Using Genetic Algorithms to Generate Estimation Models

D. Rodriguez[1], J.J. Cuadrado-Gallego[2] and J. Aguilar[3]

[1] Dept of Comp Sci
The University of Reading
PO Box 225, Whiteknights
Reading RG6 6AY
UK

[2] Dept of Comp Sci
University of Alcala
Ctra Barcelona km 33.6
28871 Alcala de Henares
Madrid, Spain

[3] Dept of Comp Sci
Univ Pablo De Olavide
Ctra. de Utrera, km. 1
41013 Sevilla
Spain

Abstract

Parametric software estimation models rely on the availability of historical project databases from which estimation models are derived. In the case of large project databases, problems can arise such as het-eroscedasticity where the size of a project can in°uence the accuracy of the estimation method.In such cases, a single mathematical model may not properly be used to estimate projects of diverse nature. In this work, we discuss how genetic algorithms can be applied to produce segmented models, i.e., the genetic algorithm searches for cut-points in the range of a variable (e.g. Function Points), and di®erent estima-tion models can be used at each side of the cut-point. A concrete case study using the ISBSG dataset is reported. Results show that with a very low number of models instead of a single one, the accuracy can be increased significantly.

Keywords:

Software Engineering, Effort estimation, Evolutionary algorithms, Genetic Algorithms

1 Introduction

1.1Parametric estimation techniques are nowadays widely used to measure and/or estimate the cost associated to software development [1]. The *Parametric Estimating Handbook* (PEH) [6] defines parametric estimation as "a technique employing one or more cost estimating relationships (CERs) and associated mathematical relationships and logic". Parametric techniques are based on identifying significant CERs that obtain numerical estimates from main *cost drivers* that are known to affect the effort or time spent in development. Parametrics uses the few important parameters that have the most significant cost impact on the software being estimated.

Parametric based estimation requires historical project databases as the empirical baseline for the models. During the last decade, several organizations such as the International Software Benchmarking Standards Group (ISBSG http://www.isbsg.org/) have started to collect project management data from a variety of organizations. One important aspect of the process of deriving models from

such databases is the heterogeneity of the data. Heteroscedasticity (i.e. nonuniform variance) is known to be a problem affecting data sets that combine data from heterogeneous sources [7]. When using such databases, classical regression models that derive a single mathematical model results in poor adjustment to data and subsequent poor accuracy. This is due to the fact that a single model can not capture the diversity of distribution of different segments of the database points. As an illustrative example, the straightforward application of a standard linear regression in the *Reality* tool of the ISBSG 8 database distribution results in measures of a *correlation coefficient* of 0.43 and a *relative absolute error* of 80% which are poor figures of quality of adjustment.

In this work, we use genetic algorithms to search for cut-points in the range of a variable (e.g. *Function Points*), so that different regression models can be used depending on the value of such variable and the accuracy of the estimates can be largely improved.

The rest of this paper is structured as follows. In Section 2, the ISBSG repository and data used are described. Section 4 describes the overall results of the clustering process. Section ?? provides the discussion of the empirical evaluation of the approaches described. Finally, conclusions and future research directions are described in Section 5.

2 The ISBSG Repository

The International Software Benchmarking Standards Group (ISBSG), a non-profit organization, maintains a software project management repository from a variety of organizations. In this work, we have used ISBSG release 8, which contains 2028 projects and more than 55 attributes per project. The attributes can be classified as follows:

- Project context such as type of organization, business area, and type of development.
- Product characteristics such as application type user base.
- Development characteristics such as development platform, languages, tools, etc.
- Project size data: di®erent types of function points (IFPUG, COSMIC, etc.)
- Qualitative factors such as experience, use of methodologies, etc.

However, before using the dataset, there are a number of issues to be taken into consideration regarding data preparation in order to generate the 2 dataset used in work. An important attribute is the quality rating given by the ISBSG which can go from A (where the submission satisfies all criteria for seemingly sound data) to

D (where the data has some fundamental shortcomings). According to ISBSG only projects classified as A or B should be used for statistical analysis so therefore only projects with A or B quality where selected.

Instance projects that used other other estimation method than IFPUG, NESMA, Albretch or Dreger were removed, since they represented smaller portions of the database. The differences between IFPUG and NESMA methods are considered to have a negligible impact on the results of function point counts [5]. Counts based on Albretch techniques were not removed since in fact IFPUG is a revision of these techniques, similarly, the Dreger method refers to the book [2], which is simply a guide to IFPUG counts. We also have selected *NormalisedWorkEffort* attribute as the dependent variable for generating linear regressions in the fitness function. The normalized work effort is an estimate of the effort for the whole software life-cycle even if the project did not cover all the phases in the software development life-cycle. The dataset used in this work with their attribute considered are:

Reality Dataset - DS1 This dataset is provided the ISBSG as part of the *Reality Checker* tool provided as part of the repository. The Reality dataset if composed of 709 instances and 6 attributes (*DevelopmentType, DevelopmentPlatform, LanguageType, ProjectElapsedTime, NormalisedWorkE®ort, UnadjustedFunctionPoints*).

NormWE Dataset - DS2 This dataset is composed of 1390 instances and 15 attributes (*FP, VAF, MaxTeamSize, DevelopmentType, DevelopmentPlatform, LanguageType, DBMSUsed, MethodUsed, ProjElapTime, ProjInactiveTime, PackageCustomisation, RatioWEProNonPro, TotalDefectsDelivered, NormWorkEff, NormPDR*).

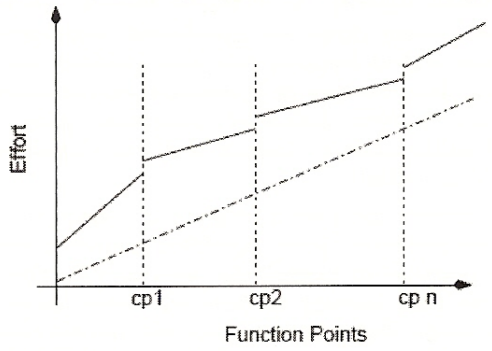

Figure 1: Single vs. Multiple models

3 Applying Evolutionary Computation to the ISBSG Repository

Evolutionary computation provides an interesting approach for dealing with the problem of generating multiple parametric models. In this case, the search space comprises a set of cut-points, so a di®erent parametric estimation model can be used for the intervals that comprise such cut-points. Two critical factors in°uence the cut-points obtained by using evolutionary algorithms: (i) the selection of an internal representation of the search space (*coding*) and (ii) the defnition of an external function that assigns a value of goodness to the potential solutions (*fitness*).

3.1 Evolutionary Computation in a Nutshell

Evolutionary Algorithms (EA) are a family of computational models inspired by the concept of evolution and natural selection. These algorithms employ a ramdomised search method to find approximate optimal solutions to a particular problem [3]. Generally, this approach is applied to search spaces that are too large to use exhaustive techniques.

Based on a generally constant size *population* of individuals, an EA follows an iterative procedure based on selection and recombination operators to generate new individuals in the search space. Such individuals are usually represented by a finite string of symbols called *chromosome*. They encode a possible solution in a given problem search space which comprises of all possible solutions to the problem. The length of the string and the population size are completely dependent of the problem in hand and the finite string of symbol alphabet can be binary, real{valued encodings, tree representations, etc. The population simulates natural evolution in the sense that iteratively good solutions (individuals) generate other solutions (offsprings) to replace bad ones retaining many features of their parents. Therefore, a critical factor is to know how good is a solution which depends on a *fitness function*, so that high{fitness individuals stand a better chance of reproducing, while others are likely to disappear. Another critical factor is how new solutions are formed. This is usually carried out using two genetic operators named *crossover* and *mutation*. Crossover creates a new individual combining parts of its parents representation. Finally, mutation changes randomly a small part of the string that represents the individual.

Evolutionary algorithms (EAs) are a family of computational models inspired by the concept of evolution. These algorithms employ a randomized search method to find solutions to a particular problem [7]. This search is quite di®erent from the other learning methods mentioned above. An EA is any populationbased model that uses selection and recombination operators to generate new sample examples in a search space [8]. The EA search can move much more abruptly, replacing a parent individual with an offspring less likely

to fall into the same kind of local minima which can happen with the other methods. EAs have been used in a wide variety of optimization tasks [9], [10] including numerical optimization and combinatorial optimization problems, although the range of problems to which EAs have been applied is much broader. The main task in applying EAs to any problem consists in selecting an appropriate representation (coding) and an adequate evaluation function (fitness). In classical EAs the members of the population (typically maintaining a constant-size) are represented as fixed-length strings of binary digits. The length of the strings and the population size are completely dependent on the problem. The population simulates natures behavior, since the relatively good solutions produce offspring which replace ones that are relatively worse, retaining many of the features of their parents. The estimate of the quality of a solution is based on a fitness function, which determines how good an individual within the population in each generation is. New individuals (offspring) for the next generation are formed by using (normally) two genetic operators: crossover and mutation. Crossover combines the features of two individuals to create several (commonly two) individuals. Mutation operates by randomly changing several components of a selected individual.

We compare the benefits of the techniques by using linear regression and least median square as prediction techniques before and after characterising the database using the classical Mean Magnitude of Relative Error (MMRE) and Pred(%).

In Software Engineering, the standard criteria for a model to acceptable are Pred(25) ? 0.75 and MMRE ? 0.25. "

MMRE is computed as , where in a sample of size n, i is the estimated value for the i-th element, and ei is the actual value.

Pred(%) is defined as the number of cases whose estimations are under the %, divided by the total number of cases. For example, Pred(25)=0.75 means that 75% of cases estimates are within the inside 25% of its actual value.

@relation AttribSelecc.Final-weka. filters.unsupervised.attribute. Remove-R20-25-weka.filters.supervised.instance.StratifiedRemoveFolds-S0-N3-F1

3.1.1 Coding

In this paper, we used positive integers for our coding method as it more natural to the software engineering attributes that we are interested in such as *function points*. In this way, every interval is encoded by one natural number, leading to a reduction of the search space size when compared with binary strings typically used by generic algorithms.
For example, if we consider 1 cut-point in the range of *FunctionPoints 2* [0, 1500], we could generate a cut-point $c1 = 750$ such that an individual of the population will be [750]. Then, we can generate 2 regression models; the first

one using projects which their Function Points values 2 [0, 750] and another one with the rest of the projects 2 [751, 1500]. Similarly, if we have 2 cut-points per individual, (e.g. [350, 850], we can generate 3 regression models, etc.

3.1.2 Fitness Function

The fitness function must be able to discriminate between correct and incorrect classifiations of samples. Finding an appropriate function is not a trivial task, due to the noisy nature of most databases. The evolutionary algorithm minimizes (or maximizes) the fitness function *f* for each individual of the population. We have used some of the evaluation methods provided by Weka [9] and particulary for this work, we have used the *relative absolute error* value:

$$\frac{|p_1 - a_1| + \ldots + |p_n - a_n|}{|a_1 - \overline{a}| + \ldots + |a_n - \overline{a}|} \qquad (1)$$

where $p_1; p_2;$, p_n are the predicted values and $a_1; a_2;$, a_n are the actual values and $\overline{a}$: $1/n \sum_i a_i|$

Such actual and predicted values are generated by classical regression techniques [?], which are a kind of algorithmic techniques which looks for an equational model to fit a set of observed data values. Linear Regression (LR) is the classical linear regression model. It is assumed that there is a linear relationship between a dependant variable (e.g., *effort*) with a set of or independent variables, i.e., attributes (e.g. *size in function points, team size, development platform,* etc.). The aim is to adjust the data to a model so that $y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + + \beta_k x_k + e$.

3.2 Mutation

Mutation, in analogy to biological mutation, is is used to maintain greater diversity from one generation of a population of chromosomes to the next one. In this study, we have used *uniform mutation* [8], which replaces the value of the chosen gene with a uniform random value within the range specified by the user. We have select a range of 30% either side on the cut-point. To do so, we just multiply the gene int he selected chromosome by 1+30*random(), where random() is a function that returns a real value between 0 and 1.

3.2.1 Algorithm

The algorithm used is the typical sequential covering evolutionary algorithm [4]. Table ?? gives the values of the parameters involved in the evolutionary process.

Table 1: Parameters of the evolutionary algorithm

Parameter	Value
Population size	20
Generations	50
Crossover probability	0.6
Individual Mutation probability	0.2

The method of generating the initial population consists of randomly selecting natural number with the range of the Function Points attribute. A termination criterion is reached when the individuals of the population have converged or the maximum number of generations has been reached.

4 Results and Discussion

The problem of heterostecasticity of software engineering datasets has been reported elsewhere [7]. An simple approach to alleviate this problem and a way to improve the general accuracy of classical regression estimation models, it is to use multiple of models depending on a variable of interest. As a case study using Weka for this work, a simple application of the classical linear regression to the ISBSG datasets gives the following expressions:

```
NormalisedWorkEffort =
  2940.5448 * DevPlat=MF,MR +
  2070.5142 * LangType=ApG,3GL,2GL +
   447.2816 * ProjElapsedTime +
     6.151  * UnadjustedFPs +
 -4293.9475
```

```
NormalisedWorkEffort =
     4.9692 * FP +
   174.4352 * MaxTeamSize +
   979.7368 * DevType=NewDev,Re-dev +
  4391.6311 * DevType=Re-dev +
 -1864.7412 * LangType=2GL,3GL,ApG +
   381.9574 * ProjElapTime +
  -617.8424 * ProjInactiveTime +
  2628.9819 * PkgCustom=Yes +
 -5270.6782
```

And Table 4 provides a summary of di®erent error measures. These values are pretty poor for any data mining or software engineering standards, let alone project managers in need of decision making tools.

Table 2: Summary error for the DS1 and DS2 datasets

	DS1	DS2
Correlation coefficient	0.438	0.781
Mean absolute error	5100.2	3420.50
Root mean squared error	11973.86	7463.64
Relative absolute error	80.01%	59.56%
Root relative squared error	89.9%	62.44%

However, Tables 3 and 4 show how the error can be significantly reduced if we divided the dataset according to a variable of interest. In this study, *Function Points* has been selected as it seems the most natural one to when we used to

estimate the e®ort based on the size of a project. Table 4 also shows the errors and cut-point values in for the DS2 dataset. As in Table 4 the largest improvements occur with a very low number of cut-points. Figure 2 shows how creating a low number of cut-points can improve the the accuracy significantly. In the Reality (DS1) dataset, the accuracy improves 30% with just 2 cut-points, i.e., 3 models. With the DS2 dataset, Figure 2 shows how the error monotonically decreases as we were expecting

Table 3: Error and Cut-points for DS1

	Error	Cut-Points in the FP axis
1	79.12	[1744]
2	54.66	[1378, 1586]
3	48.35	[1334, 1609, 2036]
4	50.89	[721,1247, 1484, 1553]
5	55.1	[135, 904, 1292, 1473,1747]
6	51.44	[457, 1210, 1315, 1534, 1835, 2433]
7	45.54	[382, 1045, 1163, 1245, 1409, 1593, 2064]
8	55.76	[330, 509,1005, 1351, 1506, 1638, 2130, 2378]
9	58.63	[247, 496, 649, 944, 1243, 1452, 1607, 2052, 2248]
10	63.04	[140, 350, 889, 1083, 1191, 1284, 1477, 1612, 2011, 2441]

Table 4: Error and Cut-points for DS2

	Error	Cut-Points in the FP axis
1	36.48	[2562]
2	28.05	[2081, 2238]
3	23.94	[1235, 1280, 1408]
4	20.57	[1206, 1250, 1373, 1667]
5	19.66	[342, 535, 1039, 2041, 2474]
6	18.55	[339, 963, 1238, 1594, 1781, 2185]
7	16.75	[507, 1260, 1395, 1462, 2047, 2090, 2359]
8	14.71	[356, 688, 1180, 1303, 1655, 1734, 2014, 2091]
9	13.23	[321, 630, 863, 890, 1014, 1881, 2004, 2160, 2663]
10	13.43	[229, 461, 705, 919, 962, 1218, 1389, 1666, 1893, 2484]

until we could hypothetically reach a value of perfect accuracy when we have as many cut-points as values in the dataset. For practical purposes, it seems that 2 or 3 cut-points works best.

5 Conclusions and Future Work

With the inception of several organizations such as ISBSG, there are a number of repositories of project management data. The problem faced by project managers is the large disparity of the their instances so that estimates using classical techniques is not accurate. In this work, we have shown how evolutionary computation can be used to group instances from software engineering databases. The algorithm searched for cut-points across the Function Points range so that

different regression models are created so that the error is minimized. In our case, this was used

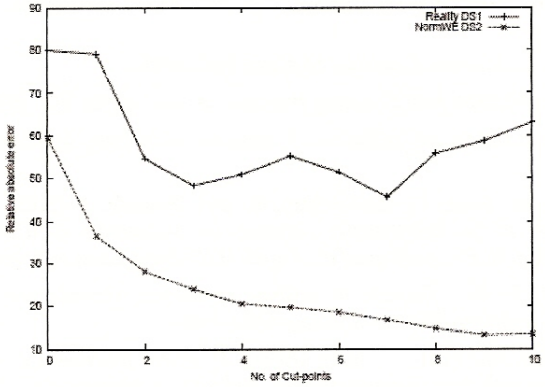

Figure 2: Error vs. no of cut-points

to provide multiple estimation models or a segmented model, such that a range in a variable of interest has an associated estimation model. Results with the ISBSG dataset has proven that with just creating such submodels across the range of a variable of interest proves to be more accurate. The comparison of using parametric models for each cluster and using the built{in cluster characterizations has resulted in evidence that the parametric approach has an improvement in average accumulated error, but not in overall predictive properties. Further work will consist of further research into using di®erent parameters for the evolutionary algorithm, fitness functions, etc. We are also looking into using evolutionary computation for feature selection, i.e., attribute selection (in this work, the attributes were selected manually using expert knowledge). More needs to be done also understanding and comparing different clustering techniques to create segmented models and its usefulness for project managers.

Acknowledgment

This research was supported by the Spanish Research Agency (CICYT TIN2004-06689-C03).

References

[1] Boehm, B., Abts, C. and Chulani, S.. . Software Development Cost Estimation approaches { a survey. *USC Center for Software Engineering Technical Report # USC-CSE-2000-505*, 2000.

[2] Dreger, J. Brian. *Function Point Analysis*. Englewood Cliffs, NJ: Prentice Hall, 1989.

[3] Koza, J.R.: *Genetic Programming*. The MIT Press, Cambridge, Massachusetts, 1992.

[4] Mitchell, T., *Machine Learning*. McGraw Hill, 1997.

[5] NESMA . *NESMA FPA Counting Practices Manual* (CPM 2.0), 1996.

[6] Parametric Estimating Initiative, *Parametric estimating handbook*, 2nd edition, 1999.

[7] Stensrud, E., Foss, T., Kitchenham, B. and Myrtveit, I. (2002). An Empirical Validation of the Relationship Between the Magnitude of Relative Error and Project Size. In *Proceedings of the Eighth IEEE Symposium on Software Metrics*

[8] Goldberg, D. E. . *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Reading (MA), 1989.

[9] Witten, I.H. and Frank, E. *Data Mining, Practical Machine Learning Tools and techniques with Java Implementations. 2nd Edition*. Morgan Kaufmann Publishers, San Francisco, California, 2005.

(adapted to the MetriKon-Template by the editors)