Effort estimation using Search-Based Software Engineering

Ioana-Gabriela Chelaru

December 2021

This paper represents a dive into the literature, presenting three different approaches of the problem. The first section will give the reader a general understanding of the subject, explaining the concept of effort estimation, while the second section will tackle the connection between effort estimation and the search-based software engineering field. We will discuss three models, namely an automatically transformed linear model, created with the intention to become a a suitable baseline model for comparison against SEE methods and having the advantage that it can be used with mixed numeric and categorical data and requires no parameter tuning. It is better referred in the literature as a *state of the art*, the OIL project, which is a lightweight adaptation of the previously mentioned one, and, finally, bi-objective effort estimation algorithm that combines Confidence Interval Analysis and assessment of Mean Absolute Error. In conclusion, search-based software engineering should be a standard component when looking at project management problems and estimations.

Keywords: Effort Estimation; SBSE; Predictive Modeling; Baseline Model; Project Management

1 Introduction

Software Project Management includes several activities critical for the success of a project (e.g. effort estimation, cost estimation, project planning, quality management). These activities often involve finding a suitable balance between competing and potentially conflicting goals. For example, planning a project schedule requires to minimise the project duration and the project cost, and to maximise the product quality. Many of these problems are essentially optimization questions characterised by competing goals/constraints and with a bewilderingly large set of possible choices. The competitiveness of software organisations depends on their ability to accurately predict the effort required for developing software systems; both over- or under-estimates can negatively affect the outcome of software projects [2]. Effort estimation is a critical activity for planning and monitoring software project development and for delivering the product on time and within budget.

Several algorithmic approaches have been proposed in literature to support software engineers in improving the accuracy of their estimations by producing a point estimate of the effort required to develop a new project, but now we are interested in the influence that search-based techniques may have in this matter.

2 Effort estimation and SBSE

Software effort estimation is the process of predicting the most realistic amount of human effort required to plan, design, and develop a software project based on the information collected, being in fact a core activity in all software processes and development life-cycles. Search-Based Software Engineering seeks to reformulate software engineering problems as search-based optimisation problems and applies a variety of meta-heuristics based on local and global search to solve them, such as Hill Climbing, Tabu Search, and Genetic Algorithms. As proven in [10], SBSE has great applicability and performance potential when applied for software effort estimation problems.

Historically, the key objective of software effort estimation has been to produce a robust, accurate predictive cost model, but this is a challenging objective because of the complex nature of software development and the diversity of software projects, meaning that no single method will likely be the best for all project types. A range of increasingly complex methods has been considered in the past decades for the prediction of effort, often with mixed and contradictory results. This is due to the fact that datasets are difficult to obtain and often bias, initial data preparation methods may have removed outlier examples or explanatory variables, therefore the comparative assessment of effort prediction methods has become a common approach when considering how best to predict effort over a range of project types. Unfortunately, these assessments use a variety of sampling methods and error measurements, making a comparison with other work difficult.

MLR	LR & ANNs	ANNs	Pareto ensemble of ANNs	ANNs & case-based reasoning	Analogy-based approaches
with appropriate transformations	similar performance to MLR, but not always	outperformed MLR	best performance compared with a variety of methods	performed better than MLR	better results than MLR, ANNs, simple decision trees, and stepwise regression
[3]	[4]	[7]	[5]	[9]	[1]

Table 1: Different approaches used in empirical studies

Table 1 illustrates different empirical approaches found in independent research papers, and upon closer inspection, it is clear that the results are contradictory, further proving that there is no method that has both high performance and large applicability over different datasets.

3 ATLM

The goal of [6] was to develop a baseline model that would define a meaningful point of reference, in order to be both useful and widely used in SEE, and, eventually, become a suitable baseline model for comparison against SEE methods.

The multiple linear regression implements a linear regression in R, and the form considered is $y_i = \beta_0 + \beta_{xi} + \beta_{2x2i} + .. + \beta_{nxni} + \varepsilon_i$,where y is the quantitative response variable, x is the explanatory variable, and β is determined using a least-squares estimator.

The ATLM will assess the suitability of log and square-root transformations of these variables based on the underlying distribution of the data. The more complex consideration of model residuals or stepwise regression which would typically be examined when developing a linear model will not be considered since ATLM is *not intended to be the best model*, and these added complexities often involve parameter settings.

```
ALGORITHM 1: Linear Model Prediction
Input: formula, training, test
Output: Predictions for test data

transforms      ← calculate.transforms (training)
trans.training  ← apply.transforms (transforms,training)
                  # Apply transformations
trans.test      ← apply.transforms (transforms,test)
lm.see          ← lm (formula,trans.training)
                  # Determine linear model
predictions     ← predict (lm.see,newdata = trans.test)
                  # Predict response
return (invert.predictions(transforms,predictions))
                  # Return untransformed predictions
```

(a) Linear Model Prediction

```
ALGORITHM 2: Calculate.Transforms
Input: data, transformfns ={none,log,sqrt},
invtransformfns ={none, exp, sqr}
Output: Transformation table

transformationTable ← {}
For each variable in data
# for the response and each explanatory variable
  If variable is a factor
  #do nothing for categorical factors
        transfn ← none
        invfn ← none
  Else
  # determine transformation
  skewness ← {}
    For each fn in transformfns
    # apply each transformation to each variable
        skewdata ← fn (data)
        # apply transformation function
        skewness ← skewness ∪ skew (skewdata)
        # record skewness
    best ← min (skewness)
    # Best function has lowest skewed data
    transfn ← transformfns (best)
    invfn ← invtransformfns (best)
  transformationTable ← transformationTable ∪
  {variable, transfn, invfn}
return(transformationTable)
```

(b) Calculate transformations

Figure 1: Base algorithms used by ATLM

The general approach is shown in Figure a; the transformations are calculated for the response and explanatory variables using the training data, applied when needed, based on skewness, to the training and test data, and a linear model constructed and used to predict the test data, with the final predictions inverted to produce the original scaling of the estimated response. These predictions can then be used to calculate the model error. The transformation step is calculated by comparing the skewness [6] for each variable when a logarithm, square-root, and no transformation are applied, as shown in Figure b. The transformation that results in the least skewed data for each variable is selected and used when constructing the linear model and predicting effort, keeping in mind that a variable that is categorical is ignored in this step. The appropriate inverse transform of the predictions is applied so that the model results can be meaningfully compared to the untransformed test data.

Overlall, ATLM is simple yet performs well over a range of different project types. In addition, it may be used with mixed numeric and categorical data, therefore it requires no parameter tuning. It is also deterministic, meaning that results obtained are amenable to replication.

4 OIL

Optimized learning is a novel configuration tool for effort estimation based on differential evolution. When tested on over 900 software projects, it significantly improved effort estimations, after exploring just a few dozen configurations. [10] Furthermore, the results are far better than two methods in widespread use: estimation-via-analogy and The Automatically Transformed Linear Model discussed before.

OIL was implemented as a layered architecture:

- At the lowest library layer uses Python Scikit-Learn.
- Above that, a utility layer containing all the algorithms missing from the previous one
- Higher up, a modeling layer specifying a feature map of data mining options.
- Finally, at the top-most optimizer layer, there is some evolutionary optimizer that makes decisions across the feature map. An automatic mapper facility then links those decisions down to the lower layers to run the selected algorithms.

It is good practice to benchmark new methods against a variety of different approaches, therefore OIL uses the following algorithms: Analogy-based Estimation, the Automatically Transformed Linear Model presented in Section 3, Differential Evolution and Random Choice, and performed a X-fold cross validation for nine data sets. To apply this, datasets are partitioned into X sets, and then for each set OIL considered it as a testing set and the remaining observations as training set, all the process being repeated 20 times, each time with different random seeds.

This is a very *CPU-lite* method that significantly outperforms standard effort estimation methods, meaning that the estimates have statistically smaller errors than standard methods, and the commissioning time for that estimator is very fast: median runtime for ten-way cross-validation is just six minutes on a standard desktop machine.

5 Multi-objective software effort estimation

Introduces a bi-objective effort estimation algorithm that combines the assessment of Mean Absolute Error with the novelty of incorporating of confidence intervals to guide a multi-objective evolutionary algorithm.

The proposed model encodes as a Genetic Algorithm using an expression syntax tree and randomly choosing the coefficients, while the factors values depend on the training data and do not change during the evolution process; the initial population was generated by building 100 random trees of fixed depth. To evaluate the fitness of each chromosome they employed a multi-objective function to simultaneously minimise the estimates accuracy and the estimates distribution uncertainty. Crossover and mutation operators were defined to preserve well-formed equations in all offspring [8], here being used a single point crossover which randomly selects the same point in each predictive model expression tree and swaps the subtrees corresponding to the selected node. Since both trees are cut at the same point, the trees resulting after the swapping have the same depth as compared to those of parent trees. After that a mutation operator is being used to select a node of the tree and randomly change the associated value. The rates for crossover and mutation were fixed to 0.5 and 0.1, respectively, since in previous work recommended similar values. The evolutionary process terminates after 250 generations.

The algorithm was evaluated on three different alternative formulations, baseline comparators and current *state-of-the-art* effort estimators applied to five real-world datasets from the PROMISE repository, involving 724 different software projects in total. [8] The results reveal that the algorithm outperforms the baseline, state-of-the-art and all three alternative formulations, statistically significantly and with large effect size over all five datasets. It also provides evidence that the algorithm creates a new state-of-the-art, which lies within currently claimed

industrial human-expert-based thresholds, thereby demonstrating that these findings have actionable conclusions for practicing software engineers.

6 Conclusions and further work

Considering all of the facts, it is clear that search-based software engineering has great potential in effort estimation since the results show that it is both necessary and simple to apply, but with so many available methods, it is now a matter of debate which one is best for a new data set. This decision also depends on what our resources are, whether we're looking for good performance or for a lightweight solution, the options and studies are increasing by day. Although there is a clear need for a solution that would have both a good performance and a general applicability, the literature does not provide, up until today, such an approach.

As to future work, I'd be interested in exploring the literature even deeper, testing out different estimators, and eventually developing an alternative approach that would aim good performance, while also considering an optimal use of computational resources.

References

[1] N.-H. Chiu and S.-J. Huang. The adjusted analogy-based software effort estimation based on similarity distances. *J. Syst. Softw. 80, 628–640.*, 2007.

[2] K. Cowing. Nasa to shut down checkout & launch control system. 2002.

[3] K. Dejaeger, W. Verbeke, D. Martens, and B. Baesens. Data mining techniques for software effort estimation: A comparative study. *IEEE Trans. Softw. Engin. 38, 375–397*, 2012.

[4] A. Heiat. Comparison of artificial neural network and regression models for estimating software development effort. *Inf. Softw. Technol. 44, 911–921.*, 2002.

[5] L. Minku and X. Yao. Software effort estimation as a multiobjective learning problem. *ACM Trans. Softw. Engin. Methodol. 22, 4.*, 2013.

[6] Whigham P.A., Owen C.A., and MacDonell S.G. A baseline model for software effort estimation. *ACM Transactions on Software Engineering and Methodology 24(3), pp.1-11 (Article 20). doi:10.1145/2738037*, 2015.

[7] H. Park and S. Baek. An empirical validation of a neural network model for software effort estimation. *Expert Syst. Appl. 35, 929–936.*, 2008.

[8] Federica Sarro, Alessio Petrozziello, and Mark Harman. Multi-objective software effort estimation. *IEEE/ACM 38th IEEE International Conference on Software Engineering*, May 2016.

[9] G. Wittig and G. Finnie. Estimating software development effort with connectionist models. *Inf. Softw. Technol. 39, 469–476.*, 1997.

[10] Tianpei Xia, Jianfeng Chen, George Mathew, Xipeng Shen, and Tim Menzies. Why software effort estimation needs sbse. *SSBSE'18*, 2018.