ELSEVIER

Information and Software Technology 43 (2001) 863–873

INFORMATION AND SOFTWARE TECHNOLOGY

www.elsevier.com/locate/infsof

# Can genetic programming improve software effort estimation? A comparative evaluation

Colin J. Burgess[a,*], Martin Lefley[b]

[a]Department of Computer Science, University of Bristol, Merchant Venturers Building, Woodland Road, Bristol BS8 1UB, UK
[b]School of Design Engineering and Computing, University of Bournemouth, Talbot Campus, Poole BH12 5BB, UK

**Abstract**

Accurate software effort estimation is an important part of the software process. Originally, estimation was performed using only human expertise, but more recently, attention has turned to a variety of machine learning (ML) methods. This paper attempts to evaluate critically the potential of genetic programming (GP) in software effort estimation when compared with previously published approaches, in terms of accuracy and ease of use. The comparison is based on the well-known Desharnais data set of 81 software projects derived from a Canadian software house in the late 1980s. The input variables are restricted to those available from the specification stage and significant effort is put into the GP and all of the other solution strategies to offer a realistic and fair comparison. There is evidence that GP can offer significant improvements in accuracy but this depends on the measure and interpretation of accuracy used. GP has the potential to be a valid additional tool for software effort estimation but set up and running effort is high and interpretation difficult, as it is for any complex meta-heuristic technique. 

*Keywords*: Case-based reasoning; Genetic programming; Machine learning; Neural networks; Software effort estimation

## 1. Introduction

Most organisations must decide how to allocate valuable resources based on predictions of the unknown future. The example studied here is software effort estimation, where volume and costs are not directly proportionally related [12]. Any improvement in the accuracy of prediction of development effort can significantly reduce the costs from inaccurate estimation, misleading tendering bids and disabling the monitoring progress. Accurate modelling can also assist in scheduling resources and evaluating risk factors. In this paper, we evaluate the potential benefits from applying the tool of genetic programming (GP) to improve the accuracy of software effort estimation. In order to do this, we compare it with other methods of prediction, in terms of accuracy and other qualitative factors deemed important to potential users of such a tool. All the parameters used for the estimation process are restricted to those available at the specification stage. For those readers who are already familiar with the concepts of GP, Sections 6.1 and 6.2 can be skipped.

## 2. Background to the problem

Learning allows humans to solve hugely complex problems at speeds which outperform even the fastest computers of the present day [36]. Machine learning (ML) techniques have been used successfully in solving many difficult problems, such as speech recognition from text [37], adaptive control [34,16], integrated circuit synthesis [31] and mark-up estimation in the construction industry [16]. Due to their minimal assumptions and broad and deep coverage of solution space, ML approaches deserve exploration to evaluate the contribution they have to the prediction of software effort.

One of the first methods used to estimate software effort automatically was COCOMO [5], where effort is expressed as a function of anticipated size. The general form of the model tends to be:

$$E = aS^b \tag{1}$$

where $E$ is the effort required, $S$ is the anticipated size and $a$ and $b$ are domain specific constants.

Others have developed local models, using statistical techniques such as stepwise regression e.g. Kok et al. [23]. Linear approaches to the problem cover only a small part of the possible solution space, potentially missing the complex set of relationships evident in many complex

* Corresponding author.
*E-mail addresses:* colin.burgess@bristol.ac.uk (C.J. Burgess), mlefley@bournemouth.ac.uk (M. Lefley).

0950-5849/01/$ - see front matter 
PII: S0950-5849(01)00192-6

Table 1
Summary of the variables available in the Desharnais data set

| Feature | Ref | Explanation |
|---|---|---|
| Project name | | Numeric identifier |
| ExpEquip | *X*1 | Team experience in years |
| ExpProjMan | *X*2 | Project manager's experience in years |
| Transactions | *X*3 | Number of transactions processed |
| Entities | *X*4 | Number of entities |
| RawFPs | *X*5 | Unadjusted function points |
| AdjFPs | *X*6 | Adjusted function points |
| DevEnv | *X*7 | Development environment |
| YearFin | *X*8 | Year of completion |
| Envergure | *X*9 | Complex measure derived from other factors — defining the environment |
| Dependent variable | | |
| Effort | *X*10 | Measured in person-hours |

domains such as software development environments. For example, Kemerer [21] and Conte et al. [6] frequently found errors considerably in excess of 100% even after model calibration. A variety of ML methods has been used to predict software development effort. Artificial neural networks (ANNs) [20,42], case based reasoning (CBR) [15], rule induction (RI) [19,22,40] offer good examples. Hybrids are also possible [22], e.g. Shukla [39] reports better results from using an evolving ANN compared to a standard back propagation ANN. Dolado [9,10] and Fernandez and Dolado [13] analyse many aspects of the software effort estimation problem.

Recent research by Dolado [11] shows promising results for a GP based estimation system on a single input variable. This research extends this idea into richer models requiring larger populations and much longer learning lifetimes. This paper investigates the potential for the use of GP methods to build software cost prediction systems and compares preliminary results against other previously researched approaches. Emphasis is placed on assisting the software engineering manager to realistically consider the use of GP as an estimator of effort. For example, parameters that cannot be measured at the outset of a project, such as lines of code, are not used. Evaluation does not merely focus on prediction accuracy but also on other important factors necessary for good design of an estimation methodology.

Table 2
The project number and effort of the 18 projects selected for measuring prediction capability

| Project number | Effort (person-hours) |
|---|---|
| 2 | 5635 |
| 8 | 3913 |
| 12 | 9051 |
| 15 | 4977 |
| 19 | 4494 |
| 20 | 840 |
| 22 | 651 |
| 35 | 5922 |
| 38 | 2352 |
| 42 | 14987 |
| 46 | 5817 |
| 50 | 8232 |
| 56 | 2926 |
| 61 | 2520 |
| 63 | 2275 |
| 72 | 13860 |
| 76 | 5880 |
| 80 | 5180 |

We attempt to assess GP against alternative techniques with minimum bias to give a fair comparison. We have tried to avoid putting a lot of tuning effort into one paradigm with the misguided aim to prove that it is better than other methods that have been comparatively sparsely explored. Some researchers make significant benefits by removing alleged outliers from the data sets, but we have avoided this since we want to use as much of the data as possible. Also, the determination of outliers can incorporate bias, complicate comparison and could be based on any number of heuristics. Thus, this is not a simple task for an estimator with only, in general, a limited data set available. Another danger is to generate many diverse solutions so that a best model, based on query performance can be selected that will always fit very closely to the evaluation data. Ultimately, we test the hypothesis that a GP based solution can learn the nuance of a data set and make its own decisions for issues such as variable weighting and pre-processing, dealing with outliers and the selection of feature subsets.

## 3. Data set used for comparisons and Weibull distribution modelling

In order to explore and compare the potential of the three ML techniques for building effort prediction models we selected an existing project effort data set. The data set used comprises 81 software projects derived from a Canadian software house in the late 1980s by Jean-Marc Desharnais [8]. Despite the fact that this data is now over 10 years old, it is one of the larger, publicly available, data sets and can be used to assess the comparative performance of new approaches.

The data set comprised 10 features, one dependent and nine independent, as summarised by Table 1. A second dependent feature, namely length of code, is ignored, as this is far less important to predict than total effort. Four of the 81 projects contained missing values so these were replaced by random samples from the other projects.

In order to compare many different prediction paradigms, the data was divided into two sets which are called a training set and a query set. The query set chosen was 18 projects, which were randomly selected from the 81 projects, i.e. approximately 22% of the total (Table 2). The remaining 63 projects were used as the learning set. In order to eliminate all possible distortion and to simulate more realistically the practical situation, the query set was not used in

Table 3
Training data output statistics

| | Effort |
|---|---|
| Mean | 4908.556 |
| S.D. | 4563.736 |
| Minimum | 546 |
| Maximum | 23940 |
| Weibull Alpha | 1.41 |
| Weibull Beta | 4830 |

any of the further analysis or tuning, apart from measuring the accuracy of the final prediction of effort.

The training output values have been found by the authors, to be fitted well by Weibull distributions, with a highly significant Chi Square and smaller error than other candidate distributions. The Weibull parameters were found by piecewise approximation and provided in Table 3.

One useful measure of how much the input or independent variables influence the output or dependent variables, is to calculate the correlation coefficients between input and output. Since variables may have functional relationships a number of functions of input variables (such as logarithmic and exponential) were used to search for the highest correlations. None of the functions was found to have increased the raw correlations by enough to justify their continued use.

An illustration of the output data's distribution is given in Fig. 1, along with the fitted distributions.

## 4. How to evaluate the techniques?

Software effort prediction is typified by comparatively small data sets with prediction errors having significant effect on costs. An important question that needs to be asked of any estimation method is 'How accurate are the predictions?'. A number of summary statistics were used, none of which can be seen as significantly better than any of the others. Ultimately the decision should be made by the user, based on estimates of the costs of under or over estimating project effort. All are based on the calculated error, i.e. the difference between the predicted and observed output values for the projects.

The methods used in this paper were:

(a) Correlation coefficient.
(b) Adjusted mean square error — AMSE.
(c) Predictions within 25% — Pred(25).
(d) Percentage of predictions within 25% — Pred(25)% Distributions.
(e) Mean magnitude of relative error — MMRE.
(f) Balanced mean magnitude of relative error — BMMRE.

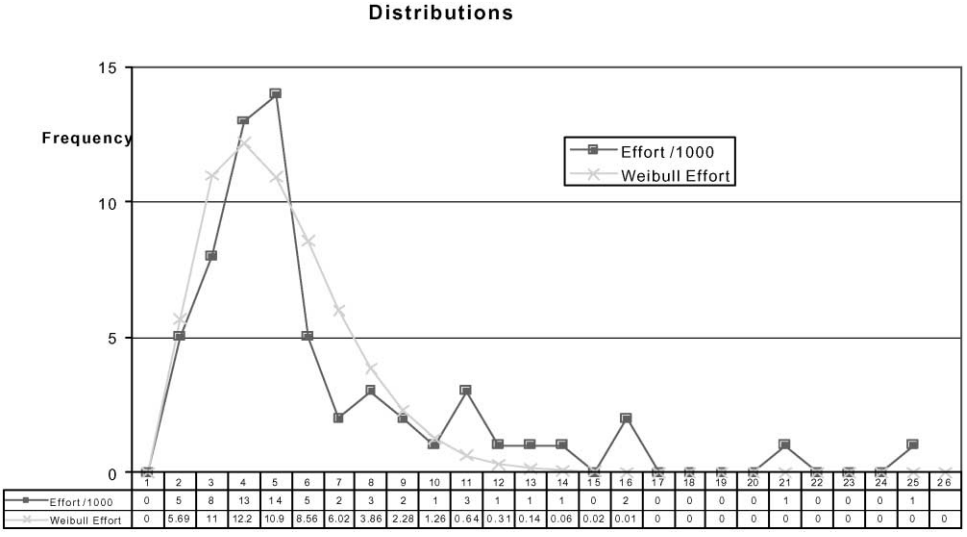
Most commonly, accuracy is defined in terms of mean magnitude of relative error (MMRE) [10], which is the mean of absolute percentage errors:

$$\mathrm{MMRE} = \sum_{i=1}^{i=n} \left( \frac{|E_i - \hat{E}_i}{E_i} \right) \frac{100}{n} \tag{2}$$

where there are $n$ projects, $E_i$ is the actual effort and $\hat{E}_i$ is the predicted effort. There has been some criticism of this measure, in particular the fact that it is unbalanced and penalises over-estimates more than underestimates. For this reason, Miyazaki et al. [33] proposed a balanced MMRE measure as follows:

$$\mathrm{BBMRE} = \sum_{i=1}^{i=n} \left( \frac{|E_i - \hat{E}_i}{\min(E_i, \hat{E}_i)} \right) \frac{100}{n} \tag{3}$$

This approach has been criticised by Hughes [18], amongst others, as effectively being two distinct measures




| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Effort /1000 | 0 | 5 | 8 | 13 | 14 | 5 | 2 | 3 | 2 | 1 | 3 | 1 | 1 | 1 | 0 | 2 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | |
| Weibull Effort | 0 | 5.69 | 11 | 12.2 | 10.9 | 8.56 | 6.02 | 3.86 | 2.28 | 1.26 | 0.64 | 0.31 | 0.14 | 0.06 | 0.02 | 0.01 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Fig. 1. Distribution of effort along with fitted Weibull distribution.

that should not be combined. Another approach is to use Pred(25)%, which is the percentage of predictions that fall within 25% of the actual value. It is clear that the choice of accuracy measure depends to a large extent upon the objectives of those using the prediction system. For example, MMRE is fairly conservative with a bias against over-estimates, whilst Pred(25) supports those prediction systems that are generally accurate but occasionally wildly inaccurate.

Other workers have used the adjusted $R$ squared or coefficient of determination to indicate the percentage of variation in the dependent variable that can be 'explained' in terms of the independent variables. In this paper, we replace this with the simpler correlation coefficient. The simplest raw measure is the mean square error, however this depends on the mean of the data sets and it is thus difficult to interpret or make comparisons. Instead, we use AMSE, the adjusted mean square error. This is the sum of the squared errors, divided by the product of the means of the predicted and observed outputs.

$$\mathrm{AMSE} = \sum_{i=1}^{i=n} \frac{(E_i - \hat{E}_i)^2}{(\bar{E}_i * \bar{\hat{E}}_i)} \tag{4}$$

### 4.1. Qualitative measures

Whatever measures are being used, it is clear that although accuracy is an important consideration, it is not sufficient to consider the accuracy of prediction systems in isolation. Hence, in assessing the utility of these techniques, we have considered three factors, adapted from Mair et al. [30]: accuracy, explanatory value and ease of configuration. Accuracy has been the primary concern of researchers and clearly it is of considerable importance; a prediction system that fails to meet some minimum threshold of accuracy will not be acceptable. However, we believe that accuracy, by itself, is not a sufficient condition for acceptance. The quality of information provided for the estimator is of great importance. Empirical research has indicated that end-users coupled with prediction systems can outperform either prediction systems or end-users alone [41]. The more explanations given for how a prediction was obtained, the greater the power given to the estimator. If predictions can be explained, estimators may experiment with 'what if' scenarios and meaningful explanations can increase confidence in the prediction.

Apart from accuracy, other evaluation factors for a prediction system, which could be important to non-expert practitioners are:-

(a) Resources required
  (i) Time and memory needed to train.
  (ii) Time and memory needed to query.
(b) Ease of set up.
(c) Transparency of solution or decision.
(d) Generality.
(e) Robustness.
(f) Likelihood of convergence.
(g) Prediction beyond learning data set space.

## 5. Previous related work using machine learning

Any method of transforming data may be used for software estimation, for example, least squares regression. This section considers ML techniques previously recommended for comparative effort estimation viz. CBR and ANNs. These techniques have been selected on the grounds that, there exists adequate previous research to promote their efficacy and because of their significantly differing approaches, they form a good basis for comparisons.

### 5.1. Artificial neural networks

ANNs are learning systems inspired by organic neural systems, which are known to be very successful at learning to solve problems. They comprise a network of simple interconnected units called neurons. The connections between the neurons are weighted and the values of these weights determine the function of the network. Input values are multiplied by the weights, summed, passed through a step function and then on to other neurons and finally to the output neurons. The weights are selected to optimise the output vector produced from a given input vector. Typically, this selection is carried out by adjusting the weights systematically, based on a given data set, a process of training. The most common method of training is by back propagation. Here, the weights begin with small random values and then a proportion of the difference between desired and current output, called the error, is used to adjust weights back through the net, proportionally to their contribution to the error. Thus, the outputs are moved towards the desired values and the net learns the required behaviour.

Recent studies concerned with the use of ANNs to predict software development effort have focused on comparative accuracy with algorithmic models, rather than on the suitability of the approach for building software effort prediction systems. An example is the investigation by Wittig and Finnie [46]. They explore the use of a back propagation neural network on the Desharnais and ASMA (Australian Software Metrics Association) data sets. For the Desharnais data set, they randomly split the projects three times between 10 test and 71 training sets, which is very similar to the procedure we follow in our paper. The results from three validation sets are aggregated and yield a high level of accuracy. The values cited are Desharnais set MMRE = 27% and ASMA set MMRE = 17%, although some outlier values are excluded. Mair et al. [30] show neural networks offer accurate effort prediction, though not as good as Wittig and Finnie, but conclude that they are difficult to configure and interpret.

### 5.2. Case-based reasoning

CBR is based on the psychological concepts of analogical reasoning, dynamic memory and the role of previous situations in learning and problem solving [36]. Cases are abstractions of events, solved or unsolved problems. New problems are solved by a weighted concatenation of the solutions from the set of similar cases. Aarmodt and Plaza [1] describe CBR as being cyclic and composed of four stages:

1. *retrieval* of similar cases.
2. *reuse* of the retrieved cases to find a solution to the problem.
3. *revision* of the proposed solution if necessary.
4. *retention* of the solution to form a new case.

Consequently, issues concerning case characterisation [35], similarity [2,45] and solution revision [28] must be addressed prior to CBR system deployment.

Software project estimation has been tackled by a number of researchers using CBR. Three doculented examples are Estora [43], finding analogies for cost estimation (FACE) [4] and ANGEL [38]. A summary of each of these works follows.

Estora uses the estimator's protocols, and infers rules from these. An analogy searching approach is used to produce estimates which the developers claim were comparable, in terms of *R*-squared values, to the expert's and superior to those obtained using function points, regression based techniques, or COCOMO.

FACE was developed by Bisio and Malabocchia who assessed it using the COCOMO data set. It allocates a normalised similarity score $\theta$, with values between 0 and 100 for each candidate analogy from the case repository. A user threshold, typically $\theta = 70$, is used to decide which cases are used to form the estimate. If no cases score above the threshold, then reliable estimation is not deemed possible. The research shows that FACE performs very favourably against algorithmic techniques such as regression.

Shepperd and Schofield report on their tool ANGEL, an estimation tool based upon analogical reasoning. Here projects, or cases are represented in a Euclidean hyperspace where a modified nearest neighbour algorithm identifies the best analogies. They report results, derived from a number of data sets, of superior performance to LSR models.

Other approaches include that of Debuse and Raywards-Smith [7] who apply simulated annealing algorithms to the problem of feature subset selection which could then be used with other tools. There are also those who consider supplementary tools to enhance another method, for example an evolutionary tool. Though such approaches could be used in conjunction with the tools here, assessing the complete tools available is the goal of this work. Whatever overall methodology is used, the aim is to find the most accurate predictors, within the measurement constraints already outlined in Section 4.

## 6. Background to genetic programming

### 6.1. Introduction to genetic algorithms

Genetic algorithms (GA) were developed as an alternative technique for tackling general optimisation problems with large search spaces. They have the advantage that they do not need any prior knowledge, expertise or logic related to the particular problem being solved. Occasionally, they can produce the optimum solution, but for most problems with a large search space, a good approximation to the optimum is a more likely outcome.

The basic ideas used are based on the Darwinian theory of evolution, which in essence says that genetic operations between chromosomes eventually leads to fitter individuals which are more likely to survive. Thus, over a long period of time, the population of the species as a whole improves. However, not all of the operations used in the computer analogy of this process necessarily have a biological equivalent.

In the computer implementation of these ideas a solution, but not necessarily a very good solution, to the problem being solved is represented typically by a fixed length binary string, which is termed a chromosome by analogy with the biological equivalent. It must be possible to measure the fitness of any solution, where a fitter individual is one that is nearer the optimal solution. One example, which is relevant to the current problem, is that a fitter solution minimises the error between the predicted values and the true values.

The basic process of the genetic algorithm is as follows, although a number of variations are possible:-

1. Generate at random a population of solutions, i.e. a family of chromosomes.
2. Create a new population from the previous one by applying genetic operators to the fittest chromosomes, or pairs of fittest chromosomes of the previous population.
3. Repeat step (2), until either the fitness of the best solution has converged or a specified number of generations have been produced.

The best solution in the final generation is taken as the best approximation to the optimum for that problem that can be attained in that run. The whole process is normally run a number of times, using different seeds to the pseudo-random number generator.

The key parameters that have to be determined for any given problem are:

(a) The best way of representing a solution as a fixed length binary string.
(b) The best combination of genetic operators. For GA,

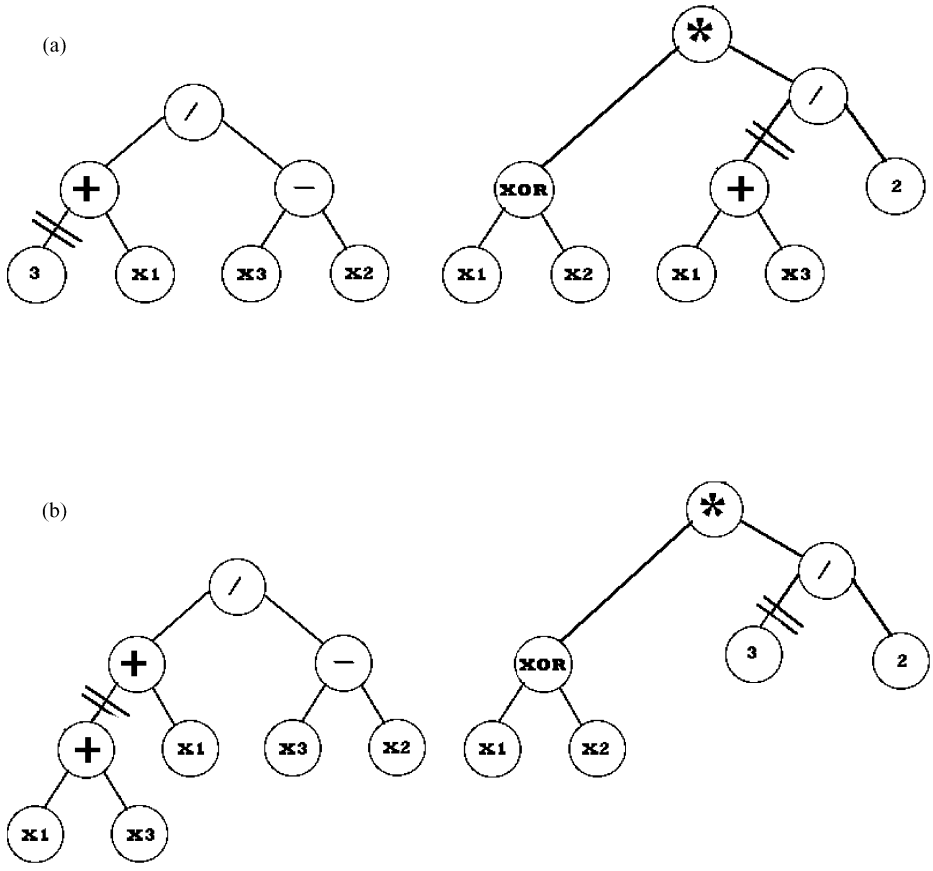



Fig. 2. (a) Illustration of crossover operator before operation. The double line illustrates where the trees are cut, (b) Illustration of crossover operator after operation. The double line illustrates where the sub-trees have been swapped.

reproduction, crossover and mutation are the most common.
(c) Choosing the best fitness function, to measure the fitness of a solution.
(d) Trying to keep enough diversity in the solutions in a population to allow the process to converge to the global optimum but not converge prematurely to a local optimum.

More information related to GA can be found in Goldberg [14] or Mitchell [32].

### 6.2. Introduction to genetic programming

GP is an extension of GA, which removes the restriction that the chromosome representing the individual has to be a fixed length binary string. In general, in GP, the chromosome is some type of program, which is then executed to obtain the required results. One of the simplest forms of program, which is sufficient for this application, is a binary tree containing operators and operands. This means that each solution is an algebraic expression, which can be evaluated.

Koza "offers a large amount of empirical evidence to support the counter-intuitive and surprising conclusion that GP can be used to solve a large number of seemingly different problems from many different fields" [24]. He goes on to state that GP offers solutions in representations of computer programs. These offer the flexibility to:-

(a) Perform operations in a hierarchical way.
(b) Perform alternative computations conditioned on the outcome of intermediate calculations.
(c) Perform iterations and recursions.
(d) Perform computations on variables of different types.
(e) Define intermediate values and sub-programs so that they can be subsequently reused.

The determination of a program to model a problem offers many advantages. Inspection of the genetic program solutions potentially offers understanding of the forces of behaviour behind the population. For example, Langley et al. [27] describe BACON, a system that successfully rediscovered the scientific laws; Ohm's laws, Coulomb's law, Boyle's law and Kepler's law from the given finite samples of data. As so many of the phenomena in our universe may be represented by programs, there is encouraging evidence for the need to explore a general program space for solutions to problems from that universe.

The initial preparation for a GP system has several steps.

Table 4
Main parameters used for the GP system

| | Value |
|---|---|
| Size of population | 1000 |
| Number of generations | 500 |
| Number of runs | 10 |
| Maximum initial full tree depth | 5 |
| Maximum number of nodes in a tree | 64 |
| Percentage of elitism | 5 |

First, it is necessary to choose a suitable alphabet of operands and operators. The operands are normally the independent input variables to the system and normally include an ephemeral random constant (ERC), which is a random variable from within a suitable range, e.g. 0–1.0. The operators should be rich enough to cover the type of functionality expected in solutions, e.g. trigonometric functions if solutions are expected to be periodic, but having too many operators can hamper convergence. Secondly, it is necessary to construct an initial population. In this paper, we use a set of randomly constructed trees from the specified alphabet, although this is dependent on the type of problem being solved and the representation chosen for the programs. For an initial population of trees, a good and common approach is called Ramped Half and Half. This means that half the trees are constructed as full trees, i.e. operands only occur at the maximum depth of the tree, and half are trees with a random shape. Within each half, an equal number of trees are constructed for each depth, between some minimum and maximum depths. This is found to give a good selection of trees in the original population. The main genetic operations used are reproduction and crossover. Mutation is rarely used, but other more specialised operators are sometimes used, but not for the problem tackled in this paper.

Reproduction is the copying of one individual from the previous generation into the next generation unchanged. This often takes the form of elitism, where the top $n\%$ of the solutions, as measured by fitness, is copied straight into the next generation, where $n$ can be any value but is typically 1–10.

The crossover operator chooses a node at random in the first chromosome, called crossover point 1, and the branch to that node is cut. Then it chooses a node at random in the second chromosome, called crossover point 2, and the branch to that node is cut. The two sub-trees produced below the cuts are than swapped. The method of performing crossover can be illustrated using an example, see Fig. 2a and b. Although this example includes a variety of operations, for simplicity in this application only the set {+, −, *} were made available. More complex operators can of course be developed by combining these simple operations. Simple operators eliminate the bounding problems associated with more complex operations such as XOR, which is not defined for negative or non-integer values. The multiply is a protected multiply which prevents the return of any values greater than $10^{20}$. This is to minimise the likelihood of real overflow occurring during the evaluation of the solutions. On average, 10% of the operands used were random constants in the range 0–1.0.

*6.2.1. Illustration of the crossover operator*

Parents before the crossover:

$$(a)\ \frac{3 + X_1}{X_3 - X_2} \qquad (b)\ \frac{(X_1 \mathrm{XOR} X_2) * (X_1 + 3)}{2}$$

New children produced by crossover operation:-

$$(a)\ \frac{(X_1 + 3) + X_1}{X_3 - X_2} \qquad (b)\ \frac{(X_1 \mathrm{XOR} X_2) * 3}{2}$$

Thus two new individuals are created, whose fitness can be evaluated.

This process is shown diagrammatically in Fig. 2a and b.

*6.2.2. Controlling the GP algorithm*

Since trees tend to grow as the algorithm progresses, a maximum depth is normally imposed (or maximum number of nodes) in order that the trees do not get too large. This is often a lot larger than the maximum size allowed in the initial population. Any trees produced by crossover that are too large are discarded. The reasons for imposing the size limit is to save on both the storage space required and the execution time needed to evaluate the trees. There is also no evidence that allowing very large trees will necessarily lead to improved results.

The basic algorithm is the same as for GA as given in Section 6.1.

Key parameters that have to be determined for any given problem are:

(a) The best way of choosing the alphabet, and representing the solution as a tree (or other structure).
(b) The best genetic operators.
(c) Choosing the best fitness function, to measure the fitness of a solution.
(d) Trying to keep enough diversity in the solutions in a population to allow the process to converge to the global optimum but not converge prematurely to a local optimum.
(e) Choosing sensible values for the population size, maximum tree size, number of generations etc. in order to get good solutions without using too much time or space.

More information related to GP can be found in Banzhaf [3] and in Koza [25,26].

## 7. Applying GP to software effort estimation

The software effort estimation problem is an example of a

Table 5
Comparing the best prediction systems, within each paradigm. Best performing estimators are highlighted in bold type. The figures for ANNs and GP are averages over ten executions of the systems. For Pred25, AMSE and MMRE; (*denotes significance at 5% level and **at 1% level)

| | Estimated effort | | | | | |
|---|---|---|---|---|---|---|
| | Random | Linear LSR | 2 nearest neighbours | 5 nearest neighbours | Artificial neural network | Genetic programming |
| Correlation | − 0.16589 | 0.557 | 0.550 | 0.586 | 0.635 | **0.752** |
| AMSE | 9.749167 | *7.378 | **6.432 | **5.733 | ****5.477** | 11.13 |
| Pred(25) | 3 | **10** | 8 | 8 | **10** | 4.2 |
| Pred(25)% | 16.67 | ****55.56** | **44.44 | **44.44 | ****55.56** | *23.3 |
| MMRE | 181.72 | **46.18 | 162.30 | 168.30 | **60.63 | ****44.55** |
| BMMRE | 191 | **59** | 66 | 70 | 69 | 75 |

symbolic regression problem, which means, given a number of sets of data values for a set of input parameters and one output parameter, construct an expression of the input parameters which best predicts the value of the output parameter for any set of values of the input parameters. In this paper, GP is applied to the Desharnais [8] data set already described in Section 3, with the same 63 projects in the Learning Set and remaining 18 projects in the Test Set as used for the other methods. The parameters chosen for the GP system, after a certain amount of experimentation, are shown in Table 4.

The results obtained depend on the fitness function used. In order to make comparison with the other methods, the fitness function was designed to minimise the MMRE measure as applied to the Learning Set of data. The values of MMRE quoted in the results are the result of applying the solution obtained to the Test Set of data.

The GP system is written in C and runs on a shared Sun $4 \times 170$ MHz Ultra Sparc, which means that timings are approximately equivalent to 300 MHz Pentium. However, the run-time core size is approximately 4 Mbytes to store the trees required in any one generation. One run of 500 generations takes 10 min c.p.u. time.

### 7.1. Comparing the GP results with other methods

The hypothesis to be tested is that GP can significantly produce a better estimate of effort than other techniques. In the comparison, those that contain a stochastic element are tested over a population of solutions, independently generated. The ANN uses random starting values, and GP's use random selection of initial population and crossover point. The range of solutions obtained has implications for both the accuracy of the solutions and the ease of configuration.

The software effort estimator must consider all available tools for improving the accuracy of estimates. The aim of this paper is to provide the estimator with the information to make an informed decision on the use of GP for this task. Since the data has been shown, to be modelled by a Weibull distribution (Section 3), it is possible to produce random sets of data by computer. This means that statistical tests can be produced to compare the predictors with random data to isolate statistically significant accuracy. Table 5 tests the predictors at 1 and 5% levels of significance. If the results between estimators show universal increases in accuracy, then, comparative statistical tests may also be applied.

Ease of configuration depends mainly on the number of parameters required to set up the learner. For example, a nearest neighbour system needs parameters to decide the weight of the variables and the method for determining closeness and combining neighbours to form a solution. Many learners need an evaluation function to determine error to decide on future search direction. The choice of method to make this evaluation is crucial for accurate modelling.

Koza [24] lists the number of control parameters for GP as being 20 as compared to the neural networks 10, but it is not always easy to count what constitutes a control parameter. However, it is not so much the number of the parameters as the sensitivity of the accuracy of the solutions to their variation. It is possible to search using different parameters but this will depend on their range and the granularity of search. In order to determine the ease of configuration for a genetic program, we test empirically whether parameter values suggested by Koza offer sufficient guidance to locate suitably accurate estimators. Similarly, all of our solutions use either limited experimentation or commonly used values for control parameters.

## 8. Results of the comparison

This study has evaluated various statistical and ML techniques including a GP tool to make software project effort predictions. We believe that the best way to assess the practical utility of these techniques would be to consider them within the context of their interaction with an example of an intended user, viz. a software project manager. However, since the data set we have used is one used by many other workers, and based on the late 1980s, this is not possible.

Thus we will perform the comparisons based on the accuracy of the results, the ease of configuration and the transparency of the solutions.

### 8.1. Accuracy of the predictions

Table 5 lists the main results used for the comparison.

Table 6
Population behaviour for the best and worst (chosen on the basis of MMRE) of the population of solutions from ANN and GP solutions

| | Estimated effort | | | | | |
|---|---|---|---|---|---|---|
| | Artificial neural network | | | Genetic programming | | |
| | Worst | Average | Best | Worst | Average | Best |
| Correlation | 0.588 | 0.635 | 0.650 | 0.612 | 0.752 | 0.824 |
| AMSE | 6.278 | 5.477 | 5.209 | 14.58 | 11.13 | 7.77 |
| Pred(25) | 10 | 10 | 10 | 2 | 4.2 | 5 |
| Pred(25)% | 56 | 56 | 56 | 11.2 | 23.5 | 28 |
| MMRE | 65.45 | 60.63 | 59.23 | 52.12 | 44.55 | 37.95 |
| BMMRE | 74 | 69 | 66 | 92.47 | 74.57 | 59.23 |

The various different types of regression equations all gave insignificantly differing results from each other, and so only, the linear LSR results are quoted. Results for ANNs are less accurate than those reported by Wittig and Finnie [24], although this is for a different data set and this may be, in part, due to the impact of their removing outlier projects in some of the validation sets. For the GP and ANN solutions, we have generated 10 solutions to assess reliability of accuracy. The population behaviour is summarised in Table 6.

The neural network seems to converge fairly consistently to a reasonable solution, with a difference of 6% in MMRE. In contrast, the GP system is consistently more accurate for MMRE but does not converge as consistently as the ANN to a good solution, with a 14% variation in MMRE. This suggests that more work needs to be done to try and prevent premature convergence in the GP system. For AMSE and Pred(25), the ANN is more accurate and consistent.

Early results from using GP, suggest that optimising one particular measure, in this case MMRE, has the effect of degrading a lot of the other measures and that a fitness function that is not specifically tied to one particular measure may give more acceptable overall results. This is illustrated in Tables 5 and 6, which give superior results for MMRE and correlation, but rather poor results for Pred(25), AMSE and BMMRE. This suggests that more research is required not only on GP, but also on the problem itself as to which of the many measures or combination of measures is the most appropriate in practice.

### 8.2. Transparency of the solution

One of the benefits of LSR is that it makes explicit the weight and contribution of each input used by the prediction system. This can lead to insights about the problem for example to direct efficiencies. CBR, or estimation by analogy, also has potential explanatory value since projects are ordered by degree of similarity to the target project. Indeed, it is instructive that this technique demonstrates the effectiveness of user-involvement in performing better, when the user is able to manipulate the data and modify predicted outputs. However, although this suggests an understanding of the data by the user, it gives little indication of the contribution of specific variables. Changing parameters could allow some exploration of this space, but complex interactions between variables and small data sets to make exploration limited.

The neural nets used within this study do not allow the user to see the rules generated by the prediction system. If a particular prediction is surprising, it is hard to establish any rationale for the value generated. It is difficult to understand an ANN merely by studying the net topology and individual node weights. To extract rules from the best ANN a method of pruning was used as suggested by Lefley [29]. This reduced the net to the following nodes and links that were making a significant contribution to the error.

```
Node 10
−0.35*X7 + −0.81*X8
Node 11
1.60 + 0.39*X1 − 0.69*X4 − 0.67*X6−
1.075981*X7 − 1.40*X8
Node 12
0.42*X5 + 0.61*X7
Node 13
−0.28*X6 − 0.74*X7 − 3.24*X8
Node 14
0.32*X6 + 0.33*X7 + 0.52*X8
Output node 15
1.59 − 2.27*N10 − 4.22*N11 + 1.12*N12−
2.41*N13 + 1.09*N14.
```

Though this does not paint a very clear picture of how a decision is made it indicates the importance of variables *X*6, *X*7 and *X*8 (see Table 1) and careful examination might yield a little more information. However, even with such an analysis, ANNs do remain hard to interpret, certainly harder than say LSR, for small gains in terms of accuracy.

Potentially, GP can produce very transparent solutions, in the sense that the solution is an algebraic expression. At present, the good solutions for this problem have the maximum number of nodes, i.e. 64 operators or operands and the expressions do not simplify a great deal. Further work may find solutions with less nodes, which may make the results more understandable analytically. In contrast, increasing the number of nodes may provide better estimates, but the results are likely to be less transparent and the computation time will increase significantly.

### 8.3. Ease of configuration of the system

The third factor in comparing prediction systems is what we term ease of configuration. In other words how much effort is required to build the prediction system in order to generate useful results. Regression analysis is a well-established technique with good tool support. Even allowing for analysis of residuals and so forth, little effort needs to be

expended in building a satisfactory regression model. CBR needs relatively little work though more might be gained by relative weighting of the inputs. The number of neighbours did not seem to have a significant effect. By contrast, we found it took some effort to configure the neural net and it required a fair degree of expertise. Generally, different architectures, algorithms and parameters made little difference to results but some expertise is required to choose reasonable values. Although heuristics have been published on this topic [17,44], we found the process largely to be one of trial and error guided by experience. ANN techniques are not easy to use for project estimation by end-users as some expertise is required.

GP has many parameters too, the choice of functions, crossover and reproduction rates, percentage of elitism, dealing with out of bounds conditions, creation strategy and setting maximum tree sizes and depths, i.e. code lengths, to name some of the more significant. We found by following the decisions suggested by Koza [24], we obtained good results but again, at present, some expertise is still needed.

## 9. Conclusions and future work

In this paper, we have compared techniques for predicting software project effort. These techniques have been compared in terms of accuracy, transparency and ease of configuration. Despite finding that there are differences in prediction accuracy levels, we argue that it may be other characteristics of these techniques that will have an equal, if not greater, impact upon their adoption. We note that the explanatory value of both estimation by analogy (CBR) and RI, gives them an advantage when considering their interaction with end-users.

Problems of configuring neural nets tend to counteract their superior performance in terms of accuracy. We are encouraged that this early investigation of GP shows it can provide accurate estimates. The results highlight that measuring accuracy is not simple and researchers should consider a range of measures. Further, we highlight that where there is a stochastic element, learners must also show acceptable, consistent accuracy for a population of solutions, which complicates simple comparisons.

We believe these results show that this approach warrants further investigation, particularly to explore the effects of various parameters on the models in terms of improving robustness and accuracy. It also offers the potential to provide more transparent solutions but this aspect also requires further research. Perhaps a useful conclusion is that the more elaborate estimation techniques such as ANNs and GPs can provide better accuracy but require more effort in setting up and training. A trade off between accuracy from complexity and ease of interpretation also seems inevitable.

The authors are grateful to Jean-Marc Desharnais for making his data set available. We also thank the journal reviewers for their helpful comments which have significantly improved the quality of this report.

## References

[1] A. Aarmodt, E. Plaza, Case-based reasoning: foundational issues, methodical variations and system approaches, AI Commun. 7 (1994) 39–59.
[2] W.D. Aha, Case-Based Learning Algorithms, 1991 DARPA Case-Based Reasoning Workshop, Morgan Kaufmann, Los Atlos, CA, 1991.
[3] W. Banzhaf, P. Nordin, R.E. Keller, F.D. Francome, Genetic Programming: An Introduction, Morgan Kaufmann, Los Atlos, CA, 1998.
[4] R. Bisio, F. Malabocchia, Cost estimation of software projects through case based reasoning, International Conference on Case Based Reasoning, Sesimbra, Portugal, 1995.
[5] B.W. Boehm, Software Engineering Economics, Prentice Hall, New York, 1981.
[6] S. Conte, H.E. Dunsmore, V.Y. Shen, Software Engineering Metrics and Models, Benjamin/Cummings, Menlo Park, CA, 1986.
[7] J.C.W. Debuse, V.J. Rayward-Smith, Feature subset selection within a simulated annealing data mining algorithm , J. Intell. Inf. Syst. 9 (1997) 57–81.
[8] J.M. Desharnais, Analyse statistique de la productivitie des projets informatique a partie de la technique des point des fonction, Unpublished Masters Thesis, University of Montreal, 1989.
[9] J.J. Dolado, A study of the relationships among Albrecht and Mark II Function Points, lines of code 4GL and effort , J. Syst. Soft. 37 (1997) 161–172.
[10] J.J. Dolado, A validation of the component-based method for software size estimation, IEEE Trans. Soft. Engng 26 (2000) 1006–1021.
[11] J.J. Dolado, On the problem of the software-cost function, Inf. Soft. Tech. 43 (2001) 61–72.
[12] S. Drummond, Measuring applications development performance, Datamation 31 (1985) 102–108.
[13] L. Fernandez, J.J. Dolado, Measurement and prediction of the verification cost of the design in a formalized methodology, Inf. Soft. Tech. 41 (1999) 421–434.
[14] D.E. Goldberg, Genetic algorithms in search, optimisation and machine learning, Addison Wesley, Reading, MA, 1989.
[15] A.R. Gray, S.G. MacDonell, A comparison of techniques for developing predictive models of software metrics, Inf. Soft. Tech. 39 (1997) 425–437.
[16] T. Hegazy, O. Moselhi, Analogy based solution to markup estimation problem, J. Comput Civ. Engng 8 (1994) 72–87.
[17] S. Huang, Y. Huang, Bounds on the number of hidden neurons, IEEE Trans. Neural Network 2 (1991) 47–55.
[18] R.T. Hughes, Expert judgement as an estimating method, Inf. Soft. Tech. 38 (1996) 67–75.
[19] M. Jorgensen, Experience with the accuracy of software maintenance task effort prediction models, IEEE Trans. Soft. Engng 21 (1995) 674–681.
[20] N. Karunanithi, N. Whitley, Y.K. Malaiya, Using neural networks in reliability prediction, IEEE Soft. 9 (1992) 53–59.
[21] C.F. Kemerer, An empirical validation of cost estimation models, CACM 30 (1987) 416–429.
[22] H.C. Kennedy, C. Chinniah, P. Bradbeer, L. Morss, Construction and evaluation of decision trees: a comparison of evolutionary and concept learning methods, in: D. Corne, J.L. Shapiro (Eds.), Evolutionary Computing, AISB International Workshop, Manchester, UK, LNCS 1305, 1305, Springer, Berlin, 1997.
[23] P. Kok, B.A. Kitchenham, J. Kirakowski, The MERMAID approach to software cost estimation, Proceedings of Esprit Technical Week, 1990.

[24] J.R. Koza, Genetic Programming: On the Programming of Computers by Natural Selection, MIT Press, Cambridge, 1993.
[25] J.R. Koza, Genetic Programming II: Automatic Discovery of Reusable Programs, MIT Press, Cambridge, 1994.
[26] J.R. Koza, F.H. Bennett, D. Andre, M.A. Keane, Genetic Programming III: Darwinian Invention and Problem Solving, Morgan Kaufmann, Los Atlos, CA, 1999.
[27] P.S.H.A. Langley, G.L. Bradshaw, J.M. Zytkow, Scientific Discovery: Computational Explorations of the Creative Process, MIT Press, Cambridge, 1987.
[28] D. Leake, Case-Based Reasoning: Experiences Lessons and Future Directions, AAAI Press, Menlo Park, 1996.
[29] M. Lefley, T. Kinsella, Investigating neural network efficiency and structure by weight investigation, Proceedings of the European Symposium on Intelligent Technologies, Germany 2000.
[30] C. Mair, G. Kadoda, M. Lefley, K. Phalp, C. Schofield, S. Shepperd, S. Webster, An investigation of machine learning based prediction systems, J. Soft. Syst. 53 (1) (2000).
[31] R. San Martin, J.P. Knight, Genetic algorithms for optimization of integrated circuit synthesis, Proceedings of Fifth International Conference on Genetic Algorithms and their Applications, Morgan Kaufman, San Mateo, CA, 1993, pp. 432–438.
[32] M. Mitchell, Introduction to Genetic Algorithms, MIT Press, Cambridge, 1996.
[33] Y. Miyazaki, A. Takanou, H. Nozaki, N. Nakagawa, K. Okada, Method to estimate parameter values in software prediction models, Inf. Soft. Tech. 33 (1991) 239–243.
[34] K.S. Narendra, K. Parthasarathy, Identification and control of dynamical systems using neural networks, IEEE Trans. Neural Network 1 (1987) 4–27.
[35] E. Rich, K. Knight, Artificial Intelligence, McGraw-Hill, New York, 1995.
[36] R. Schank, Dynamic Memory: A Theory of Reminding and Learning in Computers and People, C.U.P., 1982.
[37] T.J. Sejnowski, C.R. Rosenberg, Parallel networks that learn to pronounce English text, Complex Syst. 1 (1987) 145–168.
[38] M.J. Shepperd, C. Schofield, Estimating software project effort using analogies, IEEE Trans. Soft. Engng 23 (1997) 736–743.
[39] K.K. Shukla, Neuro-genetic prediction of software development effort, Inf. Soft. Tech. 42 (2000) 701–713.
[40] K. Srinivasan, D. Fisher, Machine learning approaches to estimating software development effort, IEEE Trans. Soft. Engng 21 (1995) 126–136.
[41] E. Stensrud, I. Myrtveit, Human performance estimating with analogy and regression models: an empirical validation, Proceedings of the Fifth International Metrics Symposium, IEEE Computers and Society, Bethesda, MD, 1998.
[42] A.R. Venkatachalam, Software cost estimation using artificial neural networks, International Joint Conference on Neural Networks, IEEE, Nagoya, 1993.
[43] S. Vincinanza, M.J. Prietula, Case based reasoning, software effort estimation, Proceedings of 11th International Conference on Information Systems, 1990.
[44] S. Walczak, N. Cerpa, Heuristic principles for the design of artificial neural networks, Inf. Soft. Tech. 41 (1999) 107–117.
[45] L. Watson, F. Marir, Case-based reasoning: a review, The Knowledge Engng Rev. 9 (1994) 327–354.
[46] G. Wittig, G. Finnie, Estimating software development effort with connectionist models, Inf. Soft. Tech. 39 (1997) 469–476.