A Genetic Algorithm Approach for the Calibration of COCOMO-like Models

Roberto Cordero
Mario Costamagna
Elio Paschetta

e-mail: (mario.costamagna, elio.paschetta)@cselt.it
CSELT, Centro Studi e Laboratori Telecomunicazioni
Via G. Reiss Romoli, 274
10148 Torino, Italy
Tel.: +39 - 11 - 2285111
Fax.: +39 - 11 - 2285520

This work concerns the usage of Genetic Algorithms for the calibration of COCOMO-like models and has been developed during the MS thesis of Roberto Cordero.

CSELT is the STET Group's Company for the study, research, experimentation and qualification for telecommunications and information technology. It contributes to the study of advanced systemistic scenarios, plays a link role between academic and applied research and has a strong presence in the international context, through the participation in common research programs and standardization activities.

The authors are researchers in the software quality and management research unit where several activities connected to cost estimation, process assessment (CSELT has given an active contribution to SPICE) and software procurement are conducted. The present work is in the context of cost estimation.

Although progress is being made towards developing effective estimation models, they can produce highly inaccurate estimates unless calibrated for the specific environment in which they will be used.

The calibration of mathematical estimation models (i.e., based on a mathematical formulation like COCOMO and COCOMO II) is usually done by means of regression methods based on ordinary least-square minimization. These methods have several drawbacks when used in the context of estimation models. The recalibration method we propose here addresses all these drawbacks. This new method uses the Differential Evolution, a recent development of the Genetic Algorithms family, as a search algorithm, instead of the least-square regression.

To evaluate our calibration and to compare it with the original model, and with a calibration obtained with linear regression, we used our new objective function. The results show that our calibration algorithm performs better than linear regression and leads to a new version of the model that is better than the original one (with respect to our objective function).

The generality of the approach and the results obtained during the experimentation give us the confidence about the usability of the algorithm also for COCOMO II and other cost n...
For this work we used COCOMO instead of COCOMO II because when we started in 1996:

- we have had experience in using only the first model;
- COCOMO II was not yet a complete model;
- the projects data base on which the COCOMO model was built is of public domain and it would be easy to compare recalibration results with those of the original model;
- besides, in literature we found some other works about COCOMO (and other models) recalibration towards which we could make a comparison.
The formulation of the COCOMO intermediate model is reproduced in this slide to show what we intend for the recalibration process. The COCOMO formula contains 64 coefficients: a and b, for the 3 development mode, plus all the coefficients that represent the Cost Drivers (CDs). In the tables is reported only a small portion of them.

In our process the calibration algorithm operates on the projects data base and the model in order to find values of the model parameters that best fit the projects DB.

In doing the calibration the algorithm has one or more objectives to optimize and some constraints to impose, like the shape of the CDs.

In the original COCOMO intermediate model the cost factors have functions shaped in three possible ways: decreasing, increasing, and U-shaped. If one do not impose these constraints, a direct recalibration of the COCOMO parameters would produce non-monotonic and non U-shaped functions.
Intro (2): Calibration Problems

- Drawbacks of regression methods on COCOMO:
  - They need a linear formulation of the model
  - COCOMO is rewritten with a linear formulation
  - Outliers have a great influence over regression
  - Software data tend to include more outliers than data in other fields of statistics
  - They only minimize the total error
    - and do not optimize other important characteristics of goodness of fit (e.g., ARE(25) < the rate of the projects of which absolute values of relative errors are < 0.25)
  - A direct recalibration of all the cost drivers would produce not good shaped functions.
Literature Results

- **Boehm '81 (Software Engineering Economics)**
  - Linear regression: cannot recalibrate CDs (unless you accept oscillating CDs)

- **Gulezian '91 (JSS)**
  - Linear regression: calibration of a, b, and CDs keeping the original shape (and starting from the original CD)

- **Miyazaki '94 (JSS)**
  - Non linear robust regression, independence from outliers: when applied on COCOMO for a, b, and CDs produces many local minimums (difficult to solve)

- **Paschetta '96 (Escom 96)**
  - Linear regression: calibration of a, b, and CDs without imposing the shape (without imposing initial CDs and assuring they have a resulting good shape)

12 th COCOMO Forum '97
The calibration of COCOMO done by Gulezian starts from a reformulation of the model. In his calibration the original values of Cost Drivers (indicated in his formula by $C_{ij}$) are used as a sort of starting point so that the computed values are not too different from the original one (and the CD shapes are the same).

The additional terms introduced into the reformulated structure are defined as follows:

- $a = \text{nominal coefficient for the semidetached mode}$;
- $a_1 = \text{multiple of } a \text{ yielding the nominal coefficient for the organic mode}$;
- $a_3 = \text{multiple of } a \text{ yielding the nominal coefficient for the embedded mode}$;
- $b = \text{nominal coefficient for the semidetached mode}$;
- $b_1 = \text{addition to } b \text{ yielding the nominal coefficient for the organic mode}$;
- $b_3 = \text{addition to } b \text{ yielding the nominal coefficient for the embedded mode}$;
- $m_1 = \text{binary variable which assumes a value of one for the organic mode and zero otherwise}$;
- $m_3 = \text{binary variable which assumes a value of one for the embedded mode and zero otherwise}$;
- $d_i = \text{coefficient corresponding to the } i\text{-th cost driver}$.

The method of solution is the traditional least-squares estimation, which underlies basic multiple linear regression analysis. This is accomplished by linearizing the equation form by taking logarithms. The sum of squares to be minimized directly takes the form represented in the figure.
The approach of Miyazaki et al. (see references) relies on the assumption that in the field of software estimation models the usage of ordinary relative error and ordinary least-squares method yields to solutions easily distorted by outliers. A solution that shows the tendency of majority is usually more practical than a solution distorted by a few outliers. When statistical techniques are used for the data from software projects, robustness of the techniques is very important to find a good solution. This is because software data tend to include more outliers than data in other field of statistics for several reasons (i.e. software terminology is not standardized, software data from a large project cannot be collected by one person, software development is not standardized enough, and the range of software data is quite large).

Taking the importance of robustness into consideration, Miyazaki et al. refine the least-squares method based on relative errors to obtain a simple and robust method called least-squares of inverted balanced relative errors and demonstrate its superiority to other methods using various actual data sets. The resulting system obtained with these errors are not linear and so it requires a major computational efforts. Miyazaki et al. in their work did not mention the applicability to COCOMO calibration.

We tried to apply the Miyazaki approach to recalibrate the COCOMO model but we found out that the non linear system to solve for the least-squares minimization has not a unique solution: that formulation has many minima! This is a very hard mathematical problem.
In a previous work of us we used a COCOMO-like model as a productivity estimation model for the evaluation of software suppliers in the Vendor Rating process of a telecommunications industry.

Each productivity factor is a function of the corresponding productivity index. The index may assume values from 1 to 5 (1 means very low, 5 means very high and fractional values are allowed to express intermediate situations).

In the COCOMO intermediate model the cost factors have functions shaped in three possible ways: decreasing, increasing, and U-shaped. A direct calibration of the COCOMO cost factors by means of the linear regression method (the same Boehm suggests for the a and b coefficients) would produce (even on the COCOMO projects data base) non-monotonic and non U-shaped functions. This results would be very undesirable.

For the productivity model we extended a little (for generality) the set of possible productivity factors functions: increasing, decreasing, and parabolic (both U-shaped and \( \cap \)-shaped). Let be \( x \) the value of the productivity index, we express the productivity factor function as the exponential formula:

\[
P_{F_i}(x) = k_{i,1}^{x} \cdot k_{i,2}^{x}
\]

where \( k_i \) and \( k_j \) depends on the particular productivity factor.

The analytical study of this function shows that given any values for positive \( k_i \), \( k_j \) and positive \( x \) (which is always the case) it has always one of the desired shapes.

On this formula we may now impose a suitable constraint: we always impose that \( CD(\text{medium}) = 1.00 \). This is a sort of normalization factor. We found no needs to disobey to this normalization because we did not know in detail the projects and were not so expert in expecting the productivity factors behaviours. The values that Boehm assigned to the intermediate COCOMO cost factors for the most part also obey to this normalization. \( P_{F_i}(x) = k_{i,1}^{x} \cdot k_{i,2}^{x} \)

With this imposed normalization the formula for each productivity factor becomes:

This formulation has produced a very good result: \( \text{ARE}(25) = 92\% \) (percentage of
Desiderata for a good calibration

- Ability to use the original model (not a transformation)
- Do not be too influenced by outliers
- Multiple Objectives optimization
- Ability to find the near-global optimum
- Ability to impose the shape of CDs only when needed

12th COCOMO Forum '97
Typical objectives in calibration

Accuracy

\[
\max ARE(25) = \frac{1}{n} \sum_{i=1}^{n} a_i = \begin{cases} 
1 & \text{if } \frac{\text{EST}_i - \text{ACT}_i}{\text{ACT}_i} \leq 0.25 \\
0 & \text{if } \frac{\text{EST}_i - \text{ACT}_i}{\text{ACT}_i} > 0.25
\end{cases}
\]

Stability

Average Absolute Relative Error:

\[
\min AARE = \frac{1}{n} \sum_{i=1}^{n} \left| \frac{\text{EST}_i - \text{ACT}_i}{\text{ACT}_i} \right|
\]

Standard Error:

\[
\min E_s = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (\text{EST}_i - \text{ACT}_i)^2}
\]

Nonbias

Average Relative Error:

\[
\min |RE| = \frac{1}{n} \sum_{i=1}^{n} \left| \frac{\text{EST}_i - \text{ACT}_i}{\text{ACT}_i} \right|
\]

Following the approach of Myiazaki the goodness of fit for a software estimation model can be expressed by the following three characteristics:

1. Accuracy. The model should be able to accurately estimate the actual values of N systems (as many as possible). An example of this if the rate of the systems of which absolute relative errors are < 0.25 (ARE25). This values should be maximized.

2. Stability. The model should be able to stably estimate the actual values of the N systems. Suppose the model can accurately estimate most of the systems, but stability may not be high if the model very poorly estimates the rest of the systems. Examples of this are the average absolute relative error (AARE) and the standard error (\(E_s\)). Both should be minimized.

3. Nonbias. The model should have no tendency of overestimation or underestimation. Such a model is at least technically sound. An example is the average relative error that should be as close as possible to zero (corresponds to minimizing the absolute value).
Genetic Algorithms (GA) are local search methods well fit for optimization of several kind of functions. They are different from other mathematical approaches because they do not search a solution to the given problem using mathematical proceedings but making a population of solutions evolve towards better ones.

These methods have been formulated from the mechanics of natural selection and natural genetics. They combine survival of the fittest among string structures with a structured yet randomized information exchange to form a search algorithm with some of the innovative flair of human search.

Here we can see how the evolution exploited by GA is achieved. The starting point is a generic population where each individual/solution has been evaluated with a fitness function (our objective function). The individuals are encoded in strings representing in some way their DNA information.

The next step is the creation of an intermediate population (the temporary population in the slide), also called mating pool, which is composed by a subset of the solutions of the initial population. The solutions taken from the initial population are randomly selected according to the fitness function (solutions with a higher fitness are more likely to be selected).

When the mating pool is complete it is possible to start the creation of a new population. Pairs of solutions are extracted from this pool until it is empty. The 2 individuals of each pair (the parents) are then combined to create a completely new pair of solutions (the children), which are inserted into the new population. The combination of the parents is done by means of two operators: cross-over (the DNA of a new individual is created by mixing the DNA of the parents; this is necessary to preserve in the new population the fittest information) and mutation (the process by which a gene is casually altered; this is necessary to explore some new part of the solution space which may contain good solutions).

When the new population is completed, it can be evaluated (by the fitness function) and the process is ready to start again. The entire process will have virtually no end. When applied for searching a solution to a difficult problem a GA will be made to stop either a sufficiently good solution has been found or the algorithm has gone on for an established number of generations.
This is a simple example where we apply the GA approach to find the value of \( x \) (in the range 0 - 31) for which the objective function \( f(x) = x^2 \) is maximum. In the slide are represented only the first and second generations of the solutions.

Every individual of the population is a binary code of a possible solution. In this case is a binary code of the unsigned integer representing \( x \). There are only 4 strings in the current population. The current population, which is the first population for the example, is randomly generated. In the third column of the first table is reported the numeric equivalence of each string in the current population. In the fourth column is computed the fitness for every individual (in this case the fitness is the function for which we want to find the maximum). As you can see the second string is the best of the first population.

In order to build the temporary population we make a random selection from the first generation according to the fitness function. The result of this selection is indicated in the last column of the first table: string 1 and string 4 are selected once, string 2 (which has best fitness) is selected twice, string 3 is not selected. The temporary population resulting from this selection is reported in the second column of the second table.

To obtain the new population for each individual of the temporary population is associated the 2nd parent (random) reported in the third column. That means that string 1 is combined with string 2 and string 3 with string 4. For each pair of parents a cut point is selected. From each pair of parents a pair of children is created just exchanging the two portions of bits around the cut point (cross-over). So string 1 of the new population is created from the first 4 bits of string 1 (temp. pop.) and the last bit of string 2 (temp. pop.) and so on.

The new population so obtained represents a set of new solutions. Among these the third string is the best so for obtained.

In this example the mutation operator has not been applied. It would mean that in the new population some random change of bits may take place.
Our first trials with classical GAs led to frustrating results, so we decided to look for other techniques (in the GA family) best fit for our problems. We resorted to using Differential Evolution (DE), an algorithm of the GA family, particularly fit for numerical optimization. Like every GA it combines solutions of the current population to create new, and better, solutions for the next population.

The basic idea under DE is that a binary encoding of solutions for numerical optimization sometime looses the adjacency information of neighboring points. For example if x of the previous example is 15 is represented with ‘01111’, and if x is 16 is ‘10000’: a very different and distant string!

In DE the way to restore the adjacency of neighboring points is to use addition as a mutation operator. Under addition, 16 becomes 15 simply by adding -1. Now the odds of a given transition depend only on the arithmetic difference between the numbers involved and not the bit patterns that are used to encode them.

Adopting addition as a mutation operator to restore the adjacency of nearby points is not, however, a panacea. Once the switch from logical to arithmetic operators is made, the fundamental question concerning mutation is no longer which bits to flip, but rather how much to add. The simple adaptive scheme used by DE ensures that these mutation increments are automatically scaled to the correct magnitude.
Here we can see how DE works. First of all it picks the i-th element of the population; eventually the algorithm will pick up every individual. This element is called target vector and will be used as the first parent of a new solution.

The second parent is created using the following expression $X_t = X_c + F (X_a - X_b)$ where $X_a$, $X_b$, and $X_c$ are 3 elements selected at random and $F$ is a control parameter of the algorithm. This second parent is a randomly chosen population vector to which a weighted random difference vector has been added (a sort of noisy random vector).

The algorithm then recombines the 2 parents to create a new element, called trial vector, which is evaluated within the fitness function. This recombination is controlled by a nonuniform crossover operation that determines which trial vector parameters are inherited from which parent.

The trial and the target vector are then compared, and only the best one, according to their fitness value, is copied into the next population (will replace the target as the i-th population element in the next generation).

This procedure is replicated for every solution of the initial population, so in turn, each element is selected to be the target vector (and only once).
Each individual is represented as a vector of real numbers, where each number is a coefficient of the COCOMO model: the a, b and every CDs (or a codification of them).

As mentioned earlier one of the desiderata for the recalibration of COCOMO CDs is that of assuring, by construction, that the CDs will have a good shape. No one would accept an oscillating behavior of CDs. E.g. if the RELY (reliability) of the final system is higher than also the effort will be higher. What will not be accepted here is for example an effort higher for some values of RELY, than lower, than higher again (i.e. oscillating behavior). The particular shape of the CDs will be deduced from the particular projects data base for which we want to recalibrate COCOMO.

In order to guarantee good shapes for CD we implemented two approaches.

In the first approach we used the function of the slide 8 from our previous work. We also developed an original implementation of the COCOMO CDs. This encoding has been developed to simplify the task of imposing constraints onto the shape of the CDs and has a great advantage: it leaves to the user the choice of the shape. Each user can choose if he wants a specific shape for each CD or leave this choice to the algorithm (and than it will choose the best fitting one).

In one of the experiments the sequence of values used by our GA was free of constraints; our coding could convert any shape into a shape that respects the constraints.

Another problem was the choice of a good objective function (the fitness). After some trials with the functions already considered in slide 10 we developed a multi-objectives function. Our purpose was that of creating a function that could maximize the predictive capacity of the recalibrated model, while maintaining low the error committed by the model.
Here we can see an example of how our CD coding scheme works. The algorithm examines, one by one, the values of the coefficients of each CD. Each time a new value must be added, the selected shape is checked. If the new value does not violate the constraints, it is added to the CD as is. If the constraints are violated (the shape has an oscillating behavior), the proposed coefficient undergoes a transformation that converts its value into an acceptable one. This transformation calculates the new good value as a function of the bad one and of the last accepted coefficient of the CD, and has been created to ensure the universality of this transformation.
This slide summarizes some of the results we obtained during our experimentations. This table shows the values for the evaluation criteria (our objectives) and the global fitness (with respect to the Differential Evolution DE algorithm) for different calibration of the COCOMO model.

The first columns are the values for AARE (absolute average relative error). For this criterion we prefer the smallest values. The second column reports the ARE(25) (percentage of systems which are estimated within a relative error of 25%). The third column is the standard error. The last column is the value assumed by are fitness function (the lower the better).

The first row presents the values assumed for the original calibration of COCOMO (from Boehm). As you can see the ARE(25) value is 74.6%. In the literature is always asserted that ARE(25) is 68%. We recomputed these values on the 63 projects reported by Boehm’s book and found out some discrepancies on the estimated value for some projects.

The Gulezian recalibration performs well on AARE and ARE(25) but is the worst on E_. Its evaluation on the DE fitness is also not so good.

The calibration by Paschetta ’96 is very good for AARE, is the best for ARE(25) (92%!) but is the second worst on E_. Its overall fitness has also a low value.

The first calibration we performed with classical genetic algorithm (GA) was very frustrating, as we said: is the worst on AARE, the worst on ARE(25) and is very bad on E_. The DE fitness is consequently the worst.

All the calibrations with differential evolution (DE) performed very well with respect to all the 3 basic criteria and have so the best fitness.

"DE exp CD" is the application of differential evolution using the exponential formula for cost drivers we already used in the model Paschetta ’96.

The second application of DE, which is the best of all, does not impose any shape to the cost drivers, it only assures the the final shape is a non-oscillating one (i.e. a good shape). Having such a freedom the algorithm manages to make a better optimization. The complete set of coefficients is reported in the last slide (# 20).

In the last row there is the experiment done with differential evolution and imposing the same CD shape as those of original COCOMO. The results are also very good and better that those obtained by the other approaches using the same CD shapes.
CONCLUSIONS

- Best calibration of COCOMO
  - with respect to different criteria
- Possibility of customizing the objective function
- Possibility of customizing the CD shapes (or ask the algorithm to find the best fitting one)
- Applicability to COCOMO II and other models
- Large computing time (big drawback)
References

- Miyazaki Y., Terakado M., Ozami K., “Robust regression for developing software estimation models”, in JSS 27, 1994
### Appendix: Coefficient Tables

<table>
<thead>
<tr>
<th>Development mode</th>
<th>Organic</th>
<th>Semi-structured</th>
<th>Embedded</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1.05</td>
<td>3.70</td>
<td>2.65</td>
</tr>
<tr>
<td>b</td>
<td>1.29</td>
<td>0.93</td>
<td>1.08</td>
</tr>
</tbody>
</table>

**Calibration with DE without imposing CD shapes**

<table>
<thead>
<tr>
<th>Coefficient</th>
<th>Very Low</th>
<th>Low</th>
<th>Medium</th>
<th>High</th>
<th>Very High</th>
<th>Error (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RELY</td>
<td>0.74</td>
<td>0.78</td>
<td>1.18</td>
<td>0.78</td>
<td>1.37</td>
<td>1.42</td>
</tr>
<tr>
<td>DATA</td>
<td>0.84</td>
<td>1.11</td>
<td>1.38</td>
<td>0.79</td>
<td>1.60</td>
<td>1.42</td>
</tr>
<tr>
<td>EPIK</td>
<td>0.91</td>
<td>1.28</td>
<td>1.59</td>
<td>0.89</td>
<td>1.90</td>
<td>1.63</td>
</tr>
<tr>
<td>TIME</td>
<td>1.00</td>
<td>1.29</td>
<td>1.76</td>
<td>1.29</td>
<td>2.09</td>
<td>1.76</td>
</tr>
<tr>
<td>SSTD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VINI</td>
<td>0.28</td>
<td>0.33</td>
<td>0.62</td>
<td>0.28</td>
<td>0.62</td>
<td>0.33</td>
</tr>
<tr>
<td>VURN</td>
<td>0.62</td>
<td>0.82</td>
<td>1.07</td>
<td>0.62</td>
<td>1.07</td>
<td>0.82</td>
</tr>
<tr>
<td>KALP</td>
<td>1.16</td>
<td>1.21</td>
<td>1.91</td>
<td>1.16</td>
<td>1.91</td>
<td>1.21</td>
</tr>
<tr>
<td>AEXP</td>
<td>1.41</td>
<td>1.51</td>
<td>1.51</td>
<td>1.41</td>
<td>1.51</td>
<td>1.51</td>
</tr>
<tr>
<td>PEAP</td>
<td>1.71</td>
<td>1.71</td>
<td>1.71</td>
<td>1.71</td>
<td>1.71</td>
<td>1.71</td>
</tr>
<tr>
<td>VLSF</td>
<td>0.62</td>
<td>0.62</td>
<td>0.62</td>
<td>0.62</td>
<td>0.62</td>
<td>0.62</td>
</tr>
<tr>
<td>LEVF</td>
<td>1.79</td>
<td>1.79</td>
<td>1.79</td>
<td>1.79</td>
<td>1.79</td>
<td>1.79</td>
</tr>
<tr>
<td>NLIN</td>
<td>1.11</td>
<td>1.32</td>
<td>1.52</td>
<td>1.11</td>
<td>1.52</td>
<td>1.32</td>
</tr>
<tr>
<td>ScAF</td>
<td>0.94</td>
<td>0.94</td>
<td>0.94</td>
<td>0.94</td>
<td>0.94</td>
<td>0.94</td>
</tr>
<tr>
<td>NLCD</td>
<td>1.37</td>
<td>1.50</td>
<td>2.00</td>
<td>1.37</td>
<td>2.00</td>
<td>1.50</td>
</tr>
<tr>
<td>RINGL</td>
<td>0.85</td>
<td>1.07</td>
<td>1.53</td>
<td>0.85</td>
<td>1.53</td>
<td>1.07</td>
</tr>
</tbody>
</table>

**12th COCOMO Forum '97**

AARE = 0.119  
ARE(25) = 84.13%  
$E_s = 19.67$