View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Scholarship@Western

Western University
Scholarship@Western

Electrical and Computer Engineering Publications | Electrical and Computer Engineering Department

12-2008

Calibrating Function Point Backfiring Conversion Ratios Using Neuro-Fuzzy Technique

Justin Wong
SAP Canada, justinplwong@gmail.com

Luiz Fernando Capretz
University of Western Ontario, lcapretz@uwo.ca

Danny Ho
NFA-Estimation, danny@nfa-estimation.com

Follow this and additional works at: https://ir.lib.uwo.ca/electricalpub


Part of the Software Engineering Commons

Citation of this paper:

@article{DBLP:journals/ijufks/WongHC08, author = {Justin Wong and Danny Ho and Luiz Fernando Capretz}, title = {Calibrating Function Point Backfiring Conversion Ratios Using Neuro-Fuzzy Technique}, journal = {International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems}, volume = {16}, number = {6}, year = {2008}, pages = {847-862}, ee = {http://dx.doi.org/10.1142/S0218488508005650}, bibsource = {DBLP, http://dblp.uni-trier.de} }

Calibrating Function Point Backfiring Conversion Ratios Using Neuro-Fuzzy Technique

Justin Wong[1], Danny Ho[2], Luiz Fernando Capretz[1]

[1]Department of Electrical & Computer Engineering
University of Western Ontario
London, Ontario, N6A 5B9, Canada
jwong343@uwo.ca, lcapretz@eng.uwo.ca

[2]NFA Estimation Inc.
London, Ontario, N6G 3A8, Canada
danny@nfa-estimation.com

Abstract

Software estimation is an important aspect in software development projects because poor estimations can lead to late delivery, cost overruns, and possibly project failure. Backfiring is a popular technique for sizing and predicting the volume of source code by converting the function point metric into source lines of code mathematically using conversion ratios. While this technique is popular and useful, there is a high margin of error in backfiring. This research introduces a new method to reduce that margin of error. Neural networks and fuzzy logic in software prediction models have been demonstrated in the past to have improved performance over traditional techniques. For this reason, a neuro-fuzzy approach is introduced to the backfiring technique to calibrate the conversion ratios. This paper presents the neuro-fuzzy calibration solution and compares the calibrated model against the default conversion ratios currently used by software practitioners.

Keywords: Backfiring, Software Estimation, Sizing, Function Point, Neuro-Fuzzy, Lines of Code

1. Introduction and Background

Software estimation is important in delivering successful software. Project estimation is used to determine the scope of a project. A manager determines the necessary development period and cost of projects. This step is vital to win project bids and helps to establish the extent of a project's success. Currently many estimation techniques have been developed such as: Constructive Cost Model (COCOMO), Putnam's Software Lifecycle Management (SLIM) and Function Point Analysis [19]. Currently, the latest technique of machine learning has been applied to these estimation methods.

In this study, neural network and fuzzy logic is used to predict the lines of code when the function point and programming language are known. The fundamental concepts of function point and backfiring estimation technique to predict the lines of code are explained. Fuzzy logic and neural network concepts are illustrated because such methods would be applied to improve

the accuracy of backfiring size estimates. Furthermore, related work of existing software estimation models that use neural networks and fuzzy logic are investigated.

The main objective of this study is to develop a prediction model that uses the backfiring approach of estimating lines of code. The estimation model is tested and compared against the original backfiring method. The threats to the validity of the approach and experiment are investigated and discussed.

1.1. Function Point

First introduced in the 1970s by Albrecht [2], the function point is a unit of measurement for determining the functional size of an information system. Function point analysis is a process that involves identifying major system components and classifying them as 'simple', 'average', or 'complex'. The unadjusted function points (UFP) are then calculated as shown in Table 1. The UFP is then adjusted for application and environment complexity through complexity adjustment factors (CAF), which can be found using the formula defined in (1). Finally, the adjusted function point (AFP) is calculated by multiplying the UFP and CAF [6].

Table 1 – Calculating the UFP

Function Type	Complexity			
	Simple	Average	Complex	Total
External Input	___ x 3	___ x 4	___ x 6	___
External Output	___ x 4	___ x 5	___ x 7	___
Logical internal file	___ x 7	___ x 10	___ x 15	___
External Interface File	___ x 5	___ x 7	___ x 10	___
External Inquiry	___ x 3	___ x 4	___ x 6	___
Total Unadjusted function points				___

$$CAF = 0.65 + 0.01N \quad (1)$$

N is the total degree of influence of the 14 characteristics. The degree of influence ranges from 0 to 5 [6].

1.2. Backfiring

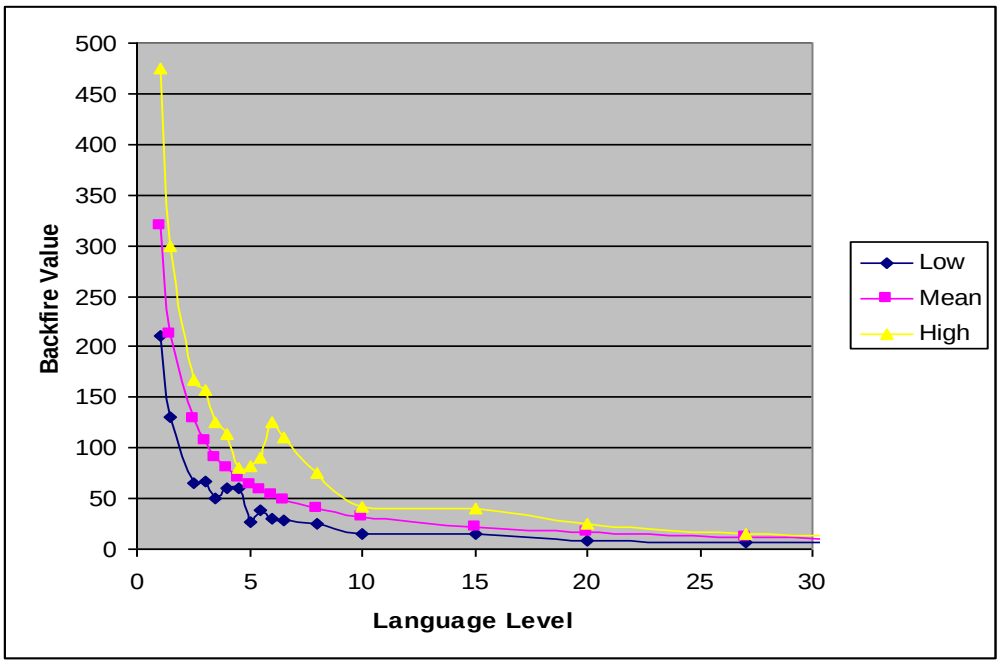
Experts have been using the term "high-level language" and "low-level language" for many years without precisely defining the phrases. Jones [10] classified programming languages by the number of statements they require for the implementation of one function point. Software Productivity Research (SPR) [17] annually publishes the conversion ratios of logical source-code statements to function points for many programming languages. Figure 1 illustrates the relationship of the conversion ratio, also known as source lines-of-code per function point (SLOC/FP), to the language level; the graph shows that as the language level increases, the conversion ratio decreases, thus, defining the phrases "high-level" and "low-level" languages.

The concept of backfiring is the conversion of function points to logical source-code statements to effectively sized source code. Source lines-of-code (SLOC) is still a very common metric used by the software industry to measure the size of an application. Backfiring can be

accomplished by multiplying the function point with the conversion ratios to obtain the SLOC. The conversion ratios can be used bilaterally mathematically; therefore, function points can be calculated from SLOC being divided by the conversion ratios [19].

While backfiring is very useful and simple, there is a high margin of error in converting SLOC data into function points [10]. This research introduces an approach to reducing this margin of error thus making the conversion ratios more reliable.

Figure 1 – Conversion ratios versus language level



1.3. Fuzzy Logic

Fuzzy logic is derived from the fuzzy set theory which uses linguistic terms or fuzzy set that represents and processes uncertainty. It is used to generate a mapping between input and output spaces. Fuzzy logic is made up of membership functions with values ranging from 0 to 1. Membership functions are used to describe linguistic terms such as low, medium and high [15].

Fuzzy logic is used in many software estimation models because it is not possible to develop a precise mathematical model of software development efforts and size [13]. Fuzzy logic was used to model the curve in Figure 1. The curve is modeled by breaking the curve into fuzzy sets based on the programming language data.

1.4. Neural Network

Neural networks are a system of weighted interconnected neurons which takes inputs into the network and produces an output function. The network also contains activation functions used to compute the inputs. The inputs may be passed through many layers of the network until it reaches the output layer [14].

A neural network can be trained to generate an appropriate output based on the input patterns. During the training phase, an input and target output are fed into the network. The resulting output is compared with the target output. A commonly used learning method is back-propagation. Back-propagation reduces the error between the target output and actual output by propagating the error from the output layer to the input layer and adjusting the weights between the neurons. The neural network's performance is affected by many different factors such as the number of hidden nodes, stopping criteria, and initial weight [14].

In this research, a neural network was used to calibrate the fuzzy sets. By calibrating the fuzzy sets, the programming language level versus statements per function point curve could be accurately modeled. This technique of calibration was similar to Huang et al. [7] proposed model of combining fuzzy logic and neural network for calibrating the COCOMO estimation technique.

1.5. Benchmark

Foss et al. [5] showed that when evaluating and comparing prediction models, Magnitude of Relative Error (MRE) should not be used. It has been demonstrated that using MRE does not prove that one model was particularly better than another because the results were very misleading. MRE favored underestimation and performed worst in small sized projects. The equation is defined in (2). However, this method of evaluation is currently still popular and commonly used in industry so it was used to evaluate to evaluate the experimental model.

$$MRE = \frac{|Actual - Predicted|}{Actual} \qquad (2)$$

Another method that was proposed for evaluating and comparing prediction models was Magnitude of error Relative to the Estimate (MER) [12]. The equation for calculating MER is defined in (3). MER was used to evaluate the proposed model and it was affected based on an individual's prediction. However, this was solved because the data used was published from SPR [17] for benchmarking. MER was encouraged to be used for evaluation, however it biased towards overestimation because the estimation was a divisor. Larger estimates tend to perform much better than small estimates.

$$MER = \frac{|Actual - Predicted|}{Predicted} \qquad (3)$$

Foss et al. concluded that Standard Deviation (SD), Residual Error Standard Deviation (RSD) and Logarithmic Standard Deviation (LSD) were good, consistent criteria [5]. The equation for SD is shown in (4). In the equation y_i represents the actual result, $\hat{y}_i$ represents the predicted result and n was the total number of projects points. The equation for RSD is defined in (5). In the equation y_i represents the actual result, $\hat{y}_i$ represents the predicted result, x_i represents the input and n was the total number of projects. The equation for LSD is defined in (6). In the equation y_i represents the actual result, $\hat{y}_i$ represents the predicted result, n was the total number of projects and the s^2 was an estimator of the variance of $\ln y_i - \ln \hat{y}_i$. In addition to MRE and MER, SD, RSD and LSD were used to for evaluation.

$$SD = \sqrt{\frac{\sum\left(y_i - \hat{y}_i\right)^2}{n-1}} \qquad (4)$$

where $\hat{y}_i$ is actual, $\hat{y}_i$ is predicted and n is number of projects

$$RSD = \sqrt{\frac{\sum\left(\frac{y_i - \hat{y}_i}{x_i}\right)^2}{n-1}} \qquad (5)$$

where $\hat{y}_i$ is actual, $\hat{y}_i$ is predicted, is x_i input and n is number of projects

$$LSD = \sqrt{\frac{\sum\left(\left(\ln y_i - \ln \hat{y}_i\right) - \left(-\frac{s^2}{2}\right)\right)^2}{n-1}} \qquad (6)$$

where y_i is actual, $\hat{y}_i$ is predicted, n is number of projects and s^2 is an estimator of the variance

Another criterion that was used to evaluate the prediction model was using a Prediction at Level (PRED). MRE less than 25%, 35%, 50% and 75% were used for PRED because other models used have used these criteria for evaluation.

2.0. Related Work

The literature in software estimation models was very extensive. The papers reviewed focused on neural network software estimation models.

Aggarwal et al. [1] presented using neural networks to estimate the lines of code when given the function points as input. Their estimation model used Bayesian Regularization to train the neural network. They investigated on various training algorithm to find the best results. The network furthermore took into account the maximum team size, function point standard and language (3rd generation language and 4th generation language). The shortcoming of the neural network was that it had a black-box design. Furthermore, it only took the generation language into account instead of the programming languages. The averages SLOC/FP between the 3rd and 4th generation languages are very large in range. The 3rd generation default language has 80 SLOC/FP, while the 4th generation default language has 20 SLOC/FP.

Jeffrey et al. [9] investigated an algorithm-based effort and management tool. They looked at the accuracy of lines of code as input, accuracy of the backfiring, and compare the organization's delivery rate. The paper proposed that generic tools need to be calibrated for individual organizations in order to be more accurate. In this research, the same idea of calibrating the generic technique was followed. The generic backfiring method was calibrated to improve the performance for a specific dataset.

Idri et al. [8] discussed about the shortcomings of the "black-box" models and investigated them. The paper used a standard feed-forward neural network with error propagation. The network was trained and the fuzzy rules were obtained. The paper discussed the interpretation of the "black-box" neural network. The paper influenced the experimental model's neural network design to be easily interpreted and understood for the calibrations of each language level.

Srinivasan et al. [18] compared between the CartX and Backpropagation learning methods to the traditional approaches of SLIM, COCOMO and function point. The paper showed the sensitivity of learning based on the data selection and representation. Because of the importance of data selection, all the mild outlier points from the experiment dataset were removed and the neural network's learning sensitivity was adjusted to avoid large changes when calibrating.

Huang et al. [7] introduced a neuro-fuzzy framework and applied it to COCOMO II. The neuro-fuzzy technique showed it can improve software estimation techniques by calibrating its parameters. In this study, it was demonstrated that a neural network could be used to calibrate fuzzy sets to improve performances for many different applications. Based on the flexibility of the technique, the neuro-fuzzy framework was applied to the backfiring technique.

3. Research Questions

In this study, the focus was to attempt to improve the precision of converting between the lines of source code and function point metric. The following set of research questions are to be answered:

1. How can the inverse curve relationship between lines of source statements per function point and programming language level be modeled precisely?
2. How can the conversion ratio be calibrated to a specific data set to improve the performance?
3. Is it possible to continue to calibrate the conversion ratios as more data become available?
4. How are missing data for certain programming languages addressed?

The model was trained and evaluated with International Software Benchmark Standards Group's (ISBSG) release 9 data suite [4]. The model's estimate was measured against SPR's programming language table [17].

4. Experimental approach

4.1 Fuzzy Parameters

The programming languages were broken into language levels which translated to the number of SLOC/FP. The relation between the language levels to the mean SLOC/FP was an inverse curve shown in Figure 1. The equation of the inverse curve found is defined in (7).

$y = 319.4x^{-0.997}$, where y is the SLOC/FP and x is the language level (7)

The approach was to group the language levels into various fuzzy levels to approximate the inverse curve. This was done by grouping all the programming languages together based on their

language levels because the SLOC/FP for each level was very close. In addition to this grouping criteria, language levels were grouped based on the number of data points available. For example, if there are many points for level 2.5, then a fuzzy set of level 1 is created. Table 2 shows the language levels being grouped into fuzzy levels to approximate the curve. The fuzzy levels were obtained based on the programming languages and data points available from ISBSG. The average SLOC/FP was obtained from the average of all the backfiring conversion values within a fuzzy level. This average value was used as the initial weight in the neural network and the initial peak of the fuzzy membership functions. This approach answered research question 1 by breaking down the curve into smaller pieces.

Table 2 – Fuzzy Level based on ISBSG research suite release 9

Fuzzy Level	Programming Language Level	Average Source statements per function point (SLOC/FP)
1	2.5	128
2	3	107
3	3.5	91
4	4	81
5	5	67
6	6	53
7	7	46
8	8	40
9	8.5	38
10	9	36
11	9.5	34
12	11	29
13	14	23
14	16	20
15	20	16
16	23	14
17	25	13
18	27	12
19	50	6

The fuzzy levels would change based on the data available. If more programming languages were to become available, more fuzzy levels could be added to model the curve more accurately. The fuzzy levels were flexible and can be added and modified, therefore it addressed research question 3.

4.2 Fuzzy Membership Functions

Figure 2 and Figure 3 illustrate the input and output fuzzy membership functions. A triangular function was used for both the input and output membership functions. The peak of the input triangle of each fuzzy level was the programming language level. The average SLOC/FP was the peak of the output membership functions. The peaks were obtained from Table 2. Each fuzzy level was directly referenced to a fuzzy output. For example, f1 referenced o1, f2 referenced o2 and so on. Figure 4 shows the fuzzy rule block. The “AND” and “activation function” used the

minimum function for the rules. For defuzzifying, the maximum accumulation method and "Center of Gravity" method were used. The input membership functions would cover missing language levels which solved the 4th research question.

Figure 2 – Fuzzy membership of the language levels

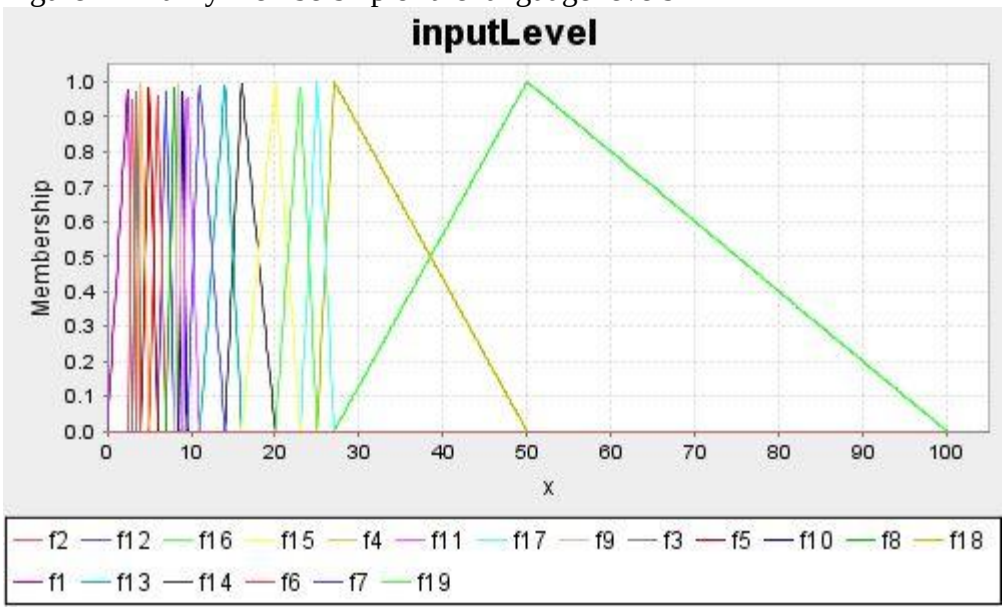


Figure 3 – Fuzzy membership functions of the output SLOC for each fuzzy level

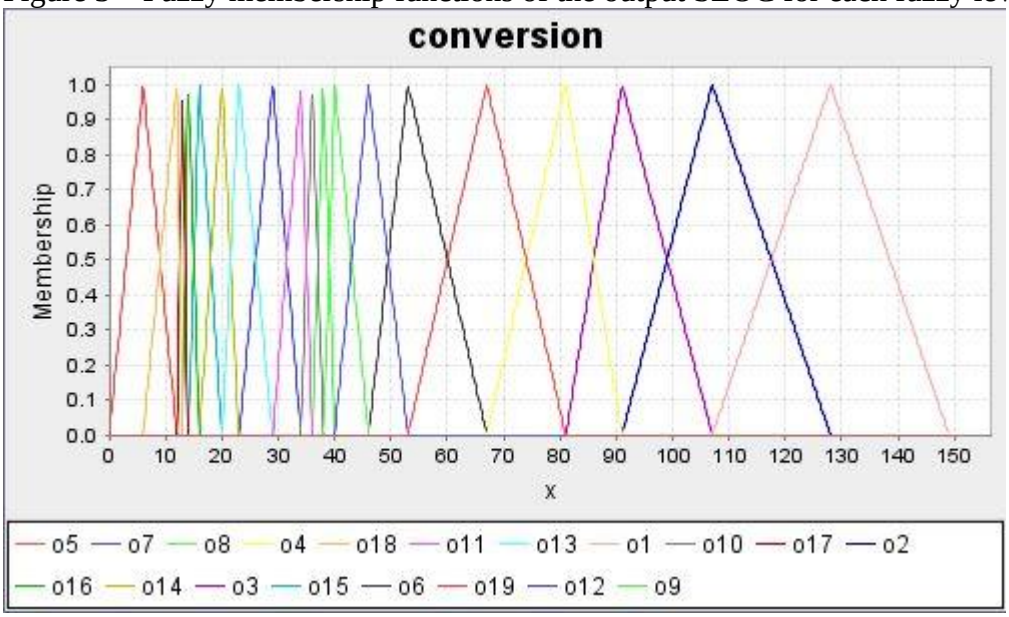


Figure 4 – Fuzzy rule block

RULE 1 : IF inputLevel IS f1 THEN conversion IS o1;
RULE 2 : IF inputLevel IS f2 THEN conversion IS o2;
RULE 3 : IF inputLevel IS f3 THEN conversion IS o3;
RULE 4 : IF inputLevel IS f4 THEN conversion IS o4;

RULE 5 : IF inputLevel IS f5 THEN conversion IS o5;
RULE 6 : IF inputLevel IS f6 THEN conversion IS o6;
RULE 7 : IF inputLevel IS f7 THEN conversion IS o7;
RULE 8 : IF inputLevel IS f8 THEN conversion IS o8;
RULE 9 : IF inputLevel IS f9 THEN conversion IS o9;
RULE 10 : IF inputLevel IS f10 THEN conversion IS o10;
RULE 11 : IF inputLevel IS f11 THEN conversion IS o11;
RULE 12 : IF inputLevel IS f12 THEN conversion IS o12;
RULE 13 : IF inputLevel IS f13 THEN conversion IS o13;
RULE 14 : IF inputLevel IS f14 THEN conversion IS o14;
RULE 15 : IF inputLevel IS f15 THEN conversion IS o15;
RULE 16 : IF inputLevel IS f16 THEN conversion IS o16;
RULE 17 : IF inputLevel IS f17 THEN conversion IS o17;
RULE 18 : IF inputLevel IS f18 THEN conversion IS o18;
RULE 19 : IF inputLevel IS f19 THEN conversion IS o19;

4.3 Calibrating Fuzzy Sets with Neural Network

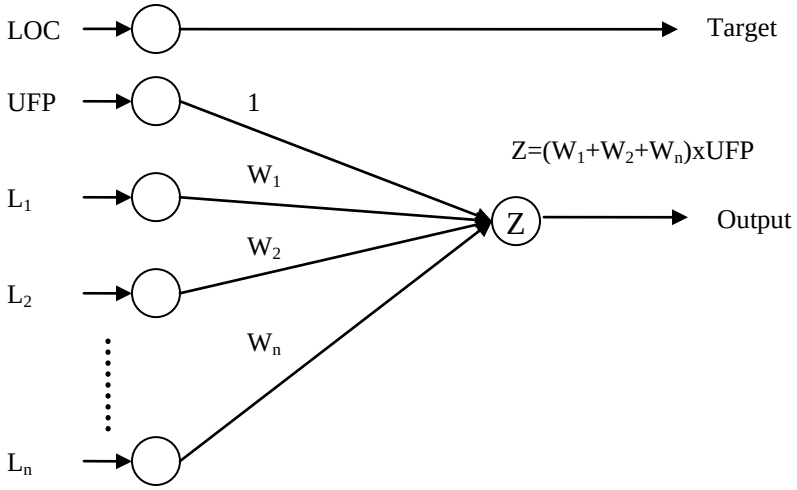
The neural network was used to calibrate the average source statements per function point for each fuzzy level. The input to the neural network was the SLOC, the unadjusted function point (UFP) and the language level. The SLOC was used as the target during training and the UFP was used for both training and simulation. The language level inputs were initially processed into fuzzy language levels. Figure 5 shows the layout of the neural network. The neural network was designed to be easily interpreted so that it avoids being a “black-box” model.

The L_1 to L_n were the fuzzy language levels input. The fuzzy language level inputs were binary. When a language level was fed into the network, the input was in a form of a matrix and only contains one 1 entry. For example, for language level 4, it would be represented as [0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0] based on the proposed fuzzy levels.

The activation function was simply multiplying the UFP with the weight of the fuzzy language. The weights were initialized initially with the values in Table 1. During training, once the output was obtained, it was compared with the target result. The difference between the actual result and predicted result was propagated back to the input layer. The weights were then adjusted based on the error.

The neural network approach solved the 1st and 3rd research questions because it attempted to minimize the error of each fuzzy level to accurately model the curve. The neural network could be used again to calibrate the weights when more data becomes available.

Figure 5 – Neural Network Design



4.4 Constraints

Every language level contained a low, median and high conversion value, therefore, each fuzzy level was constrained between lowest and highest conversion value. In cases where the fuzzy level contained more than one language level, the largest conversion value among the languages would be the upper bound of the SLOC/FP value. For the lower bound, the smallest conversion value would be used. A monotonic constraint was not used because the programming languages could overlap depending on the low and high range of the conversion value.

5 Results and Evaluation

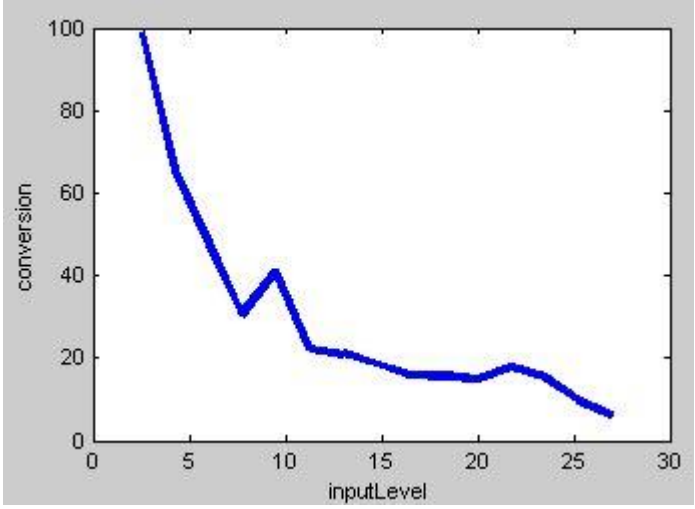
Table 3 shows the calibrated values obtained for the fuzzy language levels when the network was trained. The output triangular membership function's peaks were changed to the new calibrated values in Table 3. The results showed that the programming language levels 2.5 had a lower SLOC/FP. The language levels near 10 have a higher than mean SLOC/FP and this was due to the fact that the data in those areas are not in the mean range of the original language level. Figure 6 shows the surface view of the fuzzy rules and membership functions.

Table 3 –Calibrated Values

Fuzzy Level	Calibrated Values
1	95.6
2	99.0
3	87.7
4	82.9
5	67.0
6	49.2
7	48.6
8	24.9
9	37.5
10	46.2
11	37.4

12	39.5
13	22.1
14	22.0
15	16.3
16	14.3
17	17.0
18	13.9
19	6.3

Figure 6 – Surface View



5.1 Results

261 data points were used to compare between the calibrated and the original conversion ratios. The data points contained the parameters required in the ISBSG [4] repository data. Furthermore, the mild outlier points were removed. Table 4 shows the Mean Magnitude of Relative Error (MMRE) and Mean Magnitude of error Relative to Estimate (MMER) comparison between calibrated and non-calibrated values. Overall the MMER improves by 9% when calibrated and MMRE improved by 28%. For both MMRE and MMER, some levels had a negative improvement. After close analysis of the negative improvement levels, it was found that for the MER for level 2.5, there was an outlier point which had MER of 4. For level 8, there was an MER of 9 in one of the outlier point. For the negative improvement for MRE, level 7 contained one outlier point which had an MRE of over 3 and level 16 contained an outlier point that had an MRE of 4.

Table 4 – MMER and MMRE

	Original		**Calibrated**		**Improvement (%)**	
Level	**MMER**	**MMRE**	**MMER**	**MMRE**	**MMER**	**MMRE**
2.5	0.58	2.75	0.63	1.94	-8.05	29.29
3	0.54	2.02	0.53	1.82	1.71	9.87
3.5	0.61	3.50	0.60	3.34	0.37	4.35
4	0.42	0.31	0.41	0.32	1.91	-5.53

5	0.18	0.16	0.13	0.12	28.91	25.57
6	0.87	1.50	0.94	1.40	-7.94	7.02
7	0.74	0.51	0.69	0.52	7.39	-2.47
8	0.66	18.96	1.05	11.83	-60.14	37.59
8.5	0.34	0.60	0.34	0.59	-0.04	2.00
9	1.50	0.50	0.94	0.36	36.94	28.43
9.5	1.43	0.37	1.27	0.37	11.59	0.48
11	0.90	0.41	0.58	0.44	35.77	-7.57
14	0.48	0.43	0.50	0.42	-4.37	3.89
16	1.69	1.21	1.53	1.33	9.36	-10.04
20	3.79	0.79	3.69	0.79	2.52	0.54
23	2.25	0.84	2.19	0.84	2.69	-0.43
25	12.02	0.87	8.96	0.83	25.49	4.48
27	2.35	0.66	1.90	0.60	19.35	8.27
50	0.95	0.49	0.85	0.46	9.83	5.30
Overall	1.15	2.80	1.04	2.03	**9.43**	**27.64**

Standard Deviation (SD), Residual Error Standard Deviation (RSD) and Logarithmic Standard Deviation (LSD) were shown to be good criteria to evaluate prediction models [5]. Table 5 shows the comparison of SD, RSD and LSD between the original and calibrated model. The calibration model had an improvement of 11% for SD, 4.5% for RSD and 9.5% for LSD. Levels 5, 20 and 50 were not shown because they only have one data point. The results showed that the model was not bias towards underestimation of MRE and overestimation of MER.

Table 5 – SD, RSD, LSD

	Original			**Calibrated**		
Level	**SD**	**RSD**	**LSD**	**SD**	**RSD**	**LSD**
2.5	26647.40	94.55	0.96	22054.12	94.55	0.96
3	23158.63	62.22	0.78	21850.96	62.22	0.78
3.5	12490.09	34.80	0.91	13156.70	34.80	0.91
4	10248.58	62.01	0.49	10007.31	61.83	0.48
5	-	-	-	-	-	-
6	23229.44	70.71	1.04	22502.37	70.71	1.04
7	20681.88	45.22	0.66	21062.31	45.22	0.66
8	23751.63	56.45	1.42	19986.48	56.45	1.42
8.5	3572.90	11.73	0.38	3568.41	11.73	0.38
9	22593.76	56.48	0.55	23514.70	56.48	0.55
9.5	14847.13	60.80	0.73	14725.77	60.80	0.73
11	18157.18	32.44	0.58	17876.38	32.44	0.58
14	5057.90	13.91	0.50	4971.45	13.91	0.50
16	19989.26	73.32	1.22	19961.50	73.32	1.22
20	-	-	-	-	-	-
23	3226.34	33.84	1.00	3236.28	33.84	1.00
25	25778.59	106.84	0.78	25741.47	106.84	0.78
27	6329.42	14.40	0.41	6081.66	14.40	0.41

50	-	-	-	-	-	-
Overall	22811.87	69.55	1.30	20270.68	66.41	1.17
Overall Improvement (%)				**11.14**	**4.53**	**9.49**

The PRED results, in table 6, showed a small improvement for PRED<25%. The improvements were larger when PRED was 35, 50 and 75 percent. The results showed an objective improvement in all the different evaluation criteria. The results validate the experimental approach in solving the research questions and improving the backfiring process.

Table 6 – PRED Validation Results

PRED < (%)	**Original (%)**	**Calibrated (%)**	**Improvement (%)**
25	29.5	30.3	**2.8**
35	35.7	40.2	**12.8**
50	48.5	51.9	**6.8**
75	65.6	71.4	**8.9**

6 Threats to Validity

There were threats to validity in this experiment. Certain language levels contained limited project data. Program language level 5, 20 and 50 only contained one data point. If there were more data points available in those language levels, the results may have been different. However, other languages that contained sufficient data points show improvement and have similar behavior.

In the experiment, there were no data points for certain language levels which may not have accurately modeled the SLOC/FP versus language level curve. For example, there was no project data for the assembly programming language and high programming language levels between 27 and 50.

Another threat to validity was that other parameters may exist, which could affect the programming language's SLOC/FP. For example, specific general characteristics in function point analysis may affect the final estimate on lines of code. It was shown by Reifer [16] that in different application domain, the size and cost of the applications differ. Angelis et al. [3] showed a software cost estimation model based on attributes such as organization type, business type, development platform and development type. Therefore, these factors may have also affected the source lines of code per function point.

7 Conclusion

Backfiring had been used many years for sizing projects which uses the function point metric. However, backfiring had a high margin of error. A neuro-fuzzy approach was used to calibrate the conversion ratios to reduce the margin of error. The following conclusions were drawn from empirical results:

1. After calibrating the conversion ratios, the ratios still reflected the inverse curve relationship of the programming language level and the SLOC/FP.

2. The model overall improved in MRE, MER, SD, RSD, LSD and PRED when tested against the ISBSG data set.
3. The models had no bias towards underestimating (MRE bias) and overestimating (MER bias).
4. The model was flexible in training and modifying fuzzy sets if new data becomes available.

The calibration of backfiring conversion ratios would improve the accuracy of estimating the size of information systems and may result in more successful projects.

8 References

[1] K.K Aggarwal, Y. Singh, P. Chandra, M. Puri, "Bayesian regularization in a neural network model to estimate lines of code using function points", Journal of Computer Sciences, v 1 (4), 2005, pp. 505-9

[2] A.J. Albrecht, J.E. Jr. Gaffney, "Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation", IEEE Transactions on Software Engineering, v 9 (6), Nov. 1983, pp. 639-648.

[3] L. Angelis, I. Stamelos, M. Morisio, "Building a software cost estimation model based on categorical data", Proceedings Seventh International Software Metrics Symposium, 2000, pp. 4-15.

[4] Data CD R9 Demographics, International Software Benchmarking Standards Group, 2004.

[5] T. Foss, E. Stensrud, B. Kitchenham, I. Myrtveit. "A simulation study of the model evaluation criterion MMRE", Software Engineering, IEEE Transactions on Software Engineering, v 29 (11), Nov. 2003. pp. 985-995.

[6] Function Point Counting Practices Manual 4.2.1, International Function Point Users Group, www.ifpug.org, January 2005.

[7] X. Huang, D. Ho, J. Ren, and L.F. Capretz, "A Soft Computing Framework for Software Effort Estimation", Soft Computing, v 10 (2), Jan. 2006, pp. 170-177.

[8] A. Idri, T. M. Khoshgoftaar, A. Abran, "Can Neural Networks be easily Interpreted in Software Cost Estimation?", IEEE International Conference on Plasma Science, v 2, 2002, pp. 1162-1167.

[9] D.R. Jeffery, G. Low, "Calibrating estimation tools for software development", Software Engineering Journal, v 5 (4), Jul. 1990, pp. 215-221.

[10] C. Jones, "Backfiring: converting lines of code to function points", Computer, v 28 (11), Nov. 1995, pp. 87-8.

[11] C. Jones, “Programming Productivity”, McGraw-Hill, 1986.

[12] B.A. Kitchenham, S.G. MacDonell, L.M. Pickad, M.J. Shepperd, “What Accuracy Statistics Really Measure”, IEE Proceedings: Software, v 148 (3), Jun. 2001, pp. 81-85.

[13] J.P. Lewis, “Large Limits to Software Estimation”, ACM Software Engineering Notes, v 26 (4), Jul. 2001, pp. 54-59.

[14] R. P. Lippmann, “Introduction to Computing Neural Nets”, IEEE ASSP Magazine, v 4 (2) Part 1, Apr. 1987, pp. 4-22.

[15] J.M. Mendel, “Fuzzy Logic Systems for Engineering: a Tutorial”, in Proceedings of the IEEE, v 83 (2), Mar. 1995, pp. 345-347.

[16] D. J. Reifer, “Let the Numbers Do the Talking”, CrossTalk, Mar. 2002, pp 4-8.

[17] SPR, Programming Languages Table (PLT2006b), Software Productivity Research Incorporated, http://www.spr.com, 2006.

[18] K. Srinivasan, D. Fisher, “Machine learning approaches to estimating software development effort”, IEEE Transactions on Software Engineering, v 21 (2), 1995, pp. 126-136.

[19] R. D. Stutzke, “Estimating Software-Intensive Systems – Projects, Products, and Processes”, Addison-Wesley, 2005.