictors. If there are more than two predictors in the data file, the others are fixed at their average value for the purposes of building the table. The results in each cell are normalized (expressed in z-scores) because the output units of evolved functions are

arbitrary and therefore may be difficult to interpret. The purpose of this table is to facilitate visual inspection of the shape of the best function, and of the nature of the relation between any two predictors. The output table can be pasted into any other program with graphing capabilities in order to create the graph. We have included with NUANCE an Excel template (‘GraphingTemplate.xls’) for creating such graphs.

NUANCE's leftmost pane lists of all functions and operators used by NUANCE. The default setting is to turn all functions on, except for the *logical if*, and the *greater*, *less than*, and *equal* operators. This default can be overridden by putting a list of the function names you wish to use in a file called 'functions.txt', located in NUANCE's directory. Custom functions can also be added to this list; the format for doing so is to place the function's name and description on a single line, separated by a tab.

In the top pane on the right, you will see a set of all NUANCE's parameters and their values. NUANCE will always load its default parameters from a file called 'settings.txt', located in the same directory as NUANCE, if that file exists. You can edit this file directly to change the default settings that NUANCE loads with, or you can click on the ‘save settings’ button in NUANCE to write the displayed settings as the default settings for future use. You can also change the settings manually in the top right hand pane. We now describe the parameters in detail.

NUANCE's run-time parameters

Population size

The *population size* parameter specifies how many equations can be in the evolving population at any time. In general this variable can be fruitfully maximized, but with two caveats: the time it takes to run with larger populations will increase linearly

with population size, and NUANCE may require more RAM. The default heap size for java applications is 64mb. If memory needs to be increased, run the program from the command line. To do so, type 'java –mx###m –jar nuance.jar', where ### is the number of megabytes of RAM you would like to allocate to NUANCE. 128mb of RAM should be sufficient for all populations under size 10,000.

The default population size is 2500, which is probably fine for most problems. Increase it if you feel doubtful that the best solution is being found. A population size above 10,000 would be very large indeed. We have successfully run NUANCE with populations as large as 20,000.

Generations

The *Generations* parameter specifies how many generations you wish to evolve. It is difficult to give a general value to be entered here, as the number of generations required may vary with the complexity of the problem. As with many of the other NUANCE parameters, the rule of thumb is: the more the better. Since in the end you will probably keep only the best solution, you can never do worse by running more generations, and you will probably (and, in the long run, almost certainly) do better. However, running more generations takes more time.

It is possible to estimate if one has run sufficient generations by looking at the shape of the fitness curve, which graphs fitness by generation (for examples, see Figures 1, 2, 5, and 6). If the curve is nearly flat, then little progress is being made in each generation, and it is unlikely that further computation will lead to substantial improvement. If the curve has a clear positive slope, more computation will probably lead to further gains. However, this heuristic may sometimes be misleading. The

stochastic nature of evolutionary search means that it is possible for progress to be made after many generations of no progress. An example of this phenomenon is shown in Figure 1, which shows a NUANCE run, using a public data set, that tried to predict average January temperature in 60 American cities from average rainfall in those cities. After 47 generations with an almost flat fitness function (r = 0.11), the run graphed here found a substantially better region of search space and in one generation jumped to r = 0.29. Thereafter fitness steadily increased. For this reason it is advisable to allow leeway in the number of generations of search. The default value is 100 generations.

Runs

You can set the *runs* parameter to any N > 0. NUANCE will repeat evolution N times, with all of the settings already defined. It is desirable to repeat runs several times because of the stochastic nature of evolutionary computing. There is no guarantee that any run is the best possible. We recommend that you run at least 8 to 10 times, or more if you can, although clearly the best solution in hand may be deemed sufficient for any number of reasons at any given time. The default value is 10.

Starting Tree Depth

Users are allowed to specify how large their starting functions will be, by specifying how deep each function tree is. The default starting depth is 3, which means an average of 5 or 6 nodes per function. The final functions produced by NUANCE rarely exceed depths of 10, and this parameter should not be set higher than that. Such deep functions are very difficult for humans to understand.

Although the default starting tree depth should suffice for most purposes, longer and shorter values may be of benefit in some situations. Without allowing shorter starting

trees, it is possible that you may miss some very simple solutions. However, this is not a very serious problem, because you will almost certainly be able to deduce the missed solutions from the solutions that NUANCE offers (see Example 4c below). In some cases it may desirable to set the starting depth higher than the default. It is possible, when the best solution tree is deep, that short solutions may lead to premature convergence to regions of solution space that are suboptimal. If you encounter a problem in which you feel dissatisfied that a good solution has been found, you can try increasing this parameter.

Parsimony Pressure

One of GP's major limitations is that functions may get so large that they are either completely incomprehensible, intractable to run, or a combination of the two. A number of theories have been proposed to explain this phenomenon, which is commonly termed 'bloat' (see Soule & Foster, 1999; Langdon, 2000; Streeter, 2003). Some (Streeter, 2003; Soule & Foster, 1998) speculate that large size is actually evolutionarily adaptive, since it can lessen the probability that crossover will damage offspring. Others (e.g. Langdon & Poli, 1997) suggest bloat may be caused by distribution of fit functions in solution space. Solutions with equal fitness can have many different physical forms. With variable-length functions and any arbitrary cutoff to separate 'long' from 'short', there are typically more long solutions than short solutions, and thus long solutions have a bias to be found first.

Whatever the cause of bloat, it is undesirable from the NUANCE user's point of view. We have implemented parsimony pressure to prevent functions from growing excessively large. Parsimony pressure penalizes long solutions. By setting parsimony

pressure to a positive value, a function's fitness is decreased by a percent value equal to the parsimony pressure times the number of nodes (operators and arguments) in the function. For example, a parsimony pressure of 1 would decrease the fitness of a function whose length is 100 by 1%. For a length of 150, fitness would decrease by 1.5%, for a length of 200, fitness would decrease by 2%, and so forth.

A small parsimony pressure can have a large effect on function size, since it discourages increases in function length except where a long function is markedly better than the best short function. On the dataset that is discussed as Example 1 below, even the smallest parsimony pressure of 1% reduces average function length to less than $1/5^{th}$ of its value without such pressure (see Figure 2). The default parsimony pressure for NUANCE is 1.

Mutation Rates

Randomly mutating members of an evolving population is a way to add diversity to the population pool. Many people working with GP have deemed mutation to be generally ineffective (Koza, 1992; Luke & Spector, 1998). However, it may be a worthwhile addition to NUANCE in some specific situations. If many solutions become too similar too quickly (i.e. they converge to a local optima), evolution may halt. Mutation can help prevent such problems by introducing new genes into the population pool. It does so by randomly changing some percentage of nodes in the children that are seeded into a population. By default, the mutation rate is turned off, by setting it to 0. Users may wish to increase this value if the best function's fitness stops increasing very early in the run. If fitness still fails to increase even with mutation on, it is probable that there are very few solutions better than the best found. If on the other hand turning on

mutation helps to increase fitness, it is likely that there was a problem with getting stuck on local optima.

Testing Subsets & Subset Size

As mentioned in the introduction to this paper, to avoid problems with over-fitting, a GP program can use small portions of the dataset for testing population members. Overall fitness is the average of the fitness from all subsets. In this way, the GP system treats the input dataset as if it were made up of multiple smaller subsets, and tries to find solutions that generalize between these subsets. In NUANCE, there are two parameters associated with this ability: *Testing subsets* and *Subset size*.

The first parameter, *Testing Subsets*, is the number of different subsets that the population members are tested on each generation. In some cases, any particular subset may be unrepresentative of the problem being solved, which means that there is always a chance of killing fit functions because they cannot accurately predict that unrepresentative data. Increasing the value of this parameter to some number greater than 1 allows NUANCE to use the *averaged multi-test fitness* described above. Each evolved function is tested on multiple subsets. Averaging the function's fitness on all subsets gives the function's fitness for the generation. Data will never be included more than once within any given subset. However, they may be included more than once across multiple subsets.

The second parameter, *Subset Size*, is the size of the dataset(s) that will be used for testing the population members on each generation, expressed as a percentage of the total dataset size.

We consider these two parameters together here because their optimal values depend on each other. The general rule of thumb is that the *Testing Subsets* parameter should be as high as possible (so each function is tested on as many subsets as possible), while the *Subset Size* parameter should be as low as possible (so that each subset is small compared to the overall dataset size). However, these optimal settings depend to some extent on how many rows your input set has. A parameter value above 2 is desirable in order to maximize generalizability, but it is better to have at least 15 or 20 rows in each subset than it is to have many subsets. If the *Subset Size* is low enough so that there are very few data points in each subset, no evolution will occur: the system will simply thrash about randomly, unable to find an equation with a consistent fit. Finally, the number of rows specified by combining these two variables should ideally be a relatively small fraction of the total dataset size. If you specify a *Testing Subsets* size of 2 and a *Subset Size* of 50%, you will be using your entire input dataset to assess the fitness of every function, which will expose you to the risk of over-fitting.

The default values are a *Testing Subsets* size of 1 and a *Subset Size* of 20%, on the assumption that large datasets are rare. These parameters mean that the fitness of each function is tested on a random fifth of the input dataset. If you have more than 100 lines of input, you should increase *Testing Subsets* to 2, and perhaps decrease *Subset Size* to 15%. This will assess every function on two random subsets of the input dataset, each comprised of 15% each of that dataset. Every function will therefore be tested on a randomly-selected 30% of the input dataset.

Fitness age weight & Selection Strategy

Large datasets are rare. In problems where one cannot collect large samples of data, it is impossible to use multiple subsets to prevent unrepresentative data from killing fit population members without over-fitting. NUANCE therefore includes other tools to deal with this problem. By adding a bonus to fitness based on a function's age (the number of generations it has existed for), NUANCE can get a measure of how well functions fare when dealing with different data. If a very unrepresentative testing subset is generated, this age-weighting can act as a protective buffer for fit functions if they have survived for multiple generations (presumably because they are, indeed, fit).

The *Fitness age weight* parameter specifies how strong the weighting should be. The exact way that age-weighting affects fitness is as follows:

<new fitness> = <old fitness> * (1 + log<*Fitness age weight*>(age))

The benefits of age-weighting are dependent on the selection strategy: that is, how the program decides which functions should be allowed to produce offspring at the end of each generation. NUANCE offers two selection strategies, which can be chosen by a pull-down menu: *Greedy Over-selection* and *Median Pass* selection. Age-weighting is useful with the second strategy, but cannot help with the first. In order to understand why, it is necessary to understand how each strategy works.

Greedy Over-selection was the selection method introduced by Koza (1992) and used by Westbury et al. (2003). It is usually called *Fitness-proportionate selection with greedy over-selection*. Under this selection scheme, the population pool that represents 16% of the normalized fitness is selected for reproduction with an 80% probability, and the rest of the population is selected for reproduction with a 20% probability. Koza (p. 98) specifically recommended it for 1/3 of difficult problems in his book that required

populations of more than 1000. However, computer technology has advanced greatly since 1992. We can think of no reason to ever use a population smaller than 1000, and suggest using populations several times that size.

Greedy Over-selection does not work with age-weighting because it allows too small a proportion of the population to survive each generation. It thereby eliminates most of the important age information that age-weighting needs. Because of this limitation, we have implemented *Median Pass* selection. This selection scheme removes the least fit 50% of the population in each generation. It replaces that 50% with children of the top 50%, in such a way as to guarantee that every member of that population will get to parent at least one child. The second parent of each child is chosen at random from the parenting population.

NUANCE automatically implements a redundancy check with both selection schemes. This replaces any identical twins in the population with new functions, in order to maximize the population diversity and minimize computational redundancy.

NUANCE's default settings use *Median Pass* selection, because it is the more computationally efficient selection strategy. There are two reasons for this. One is that *Median Pass* selection is not so dependent on (though it may still benefit from) having a large dataset that allows averaging of multiple measures for each fitness value, because it can spread out that averaging across generations. The other is that the fitness barrier imposed by *Median Pass* selection is more conservative than that imposed by *Greedy Over-selection*. The use of age-weighting imposes a penalty on new equations that, by chance, just happen to do well on the current subset of the input file that is being used for assessing fitness. In contrast, when using *Greedy Over-selection* as a selection strategy,

an equation that does well because it happens to be especially fit to the current fitness subset will be allowed to seed the next generation with its children. *Median Pass* selection is also more likely to promote functions that were created earlier, because age counts. Since older functions are almost always shorter than newer functions, equations tend to get longer and more complex under *Greedy Over-selection* than they do under *Median Pass* selection. A related result is that evolution under *Greedy Over-selection* is usually slower than evolution under *Median Pass* selection

Greedy Over-selection has one advantage over *Median Pass* selection: it tends to perform slightly better at accounting for variance in the dependent variable, for the same reason that it is slower: because larger, more complex equations can evolve and be tested. The prices to be paid for this increase in explained variance are two-fold: slower evolution, and much more difficulty in understanding the evolved equations. Whether or not this trade is worthwhile will depend on the nature of the problem under consideration.

We have conducted several test runs to try to determine what the best value for the *Fitness age weight* parameter should be. Our tests have made clear that large bases for the age-logging function tend to be better than low bases, which over-value age relative to fitness. However, it has proven difficult to definitively select one large base value over another, as there is much variance in performance on different data sets. In the end, NUANCE uses (log1000(age+1) + 1) as the default value.

One piece of motivating evidence for this value is shown in Figure 3. In this example, we used NUANCE to predict lexical decision times using a large set of lexical variables as predictors. Figure 3 graphs the probability that a run using a given age-weighting factor would contain the most predictive equation in each of three consecutive

bins of 33 generations. The probability was maximized in the final bin of 33 generations when the weighting scheme used a base of 1000. Moreover, only this weighting scheme showed a monotonically increasing probability of having the best function across all three bins.

Note that Figure 3 also clearly shows that lower base values perform worse. Bases below 500 have a much lower probability of containing the best function than higher bases.

Notwithstanding this evidence, Figure 4 illustrates why we need not worry too much about whether or not we have identified the optimal value for the age-weighting scheme. This figure graphs the average correlation (on the same data set as used in Figure 3) of the best performing function in 20-generation bins for several different age-weighting schemes. The final performance of the different schemes is closely comparable. The correlations of the output of best functions in the final generation- Generation 100- with the value to be predicted range between 0.5399 (using Log5000) and 0.5484 (using Log2000), amounting to a negligible difference of 0.06% in the amount of variance accounted for in the dependent variable by the best and worst evolved equations at Generation 100.

Local Search Priority

Genetic Programming is designed to be a global search technique. It sends multiple agents out into the search space, all in their own directions, each trying to get to the best point in the fitness landscape around them, rather than each population member attempting to get to the best known point in the fitness landscape. Because of this, it may

be the case that the best known places in the search space are not as rigorously explored as they could be.

The *Local Search Priority* parameter allows the user to force 'micro-exploration' around the best known solution. It does so by generating a number of mutant offspring of the best function in the population pool. The number of mutants to be created is equal to the value of the *Local Search Priority* parameter. Each mutant is created by replacing one random subtree of the best function with a new, randomly-generated subtree. The fitness of each mutant offspring is compared to that of its parent. If an offspring's fitness exceeds that of its parent, its is added to the population pool. Otherwise, it is discarded. This may potentially bring the total population size to *local search priority + N*, although this will only happen if all mutant offspring exceed the fitness of their parent. Population size does not ever increase beyond this size, since each new generation is re-set to the specified population size before adding any mutants.

We have not yet been able to document any compelling evidence that searching this space with mutation does in fact provide an evolutionary advantage. However, since the cost of allowing mutation of the best function is low and the potential advantages seem clear, we have left it in as an option.

It is possible that using too large a value for this parameter may lead to exponential growth in the population of mutants that are related to the 'ancestral' best solution, with a resultant under-exploration of other fruitful regions of the solution space. We therefore suggest that this parameter be set low compared to the total population size. The default value is 20.

Functions to display

The *Functions to display* parameter allows you to specify how many of the best functions from each generation you want to display. If the value is 1, only the single best function of each generation will be shown. The optimal value for this display parameter is purely a matter of personal taste. The default value is 5.

The OUTPUT Panel

The second tab in the graphical user interface takes the user to the OUTPUT panel. This is a tool intended for testing the goodness of fit and generating graphable tables of any evolved function on any dataset. It can be used to examine functions from NUANCE's evolutionary history, to cross-validate functions from one dataset to another, and to explore how small changes may impact on a function's fitness.

There are two fields. The top 'Function' field is for the function to be examined- it can be pasted or typed in. The second 'Dataset' field allows the user to specify which dataset the equation in the 'Function' field should be used with. When the user clicks on the 'print output' button, the specified equation is applied to the specified dataset. The output contains the first column of the input file (the dependent variables) and the value generated by applying the specified equation to each row of that input file. Beneath these two columns their correlation is given. Finally, 10x10 tables are printed for each pair of predictors. These are the same as the tables, described above, that are written to the 'graphing-data.txt' file: Each table ranges across the range of the two predictors in 10 equal steps, and each cell contains the normalized value of the output function over the ten steps of the two predictors. When there are more than two predictors, the others are fixed at their average value.

The FORMAT CONVERTER Panel

The final panel is the FORMAT CONVERTER panel. This contains tools for re-representing NUANCE's prefix equations, which place the operator at the front of its arguments. This representation is uncommon, and may be particularly difficult to decode when the equations are complex because of deeply nested parentheses. The converter is easy to use: just paste or type a prefix equation into the top panel, and then click on one of the three converter buttons to convert to a new notational system. Three notational systems are available. The first is prefix notation, the more commonly used but less consistent representation that places the operator between its arguments, when there is more than one argument, and before its argument, when there is just one argument. The second representation is that used by the mathematical software, Mathematica (Wolfram Research Inc., 2003). This notation can be pasted directly into a Mathematica notebook, in order to simplify the equations using Mathematica's 'Simplify' command. Finally, prefix equations can be converted to LaTeX notation. This is a notation that can be pasted into compatible programs to produce typeset mathematical output. Many LaTeX typesetting programs are available for free.

Four examples

In this section we briefly present four examples of how NUANCE can be used.

Example 1: Predicting lexical decision RTs from two lexical variables

Our first example uses the same data set as was used in Example 1 of Westbury et al (2003), which looks at how well orthographic neighbourhood (ON) and word frequency can predict lexical decision reaction times (RT). This data set was originally chosen because the relationship between the variables has been well studied and is of theoretical interest, and because data are easy to obtain. It is of additional interest here

because it provides us an opportunity to test NUANCE against a similar (but more limited) genetic programming system, the Common Lisp based *Greedy over-selection* system described by Westbury et al (2003).

ON is a measure of how many words in a given language are exactly one letter different from a target word. For example, the neighbourhood of the word *free* includes the words *tree*, *flee*, and *fret*, among others. Decades of experimental work (see Andrews, 1997, for a review) have demonstrated that ON has an effect on lexical decision task RTs. In a lexical decision task, subjects have to decide as quickly as possible whether a letter string displayed on the screen is a word or a nonword. Decisions to high ON words are made more quickly than low ON words when the words are low frequency. ON has no effect on high-frequency words. Westbury et al. (2003) showed that genetic programming was able to deduce this relationship with a high degree of precision by fitting the two parameters to lexical decision RTs from a large arbitrary set of words. Using 8 75-generation runs of size 2500, they evolved equations of the two variables whose output correlated with 450 RTs at 0.48 ($p < 0.01$). The equation generalized to 150 unseen data points with a correlation of 0.61 ($p < 0.01$). Relying on linear regression equations would cause one to under-estimate the size of the effect. Linear fits correlated with the input data set with $r = 0.22$ ($p < 0.01$) and generalized to the unseen data set with $r = 0.20$ ($p < 0.01$).

We used NUANCE to estimate the same 450-item dataset as had been used by Westbury et al. (2003), using both *Greedy over-selection* and *Median Pass* selection. We ran ten runs of 100 generations of population size 5000. Sample fitness curves for this problem are shown in Figure 5.

The best evolved predictor correlated with the RTs in the fitness set at |r| = 0.52 using *Greedy over-selection* ($p < 0.0001$) and |r| = 0.48 using *Median Pass* selection ($p < 0.0001$), about the same as the best previously found solution. The two estimates correlated with each other with $r = 0.87$ ($p < 0.0001$). They correlated very highly ($p < 0.0001$) with the best estimates from Westbury et al., at |r| = 0.86 (*Greedy over-selection* solution) and |r| = 0.99 (*Median Pass* selection solution).

We also tested the ability of the two evolved predictor equations to generalize to unseen data, by testing their ability to predict the RTs for 150 words that had not been in the fitness set. The best ever evolved predictor equation using *Greedy over-selection* correlated with the RTs in the entire fitness set at |r| = 0.52 ($p < 0.0001$). The best evolved predictor equation using *Median Pass* selection correlated with the RTs in the entire fitness set at |r|= 0.58 ($p < 0.0001$). These two predictions correlated with each other at $r = 0.75$ ($p < 0.0001$). The older Westbury et al. (2003) equation produced output that correlated with those unseen data at 0.60 ($P < 0.0001$). The correlation of the present output with that earlier estimate is 0.98 ($p < 0.0001$) for the estimates derived by *Median Pass* selection and 0.74 ($p < 0.0001$) for the estimates derived by *Greedy over-selection*.

The best evolved equations are shown in their raw form in Table 2, which illustrates the difference in complexity of the two solutions. The equation evolved using *Greedy Over-selection* is far more complex than the elegant simple equation evolved with *Median Pass* selection. The complex solution derived with *Greedy over-selection* was the best one at predicting RTs in the original input set. However, it had obvious weaknesses: it was highly complex, and it correlated poorly with RTs from a related data set and with other solutions evolved on the same dataset. The simpler *Median Pass* selection equation

is almost as good at predicting RTs in the input dataset as the complex equation, but also generalizes very well to the new dataset.

Overall, the fitness and test set correlations produced by NUANCE are all significantly correlated with those reported previously and discussed above, using the same dataset but an entirely different GP system. The close convergence of three different evolved estimates using different computers, fitness functions, and methods suggests that we may have a high degree of confidence that the solutions discovered are indeed the best solutions that may be found using GP.

Example 2: Estimating leukemia incidence from per capita cigarette consumption

Our second example uses published data from the Data And Story Library (DASL, at http://lib.stat.cmu.edu/DASL/). Originally published in Fraumeni (1968), this data set presents the relationship between average cigarette consumption per capita and leukemia deaths per 100,000 people in 44 American states in 1960. The linear correlation between cigarette consumption and leukemia deaths in this data set is insignificant, at –0.07 ($p > 0.05$). A cubic regression yielded the best (but also statistically unreliable) correlation we could find using the standard regression options in SPSS, with $r = 0.32$ ($p > 0.05$). We used NUANCE to see if it could find a transformation of the cigarette consumption predictor that would yield a better prediction of leukemia deaths from cigarette consumption.

We ran ten runs of 100 generations of population size 2500, using the default settings for other values. The average and best runs are shown in Figure 6. The best equation that was evolved was a complex equation that correlated with leukemia deaths at 0.53 ($p < 0.001$). This suggests that cigarette consumption may have a stronger

relationship to leukemia deaths than originally suggested, though the relationship may be a complex one.

The NUANCE-evolved estimate and the raw leukemia data are graphed, in standardized units, against cigarette consumption in Figure 7. This graph suggests hypotheses that may shed light on the relationship under consideration. In particular, the idealized NUANCE estimate suggests that the relationship between leukemia incidence and cigarette consumption may be quite linear when the number of cigarettes is low (below 23, which is the turning point of the sideways parabolic shape of the NUANCE estimates in the graph). Above this number of cigarettes the correlation no longer holds, and may even be reversed.

This possibility can be easily verified by going back to the original data. For the 21 states with average per capita consumption below 23 cigarettes, there is a statistically reliable linear correlation between leukemia incidence and cigarette consumption, with $r = 0.50$ ($p < 0.05$). For the 23 states with average per capita consumption above 23 cigarettes, the correlation between leukemia incidence and cigarette consumption is –0.06 ($p > 0.05$). The suggestion drawn from a brief glance at NUANCE's idealization of the data holds.

The proper interpretation of this fact is outside of our own range of knowledge. Perhaps leukemia incidence is lower when cigarette smoking is higher because the population is dying of other smoking-related disorders before they contract leukemia, or perhaps there are other mediating factors for which cigarette consumption is an appropriate proxy variable. For the purposes of this paper, we merely wish to demonstrate that NUANCE was able to quickly create a simplified summary of a

complex data set, which shed light on the structure of the relationship between variables and suggested hypotheses that may be worthy of closer study.

Example 3: The Significance of Random Data

One criticism that is often raised against curve-fitting programs like NUANCE is that they generate spurious solutions by over-fitting data. The third example is intended to both illustrate this danger, and to discuss how the danger may be avoided and appropriately conceptualized when using NUANCE.

We used Excel's random number generator to generate 500 quadruplets of random numbers. We then ran NUANCE to find a function that used the last three numbers to predict the first. We ran 10 runs, with a population size of 10000, using the default settings with *Median Pass* selection, with all primitive functions enabled. The best solution found used two of the three predictors to produce an equation that correlated with the first column at $r = 0.17$, which is a highly reliable correlation ($P < 0.001$) for 500 items.

Since the input was random, this highly reliable correlation may seem dismaying. The fact that curve fitting programs can reliably fit random data is what moves critics to dismiss their solutions as 'spurious' or 'over-fit'. However, the terminology used in such dismissals is misleading as to the real nature of the problem. If NUANCE returns an equation that fits random data reliably, it can only do so because the relation described by the equation actually exists. The algorithm used in NUANCE does not allow it to return erroneous fits. The equations returned cannot be 'spurious' or 'overfit' (as least with respect to the input dataset alone; see the discussion below); however, they certainly can be 'misleading' or 'uninteresting'. The temptation to use the stronger derogatory

terminology springs from a confusion of *description* with *explanation*. NUANCE can offer a mathematically guaranteed *description* of data relations, but it can offer no help at all in *explaining* those relations. It is up to human beings to decide whether any particular description of a dataset is of any theoretical or practical utility.

Westbury et al. (2003) discuss this problem, and its relation to degrees of freedom and psychological theoretical concepts, in some detail in the concluding discussion of their paper introducing GP as a tool for psychological theorizing. Their conclusion is that

> "GP is a way of 'cloning our imagination'. By providing it with hypothetical constructs and operators, we supply the prior understanding that constrains the search for an appropriate representation of relationships in the data. The fitness function instantiates and provides a quantifiable measure of what constitutes a solution 'worth imagining'.
>
> Equating a search through relation space to imagining ways of representing a problem is not a rejection of the role of theory or of the hypotheticodeductive method in science. It is simply a recognition of the fact that theoretical claims and empirically-based descriptions of the relations between theoretical elements must be separate. Theories can postulate which hypothetical constructs may play a role in any particular phenomena and may certainly attempt to specify the precise nature of that relation. However, when it comes to questions of the real nature of the relation between those constructs, theoretical claims cannot trump empirical facts. If a systematic relation between hypothetical constructs can be reliably demonstrated, by any means, then theory is not in a position to rule this empirical finding in or out– it can only adapt to the finding as it must adapt to any other relevant empirical evidence" (pp. 215).

The question of practical importance is: How can we decide if any evolved solution is 'worth imagining' or not? There can be no general answer to this question, because what counts as a worthwhile relation must depend largely on the current state of knowledge with respect to the issues under examination; the goodness of fit of the solution; and other specifics of each case. However, Westbury et al. offered two heuristics for assessing the worthiness of solutions evolved with GP.

The first suggestion is "simply to avoid giving the method inputs that might have a spurious relation to the problem– that is, avoid using GP for wild fishing expeditions" (p. 215). If we restrict ourselves to using NUANCE to describe relationships in which we have a pre-existing interest, then the question of how to interpret reliable random relationships simply does not arise.

If we choose to ignore this heuristic by using NUANCE to search for relationships in which we have no a priori interest (and we admit to having given in to this temptation ourselves), then it is incumbent on us to temper the conclusions drawn from such an undertaking, and to seek independent sources of confirmation for any conclusions we may wish to draw. When used for fishing expeditions, curve-fitting programs like NUANCE must be only a starting point, not an end point.

The second suggestion can help us to temper our conclusions appropriately. This is the suggestion (following Stone, 1974) that we cross-validate any curve-fitted solutions by 'holding back' some of the data which we give as input to our curve-fitting program, and then testing the best predictor equation on this unseen dataset. NUANCE has within-dataset cross-validation built into its selection mechanism, because it judges the fitness of evolved solutions on random subsets of the input dataset. However, it can still find solutions that capture dataset-specific variance, thereby over-estimating the general goodness of fit of the evolved solution. What it really means for a solution to be 'over-fit' is precisely that the solution relies on dataset-specific variance. Dataset-specific solutions will by definition perform very badly when tested on a different dataset than the one from whence they were derived. To illustrate this, we took the best-fitting equation to our random dataset above and used it to predict a different random dataset of 600 items. The

fit was not reliable ($r = -0.009$; $p > 0.05$), clearly demonstrating what we already knew: there is no *general* solution to the problem of fitting randomly generated data.

Together these two suggestions make clear why we need not be worried to discover that there are statistically reliable relations in a random data set. Since the relation is of no theoretical or practical interest, and since it fails to generalize to a related dataset, we have no reason to waste any calories in pondering it.

Example 4: Four Simple Problems

The three examples above suggest that NUANCE can fit data in complex ways. Does NUANCE choke when the fit is not complex? Is GP too powerful for its own good, obscuring the possibility of a simple equation through brute computational force? The answer to both questions is no. If there is a simple, shallow equation that is a very good description of the fit, NUANCE usually finds it very quickly. Indeed, it offers an embarrassment of riches, by representing the solution in various different ways. The four simple examples in this section illustrate this for several sets of perfect and noisy data. All were run on a 867 Mhz Power PC G4 Macintosh with 640 megabytes under System 10.3, while a variety of other processes were running.

Example 4a: $x = sin(sqr(a))$

The first example used the equation:

$$x = sin(sqr(a))$$

We built a data set of 500 pairs (with random values for 'a' between 0 and 1) that fit this equation and presented the problem to NUANCE, with a population size of 10,000 and AWF to assess fitness. In the first generation (i.e. by random search, before any evolution has occurred) NUANCE found the perfectly correlated solution:

*((sin a) * (cos a))* [= *0.5 * (sin(sqr(a))*]

It also reported the trivially equivalent solutions *(sin (a * a))* and *sin(sqr(a))* in that generation. Since a perfect solution had been attained, the experiment was halted at this point, having taken 40 seconds.

We repeated the same experiment with noise added to *x*, so that the correlation between *sin(sqr(a))* and the value of *x* was 0.929. NUANCE returned all three of the solutions above in the first generation, after 29 seconds. We let the program continue for 10 runs of 100 generations. The best fit after that time was:

*(sin ((sin(a) / cos (6)) * (cos a)))*

Disregarding the irrelevant constant *cos(6)*, this may be expressed as:

sin((sin(a)/0.96) cos(a))

This correlated with the dependent variable at r = 0.932, adding a minuscule correction factor to the target function *sin(sqr(a))*, with which it correlated at 0.9999.

Example 4b: *x = a * sqr(b)*

The second example used two predictors a and b, related by the representationally-shallow (if conceptually deep) equation made famous by Einstein:

*x = a * sqr(b)*

We used the same NUANCE parameters as above.

When the data were perfectly clean, the following exact solution was returned in Generation 1, after 8 seconds, before evolution had begun:

((sqr b) / (/ 1 a)) [= *1/a / sqr(b),* or *a * sqr(b)*]

We added noise to the equation, so that x correlated with the target function at 0.818. After 9 seconds of computing, all five of the top five equations returned from

Generation 1 were perfectly correlated with this solution: one stating it precisely, and the other four merely including an additional irrelevant constant. At the end of 100 generations, the best solution offered was a more complex equation that correlated with the target variable only slightly better, at 0.821. That equation correlated with the 'canonical' solution at 0.999, suggesting that it was indeed picking up on the same variation.

Example 4c: $x = 1.3a - 144.2b + 163.5$

As its name suggests, NUANCE was developed mainly to aid in finding the shape of non-linear relations between variables. Excellent tools already exist for characterizing the parameters of relationships that are definitely known to be linear. Such tools are better suited for characterizing those linear relationships than NUANCE, since they are designed specifically to return the solution in its canonical linear form, which NUANCE is not. Nevertheless, NUANCE is able to recognize and characterize linear relationships as easily as any other relationships. Moreover, in doing so it may open our eyes to the new ways of construing those linear relationships.

The final two examples we consider here are randomly chosen linear equations with two variables. We continued to use the same NUANCE parameters as in the above two examples.

The first equation was:

$$x = 1.3a - 144.2b + 163.5.$$

We began again with non-noisy data, with both a and b given random values between 1 and 100. The best equation to appear from the first generation, which took 14 seconds to run, was:

(- (/ 1.25 a) (- 5.46 b)), or, in prefix notation, *1.25/a – 5.46b*.

Within the range of *a* (1 to 100), the value of *1.25/a* is likely to be very small on average, so this equation already suggests what we know to be true from the target equation: the role of *a* must be small or nonexistent.

By Generation 22, after a few minutes of computation, the best equation correlated with the entire set at 0.999999, which is probably good enough for anything except a test with a known perfect answer.

The following perfect solution evolved at Generation 65:

(+ (sin 8.77) (- (+ (b (sqr 10.23)) (- (* b 6.08) (cbrt 1.09))) a))*.

Ignoring the constant values that have no effect on the goodness of a correlational fit, this is easily simplified to:

a – 110.92b

This is the original equation divided by 1.3

We added noise so that the correlation between the 'canonical' output of the equation and the dependent measure given to NUANCE was 0.711. After 100 generations, the best equation correlated with the entire test set slightly better, at |r| = 0.724. In order to assess whether this solution was the 'correct' one, we correlated the evolved solution with the 500 real values of the predictor equation. The correlation was very high (|r| = 0.97), indicating the evolved solution was indeed functionally similar to the target equation.

The best evolved equation was complex. Rather than simplifying it as above, we will demonstrate an alternative way of approaching the simplification process when using NUANCE. This method uses the evolved solution and NUANCE's normal output to find

a better model. Having the best solution in hand sets a bound that enables us to easily search for a simpler model of our phenomenon that is more easily comprehensible and almost as good. In NUANCE's log file we found (among several other candidates worth considering) the very simple function $b * ln(b)$, which NUANCE reported to be correlated with the predictor variable almost as well as the final best evolved equation: $|r| = 0.70$. The difference in the variance accounted for by these two models- the best evolved equation versus the extremely simple equation, $x = b * ln(b)$- is less than 1%.

If this were a real problem, we could surmise from this finding that predictor *a* is not a relevant factor in predicting the dependent variable. In this toy example, since we have the true values of the phenomenon being modeled, we are able to test whether $b * ln(b)$ is a good model. The values of $b * ln(b)$ correlate with the output of the equation to be predicted at $|r| = 0.997$. Since we have a mathematically precise definition of the phenomenon under study, we are also able to establish whether or not this is a good general solution, by again conducting Monte Carlo simulations as we did above. Over 20 random sets of 500 pairs, the average [SD] absolute correlation between $b * ln(b)$ and the output of the original linear equation defined over two variables is 0.997 [0.0001]. Knowing the equation as we do, it is not surprising that this is such a good model. In fact *b* alone (which is very highly correlated with $b * ln(b)$) is an even better model. NUANCE is unlikely to find single-node solutions, so it missed the simpler and better model *b*. However, it was nevertheless able to quickly guide us to see that it is only variable *b* that really matters in this example.

Example 4d: $x = 5a + 8b + 6$

In the final example, we look at a less skewed linear relation with integral coefficients:

$$x = 5a + 8b + 6$$

We again used random values between 1 and 100 for *a* and *b*. The best equation from the first generation was

(+ (/ a 10) (sqrt b)) or, in infix notation, *a/10 + sqrt(b)*

This correlated with the 500 target values at 0.982.

A perfect solution was found at Generation 28 of the first run and expressed succinctly as:

(+ (+ (b 4) a) (+ a (/ a 2)))*

We simplified by inspection to:

2.5a + 4b

This is the original equation divided by 2.

We added noise so that the actual dependent values correlated at 0.672 with the target values. The best function over 100 runs correlated with those target values only slightly better, at 0.677. The correlation between the output of this evolved equation and the values given by the original equation was 0.988, suggesting, as the previous noisy examples did, that the best solution captures the same variance as the original target equation.

We simplified the best function by analogy, as above, by finding a simpler function in NUANCE's log files that did almost as well as the best equation. The following equation correlated with the target values at 0.676:

(+ 2.44 (+ (+ (+ b a) (+ a (+ (+ b b) b))) (- (+ b a) (tan a))))

This equation was selected because it can be easily simplified by inspection (ignoring the constant) to:

$$3a + 5b - tan(a).$$

This equation is a good model of this data set, since it produces output that correlates with the actual values of the original target equation in the input at 0.991. However, it is somewhat specialized to that particular file: a Monte Carlo simulation over 20 sets of 500 sets of two random numbers related by the target equation shows that its average [SD] correlation with the original target equation is 0.70 [0.09]. However, since the file itself contained values that correlated at only 0.68 with that target equation, we may not reasonably expect an exact reconstruction of that target equation.

Together these four final examples illustrate two main points. The first is that NUANCE is always able to find the exact solution to fit data that have a known simple characterization. This gives us confidence in the solutions it offers to real problems, where the characterization of the problem is not known in advance. The second point is that the evolutionary history reported by NUANCE can be a fruitful source of hypotheses, potential models, and general 'mathematical commentary' on the phenomenon under study. By looking for equations that are both attractive to the human mind and as successful at explaining variance as any known model, we may come up with new ways of conceptualizing our problems and the data we have collected on them.

Conclusion

The purpose of this paper is to introduce a computer program called NUANCE, the Naturalistic University of Alberta Nonlinear Correlation Explorer. We have contributed this program to the archival web site of the Psychonomic Society

(http://psychonomic.org/archive), from where it may be freely downloaded. As its name suggests, NUANCE is a tool for exploring arbitrarily complex relationships between a set of predictor variables, and a dependent measure to be predicted. Written in Java in order to be platform-independent, NUANCE explores the space of possible relationships using genetic programming, which computes using natural selection. Our program makes use of a new age-weighted fitness function that is superior to existing published fitness functions. We have briefly introduced the concepts underlying genetic programming, explained the parameters and use of NUANCE itself, and presented some examples of its possible uses.

We believe that NUANCE is a useful tool for anyone faced with understanding complex relationships in a data set with one or more predictors and a single dependent variable.

References

Andrews, S. (1997). The effect of orthographic similarity on lexical retrieval: Resolving neighborhood conflicts. *Psychonomic Bulletin & Review, 4*, 439-461.

Fraumeni, J.F. (1968). Cigarette Smoking and Cancers of the Urinary Tract: Geographic Variations in the United States. *Journal of the National Cancer Institute, 41*, 1205-1211.

Holland, J. (1992). *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence*. Cambridge, MA: MIT Press.

Koza, J. (1992). *Genetic programming: On the programming of computers by means of natural selection*. Cambridge, MA: MIT Press.

Kushchu, I. (2002). Genetic programming and evolutionary generalization. *IEEE Transactions on evolutionary computation, 6*, 431-442.

Luke, S. & Spector, L. (1998). A Revised Comparison of Crossover and Mutation in Genetic Programming. In J. Koza, W. Banzhaf, K. Chellapilla, K. Deb, M. Dorigo, D. B. Fogel, M. H. Garzon, D. E. Goldberg, H. Iba, & R. Riolo (Ed.), *Genetic Programming 1998: Proceedings of the Third Annual Genetic Programming Conference* (pp. 51-84). San Fransisco, CA: Morgan Kaufmann.

Langdon, W. B. (2000). Quadratic Bloat in Genetic Programming. In D. Whitley, D. Goldberg, E. Cantu-Paz, L. Spector, I. Parmee & H. Beyer (Ed.), *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2000)* (pp. 451-458). San Fransisco, CA: Morgan Kaufmann.

Langdon, W. B., & Poli, R. (1997). *Fitness causes bloat* (Tech. Rep. No. CSRP-97-09).

Birmingham, UK: University of Birmingham, School of Computer Science.

Moore, F. W., & Garcia, O. N. (1997). New methodology for reducing brittleness in genetic programming. *Proceedings of the National Aerospace and Electronics Conference (NAECON-97)*, E. Pohl (Ed.), Dayton, OH, July 1997.

Streeter, M. J. (2003). The Root Causes of Code Growth in Genetic Programming. In C. Ryan, T. Soule, M. Keijzer, E. Tsang, R. Poli, & E. Costa (Ed.), *Genetic Programming, Proceedings of EuroGP'2003* (pp. 449-458). Berlin, Germany: Springer-Verlag.

Stone, M. (1974). Cross-validation choice and assessment of statistical predictions. *Journal of the Royal Statistical Society*, B-36, 111–147.

Soule, T., & Foster, J. A. (1998). Removal Bias: a New Cause of Code Growth in Tree Based Evolutionary Programming. In D. Fogel (Ed.), *1998 IEEE International Conference on Evolutionary Computation* (pp. 781-786). Los Alamitos, CA: IEEE Computer Society Press.

Soule, T., & Foster, J. A. (1999). Effects of Code Growth and Parsimony Pressure on Populations in Genetic Programming. *Evolutionary Computation*, 6:4, 293-309.

Westbury, C., Buchanan, L., Sanderson, S., Rhemtulla, M., & Phillips, L. (2003). Using genetic programming to discover non-linear variable interactions. *Behavior Research Methods, Instruments, and Computers, 35:2*, 202-216.

Wolfram Research Inc. (2003). *Mathematica*, Version 5.0, Champaign, IL.

Figure 1. A NUANCE fitness curve, using *Median-pass* selection, illustrating the danger of relying overly on the shape of the fitness curve as a measure of whether or not one has run sufficient generations. This problem used a data set from DASL, at http://lib.stat.cmu.edu/DASL/Datafiles/SMSA.html, to estimate average January temperature in 60 American cities from average rainfall in those cities. The fitness curve is almost perfectly flat with a correlation of approximately 0.11 for 47 generations, which might lead one to conclude that no better solution was possible. However, a breakthrough at generation 48 lead to substantial and thereafter increasing improvement on this early estimate. This illustrates why it is a good idea to run as many generations as possible.

Figure 2. Average function length & fitness (* 1000) by generation, with (P1) and without (PO) the minimal parsimony pressure of 1%. This figure shows that even a small parsimony pressure (penalty for long functions) can have a marked effect on average fitness length, while having little effect on fitness. In order to show both fitness and length on the same graph, we have multiplied fitness values (correlations with the predictor) by 1000. These data are from the ON x FREQ problem discussed in Example 1.

Figure 3. Probability of containing best predictor, by generation and age-weight function using AWF (33-generation bins). The dependent variable in this example is lexical decision RTs using 10 predictors.

Figure 4. Correlation of the best evolved predictor equation with a dependent variable, in 20-generation bins, by age-weighting scheme. The dependent variable in this example is lexical decision RTs using 10 predictors.

Figure 5. Two typical evolutionary runs of the best evolved predictor, by generation, for the two selection schemes offered in NUANCE. *Median-pass* selection with age-weighting is always less erratic, but *Greedy Over-selection* (which cannot benefit from age-weighting) almost always performs slightly better at predicting variance. The dependent variable in this example is lexical decision RTs using two predictors (see Example 1 in the text).

Figure 6. Best and average of 10 NUANCE runs for Example 2: Predicting leukemia deaths from cigarette consumption.

Figure 7. Predicting leukemia deaths from cigarette consumption. In this graph the NUANCE-evolved estimate and the raw leukemia data are both graphed, in standardized units, against cigarette consumption.

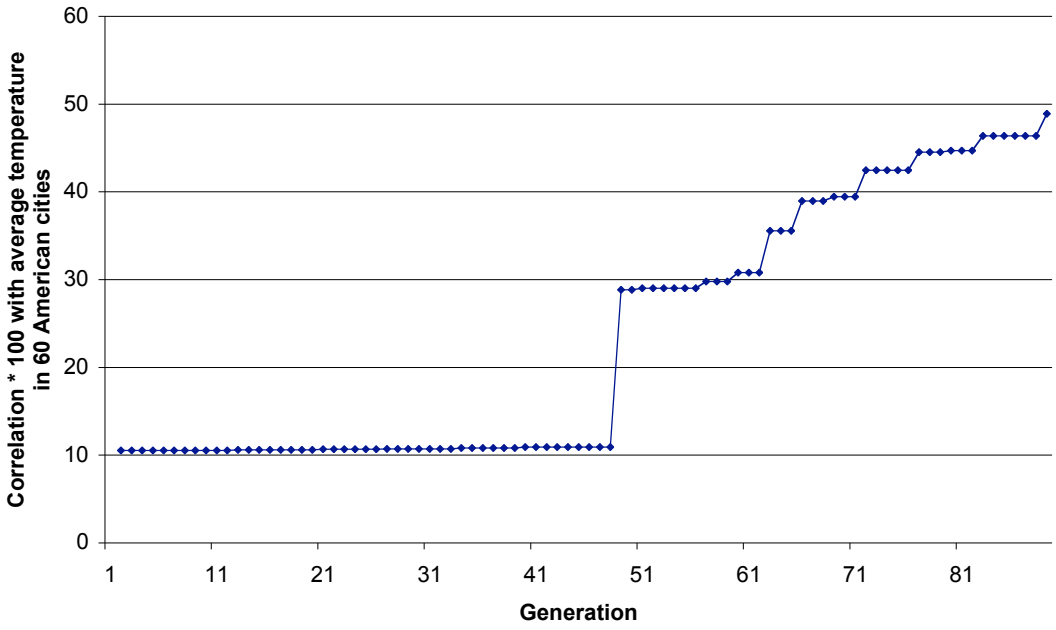
Correlation * 100 with average temperature in 60 American cities
60
50
40
30
20
10
0
1
11
21
31
41
51
61
71
81
Generation

Fig. 1

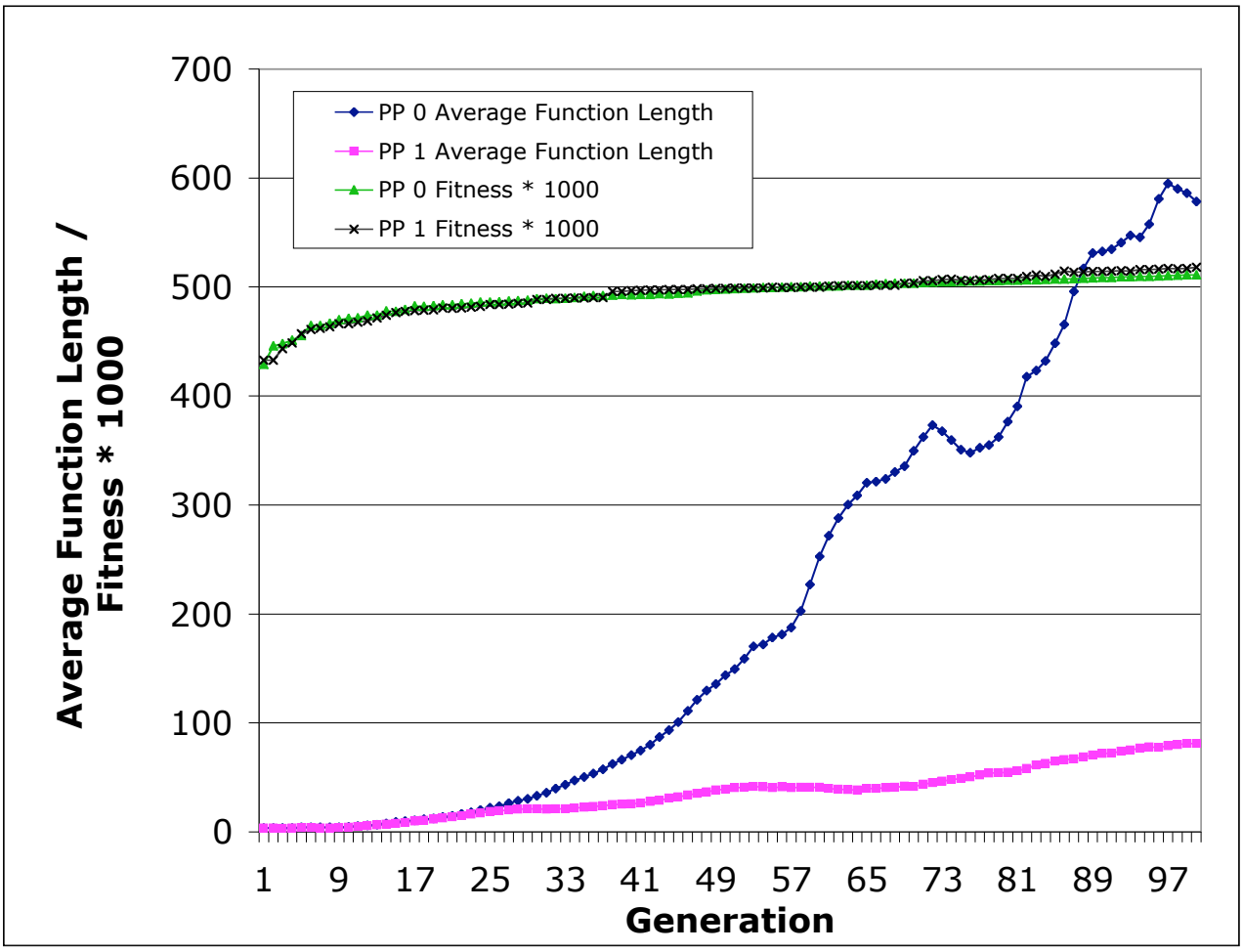
700
600
500
400
300
200
100
0
1
9
17
25
33
41
49
57
65
73
81
89
97
Generation
Average Function Length / Fitness * 1000
PP 0 Average Function Length
PP 1 Average Function Length
PP 0 Fitness * 1000
PP 1 Fitness * 1000

Fig. 2

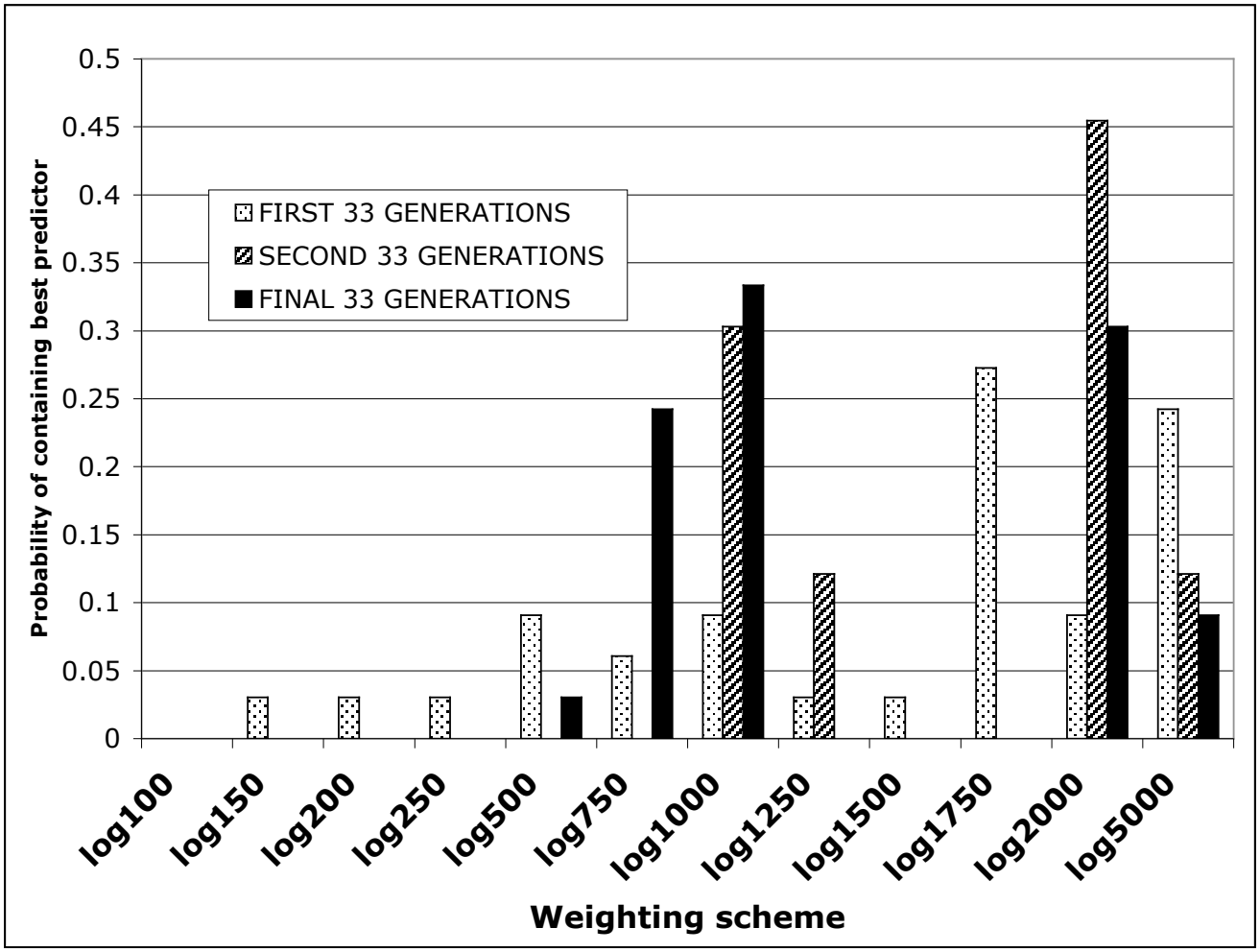
0.5
0.45
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
Probability of containing best predictor
FIRST 33 GENERATIONS
SECOND 33 GENERATIONS
FINAL 33 GENERATIONS
log100
log150
log200
log250
log500
log750
log1000
log1250
log1500
log1750
log2000
log5000
Weighting scheme

Fig. 3

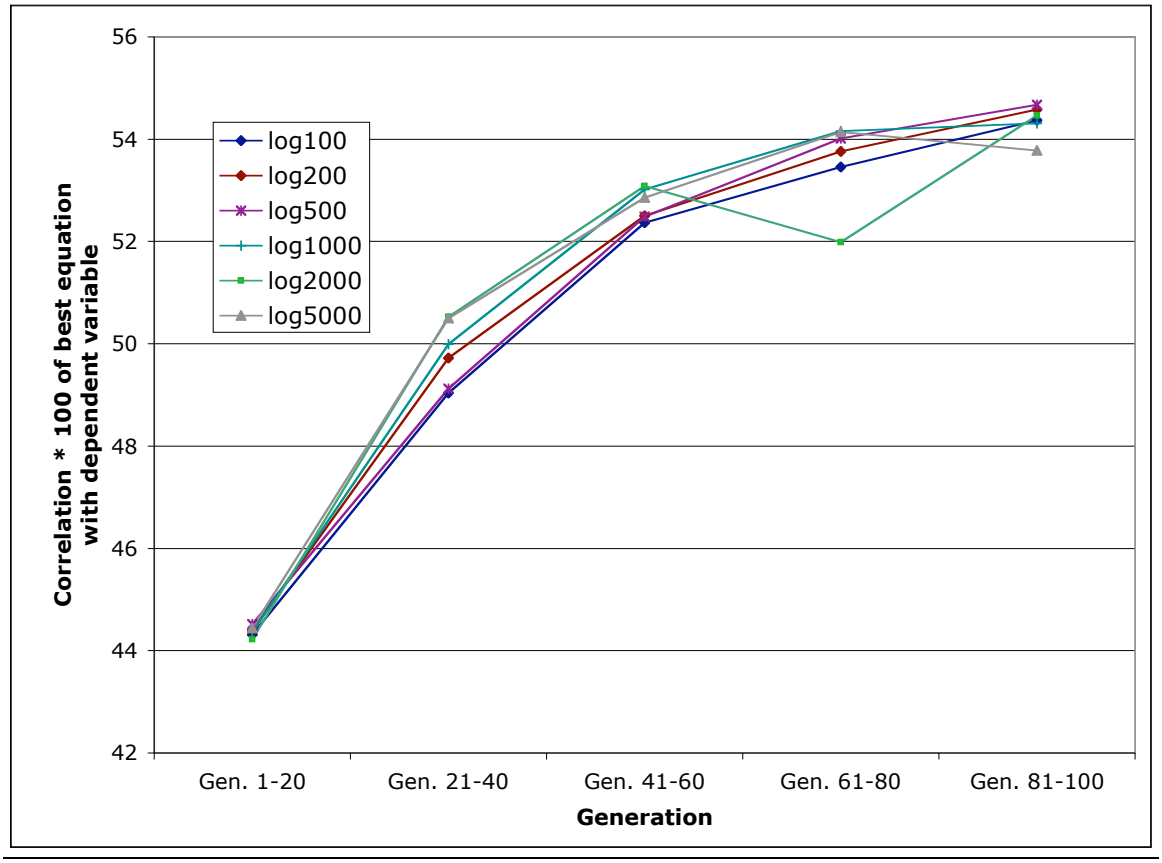
56
54
52
50
48
46
44
42
log100
log200
log500
log1000
log2000
log5000
Correlation * 100 of best equation with dependent variable
Gen. 1-20
Gen. 21-40
Gen. 41-60
Gen. 61-80
Gen. 81-100
Generation

Fig. 4

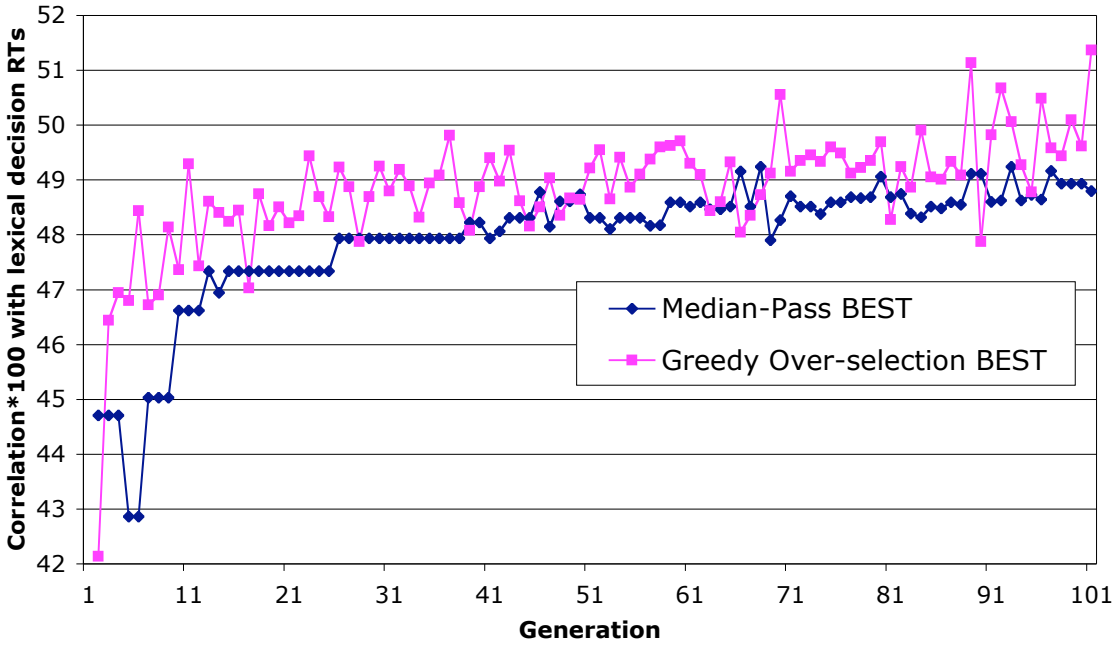
Correlation*100 with lexical decision RTs
52
51
50
49
48
47
46
45
44
43
42
1
11
21
31
41
51
61
71
81
91
101
Generation
Median-Pass BEST
Greedy Over-selection BEST

Fig. 5

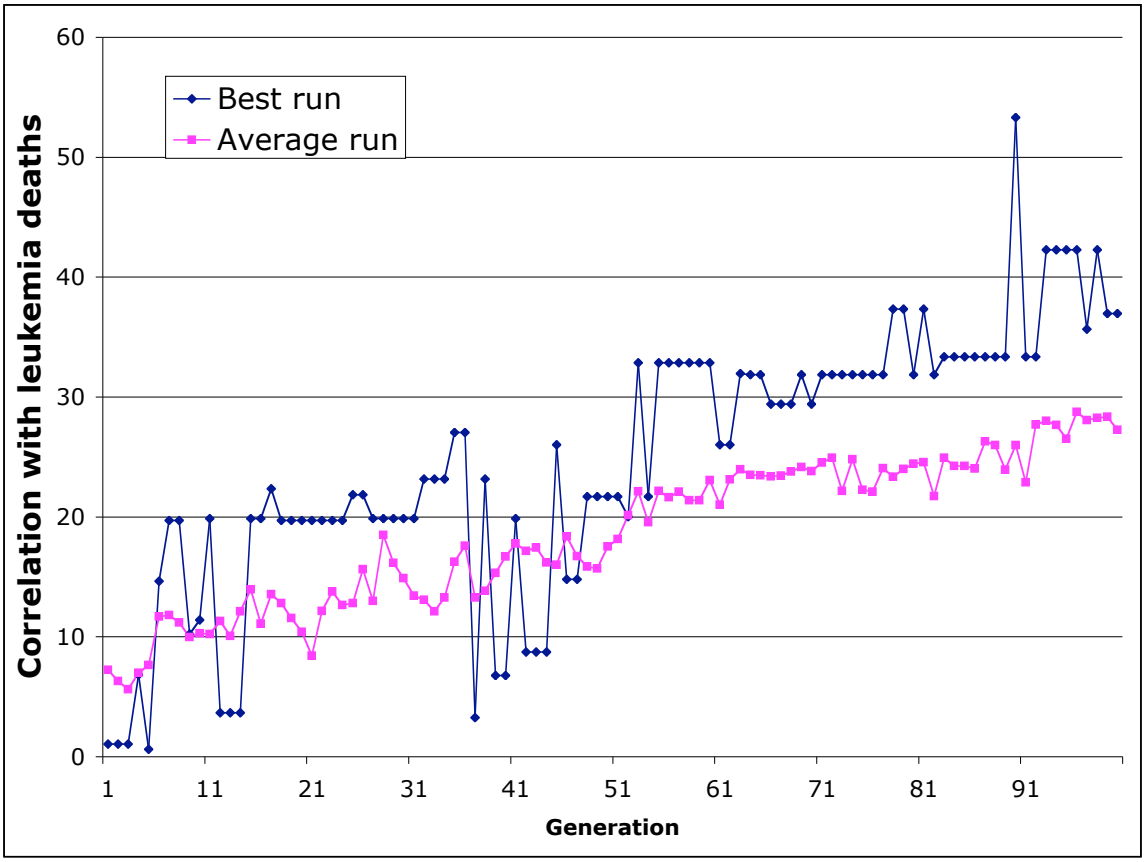
Correlation with leukemia deaths
Best run
Average run
60
50
40
30
20
10
0
1
11
21
31
41
51
61
71
81
91
Generation

Fig. 6

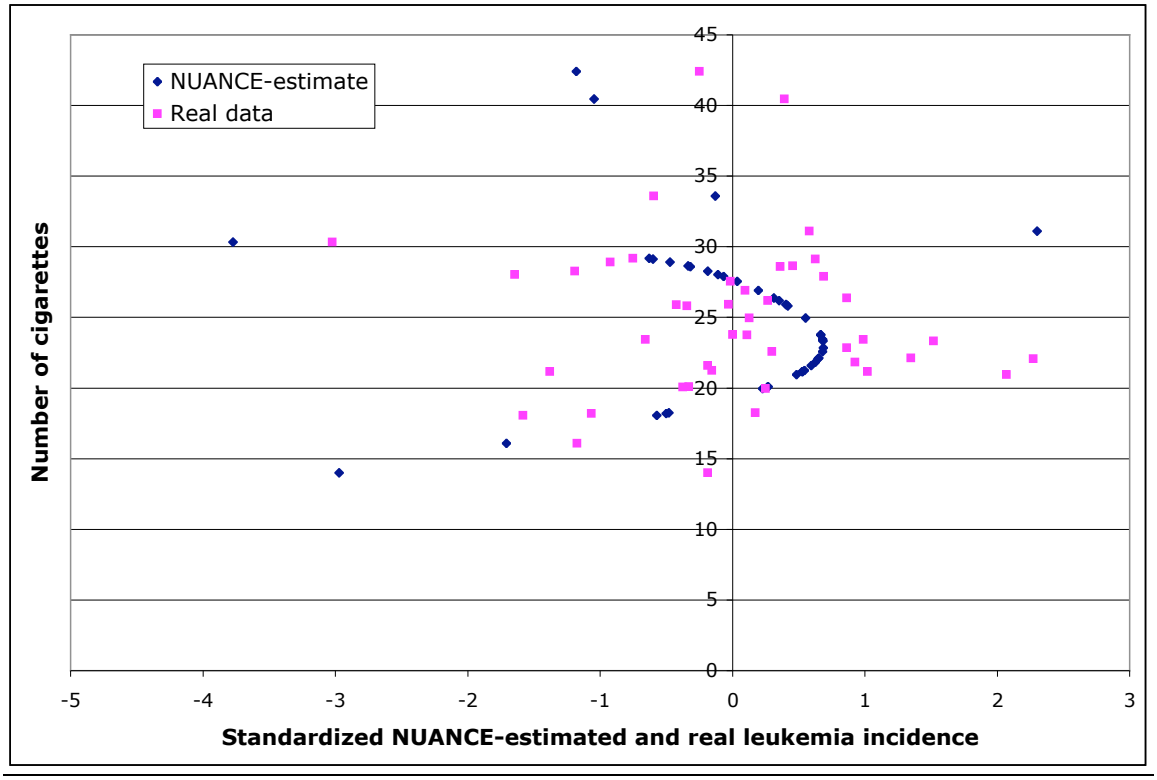
NUANCE-estimate
Real data
Number of cigarettes
45
40
35
30
25
20
15
10
5
0
-5
-4
-3
-2
-1
0
1
2
3
Standardized NUANCE-estimated and real leukemia incidence

Fig. 7

Greedy Over-selection best solution (|r| = 0.52):
(/ (abs 2.0) (+ (+ (/ (abs (+ (- OFREQ (- (sqrt (sqrt (sqrt (abs 2.0)))) (- (+ (+ OFREQ (abs (+ (sqrt (sqrt OFREQ)) 9.0))) (+ (+ (sqrt ON) OFREQ) ON)) (+ 0.0 (+ 9.0 OFREQ))))) (sqrt OFREQ))) (+ (- OFREQ (- (+ (/ OFREQ (+ (+ (- OFREQ OFREQ) 9.0) 7.0)) (+ (+ (abs OFREQ) OFREQ) (sqrt 7.0))) (+ 0.0 (+ 9.0 9.0)))) 9.0)) (+ ON OFREQ)) (+ 9.0 OFREQ)))

$$2\,\mathrm{OFREQ} + \mathrm{ON} + \frac{\mathrm{Abs}\left[-10.09 + \sqrt{9 + \sqrt{\mathrm{OFREQ}}} + \sqrt{\mathrm{OFREQ}} + 2\,\mathrm{OFREQ} + \sqrt{\mathrm{ON}} + \mathrm{ON}\right]}{24.35 - 1.06\,\mathrm{OFREQ}}$$

Median Pass selection best solution (|r| = 0.48):

(/ 1 (+ ON (+ OFREQ (+ 10.0 OFREQ))))

$$\frac{1}{10 + 2\,\mathrm{OFREQ} + \mathrm{ON}}$$

Table 1: The two best solutions, in prefix and standard notation, as evolved for a problem (considered in Example 1) relating ON and word frequency to lexical decision RTs for words. Both estimates correlate highly with each other and with the best estimate reported in Westbury et al. (2003), which was evolved using a different GP system. We have reproduced the raw NUANCE output here for illustrative purposes, as well as simplifications derived from Mathematica. The first equation was also simplified by hand. See text for further discussion.