International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems
© World Scientific Publishing Company

CALIBRATING FUNCTION POINT BACKFIRING CONVERSION RATIOS USING NEURO-FUZZY TECHNIQUE

JUSTIN WONG*
Department of Electrical & Computer Engineering
University of Western Ontario
London, Ontario N6A 5B9, Canada

DANNY HO
NFA Estimation Inc., 32-538 Platt's Lane
London, Ontario N6G 3A8, Canada

LUIZ FERNANDO CAPRETZ
Department of Electrical & Computer Engineering
University of Western Ontario
London, Ontario N6A 5B9, Canada

Software size 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 this 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 size estimation is important in delivering successful software. Project estimation, used to determine the necessary development period and cost of projects, is vital to win project bids and help to establish the extent of a project's success. Many estimation techniques have been developed such as: Constructive Cost Model (COCOMO), Putnam's Software Lifecycle Management (SLIM), and Function Point Analysis.[1] Furthermore, machine learning techniques, such as neural networks and fuzzy logic have been applied to these estimation methods.

*4823 Dundas Street
Burnaby, British Columbia V5C 1B8, Canada

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 will 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. In addition, 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], 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.[3]

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				___

$$CAF = 0.65 + 0.01N$$

N is the total degree of influence of the 14 characteristics.

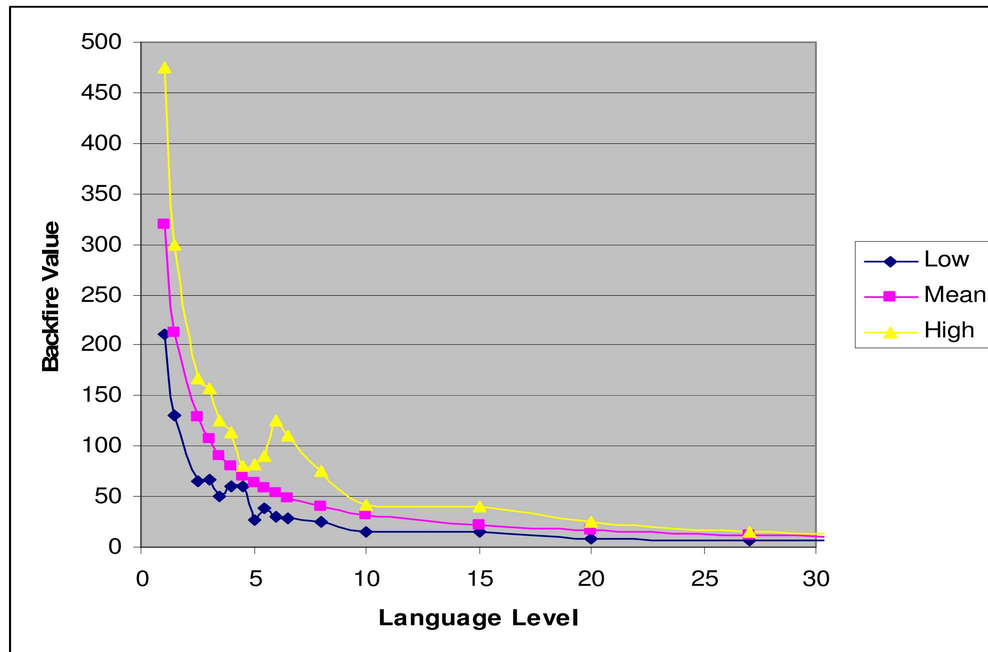
The degree of influence ranges from 0 to 5.[3] (1)

1.2. *Backfiring*

Experts have been using the term "high-level language" and "low-level language" for many years without precisely defining these phrases. Jones[4] classified programming languages by the number of statements they require for the implementation of one function point. Software Productivity Research (SPR)[5] 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. "High-level language" is defined as having less than 50 source lines-of-code per function point (SLOC/FP), while "low-level language" has over 100 SLOC/FP.[4]

Figure 1 – Conversion ratios versus language level.



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. For example: An application written in JAVA (36 SLOC/FP) with 300 function points would have an estimated SLOC of 10800. The conversion ratios can be used bilaterally mathematically; therefore, function points can be calculated from SLOC being divided by the conversion ratios.[1]

While backfiring is useful and simple, there is a high margin of error in converting SLOC data into function points.[4] This research introduces an approach to reducing this margin of error; thus, making the conversion ratios more reliable.

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. In classical set theory, an element either belongs or does not belong in a set. Fuzzy sets are an extension to the classical set theory because elements can be partially in a set. It is used to generate a mapping between input and output spaces.

The amount an element is in a set is measured with a membership function. Membership functions range from 0 to 1. Membership functions are used to describe linguistic terms such as low, medium and high. There are various types of fuzzy membership functions such as triangular, trapezoidal, and Gaussian.[6]

In fuzzy logic, fuzzy rules are used to define a fuzzy operator and are usually expressed using IF-THEN statements. An example of a fuzzy rule is shown in (2). The IF portion of the statement is the antecedent and the THEN portion is called the consequent. The antecedent statements can be linked with AND and OR.[6]

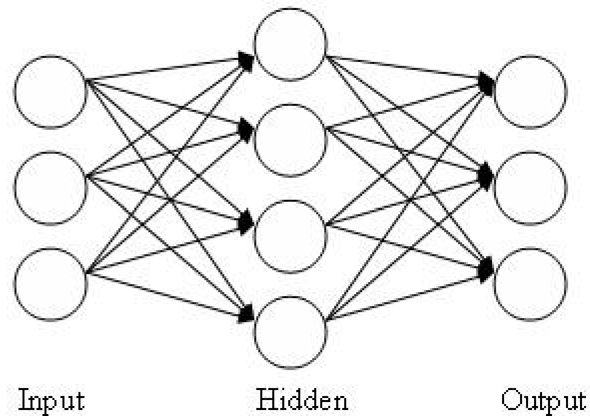
RULE 1 : IF inputLevel IS f1 THEN conversion IS o1;
RULE 2 : IF inputLevel IS f2 THEN conversion IS o2;
…
RULE 19: IF inputLevel IS f1 THEN conversion IS o19; (2)

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.[7] 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. Inputs are fed into neurons. Afterwards, the inputs are multiplied by their input weights and summed up. The summed result is then applied to an activation function which produces an output from the summed results. The outputs may be passed into another layer of neurons as inputs depending on the architecture of the neural networks. These inputs may be passed through many layers of the network until it reaches the output layer.[8] Figure 2 shows the design of a neural network which has one hidden layer containing four hidden nodes. In this neural network design, there are three inputs and it produces three outputs.

Figure 2 – A neural network



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.[8]

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 the Huang *et al.*[9] proposed model of combining fuzzy logic and neural network for calibrating the COCOMO estimation technique.

1.5. *Benchmark*

Foss *et al.*[10] 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 is particularly better than another because the results were misleading. MRE favored underestimation and performed worse in small sized projects. The equation is defined in (3). However, this method of evaluation is still popular and is commonly used in industry; for that reason, it was used to evaluate the experimental model.

$$MRE = \frac{|Actual - Predicted|}{Actual}. \quad (3)$$

Another method that was proposed for evaluating and comparing prediction models was Magnitude of error Relative to the Estimate (MER).[11] The equation for calculating MER is

defined in (4). MER was encouraged to be used for evaluation, yet it favors overestimation because the estimation is a divisor. Larger estimates tend to perform much better than small estimates.

$$MER = \frac{|Actual - Predicted|}{Predicted}. \quad (4)$$

Foss *et al.* concluded that Standard Deviation (SD), Residual Error Standard Deviation (RSD) and Logarithmic Standard Deviation (LSD) were good, consistent criteria.[10] The equation for SD is shown in (5). In the equation, y_i represents the actual result, $\hat{y}_i$ represents the predicted result, and n was the total number of projects. The equation for RSD is defined in (6). 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 (7). 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 for evaluation.

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

where y_i is actual, $\hat{y}_i$ is predicted and n is number of projects. (5)

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

where y_i is actual, $\hat{y}_i$ is predicted, x_i is input and n is number of projects. (6)

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

where y_i is actual, $\hat{y}_i$ is predicted, n is number of projects and s^2 is an estimator of the variance. (7)

Another criterion that was used to evaluate the prediction model was Prediction at Level (PRED). MRE less than 25%, and 50% were utilized for PRED because other models have used these criteria for evaluation.[11]

2. Related Work

The literature in software estimation models was extensive. The papers reviewed are neural network software estimation models.

Aggarwal *et al.*[12] 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. Furthermore, the network 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. In addition, the research 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.

Jeffery *et al.*[13] investigated an algorithm-based effort and management tool. They looked at the accuracy of lines of code as input, accuracy of the backfiring and comparison of the organization's delivery rate. The paper proposed that generic tools need to be calibrated for individual organizations to increase accuracy. It was shown that the SLOC/FP was not consistent across organizations for the same programming language. In this research, a neural network is used for calibrating the generic conversion ratios. The generic backfiring method was calibrated to improve the performance for a specific dataset.

Idri *et al.*[14] discussed 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 authors 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.*[15] compare 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. Mild outliers are data values, which lie between 1.5 to 3.0 times the interquartile range below the first quartile or above the third quartile.

Huang *et al.*[9] introduced a neuro-fuzzy framework and applied it to COCOMO II. The neuro-fuzzy technique showed it could be used to improve software effort 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:

(i) How can the inverse curve relationship between lines of source statements per function point and programming language level be modeled precisely?
(ii) How can the conversion ratio be calibrated to a specific data set to improve the accuracy?
(iii) How can we develop a system that can update the calibrated the conversion ratios when new data is available?
(iv) How are missing data for certain programming languages addressed?

The model was trained and evaluated with International Software Benchmark Standards Group's (ISBSG) Repository Data (release 10).[16] The model's estimate was measured against SPR's programming language table.[5]

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 (8).

$$y = 319.4x^{-0.997}$$

where y is the SLOC/FP and x is the language level. (8)

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 similar SLOC/FP. The number of fuzzy levels depends on the number of data points available. For example: In Table 2, levels 2.5 and 3.5 contain a lot of data points, therefore fuzzy levels 2.5 and level 3.5 can be created. There was no data available for many of the language levels therefore those programming language levels were skipped. 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[16]. The average SLOC/FP was obtained from the average of all the backfiring conversion values within a fuzzy level. Moreover, 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 10.

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

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 address research question 3.

4.2 *Fuzzy Membership Functions*

Figure 3 and Figure 4 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. 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 to convert the fuzzy sets into crisp numbers. The "Center of Gravity" method is used in this research because of its simplicity and popularity. Furthermore, the method allows generating a compromised value between the triangular functions to fill in the missing input SLOC/FP for certain language levels. The input membership functions would cover missing language levels which solved the 4th research question.

Figure 3 – Fuzzy membership of the language levels.

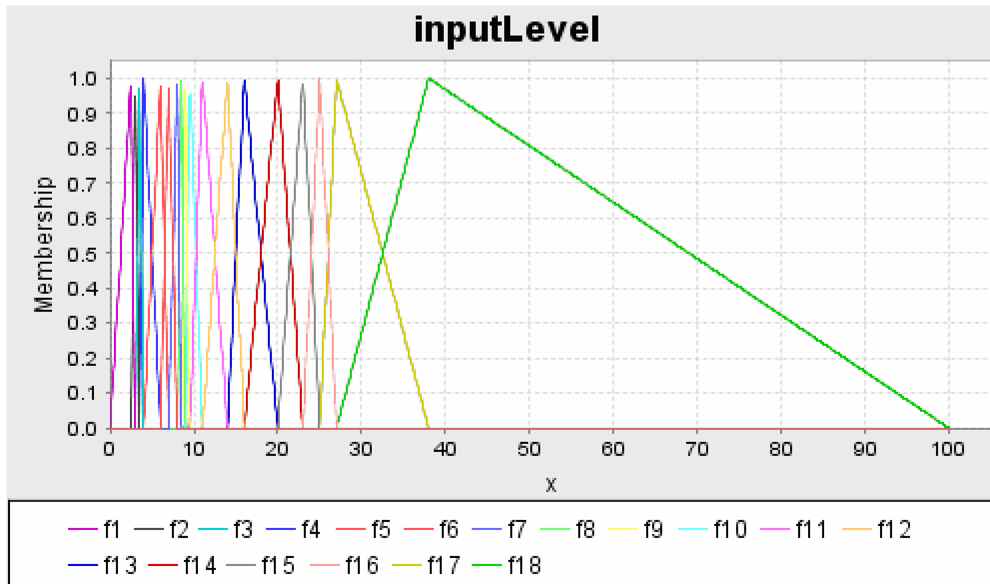
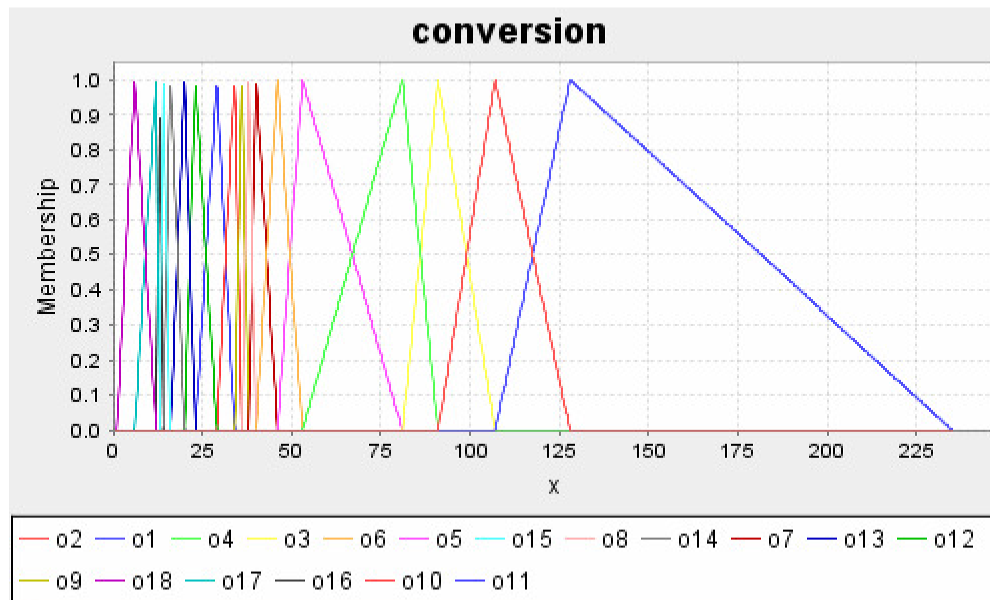


Figure 4 – Fuzzy membership functions of the output SLOC for each fuzzy level.



4.3. *Calibrating Fuzzy Sets with Neural Network*

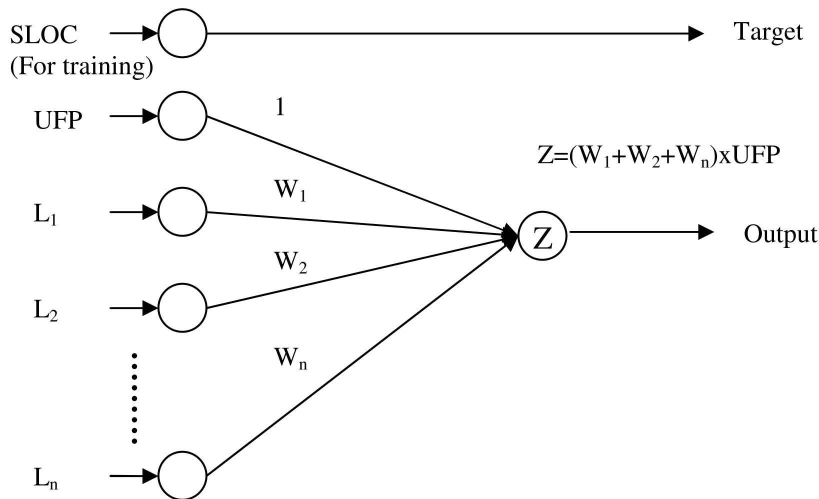
The neural network was used to calibrate the average source statements per function point for each fuzzy level. The input to the neural network was the SLOC, the UFP and the language level. The SLOC was used as the target during training and the UFP was used for both training and simulation. The language 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 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 limit of the SLOC/FP value. For the lower limit, 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

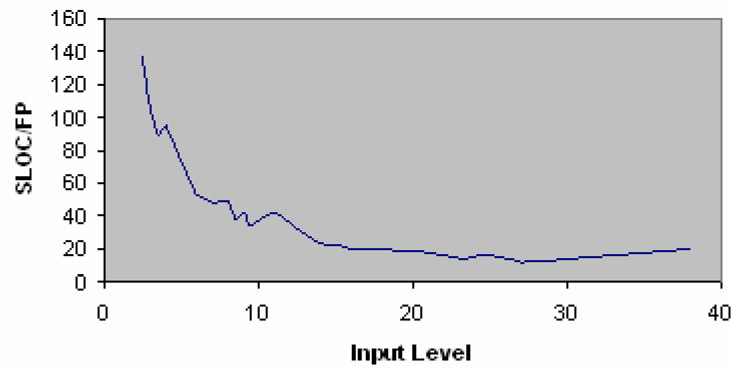
287 data points were used to compare the calibrated and the original conversion ratios. The data points contained the parameters required in the ISBSG[16] repository data. Furthermore, the mild outlier points were removed. Eight experiments were conducted for the original and calibrated model. In each experiment, half of the data was used for training and the other half was used for simulation. The data for training and simulation were randomly selected from the dataset for each experiment. Experiment 1's training data was separated into two batches of training data to solve research question 3.

Table 3 shows the initial and final calibrated values obtained for the fuzzy language levels when the network was trained in Experiment 1. Initially, 91 training points were used to generate the initial calibrated values. Afterwards, 54 more data points were added and the initial calibrated values were again recalibrated. 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 higher SLOC/FP. The language levels 8 and 9 have a higher than mean SLOC/FP and this was due to 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 in Experiment 1.

Fuzzy Level	Initial Calibrated Values (91 training points)	Final Calibrated Values (145 training points)
1	128.11	137.78
2	106.80	104.60
3	90.19	89.72
4	68.31	95.12
5	52.90	53.12
6	47.00	47.82
7	48.98	48.98
8	37.92	37.81
9	38.22	42.39
10	34.28	34.28
11	41.40	41.91
12	23.39	23.37
13	20.03	20.05
14	18.38	18.38
15	14.05	14.06
16	17.0	17.00
17	12.04	12.05
18	19.71	19.71

Figure 6 – Surface view.



5.1. *Results*

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 13.72% when calibrated and MMRE improved by 4.92%. For MMRE, experiment 1 had a negative improvement, and experiments 1, 2, 3, 4 and 6 showed small improvements.

Table 4 – MMER and MMRE.

	Original		**Calibrated**		**Improvement (%)**	
Experiment	**MMER**	**MMRE**	**MMER**	**MMRE**	**MMER**	**MMRE**
1	1.88	1.45	1.44	1.51	23.28	-4.66
2	2.02	3.63	1.70	3.60	15.58	0.80
3	1.58	1.83	1.48	1.78	6.70	2.96
4	2.09	3.32	1.56	3.27	25.51	1.60
5	1.77	3.37	1.46	3.08	17.50	8.66
6	1.30	1.26	1.20	1.21	7.89	4.29
7	1.07	0.92	0.99	0.82	7.30	10.77
8	1.34	4.71	1.28	4.56	6.00	15.00
Overall					**13.72**	**4.92**

Standard Deviation (SD), Residual Error Standard Deviation (RSD) and Logarithmic Standard Deviation (LSD) were shown to be good criteria to evaluate prediction models.[10] Table 5 shows the comparison of SD, RSD and LSD between the original and calibrated model. The calibration model had a small improvement for LSD. For the RSD and SD criteria, the results did not show any improvement and consistent result. Overall, there was no conclusive evidence that the calibration model had a SD, RSD and LSD improvement over the original model.

The PRED results, in table 6, showed inconsistent results. The improvements were more consistent for MER with PRED < 50%. Overall, the PRED results do not show that the calibrated model outperforms the original model because of the inconsistency.

The results for MMRE and MMER validated the experimental approach in solving the research questions and showed a small improvement the backfiring process, however there are still risks as the solution failed to show consistent improvements in the SD, RSD, and PRED criteria. Low improvements for MMRE, SD, RSD, and PRED could be caused because of limited training data for certain language levels. Another reason for the small and inconsistent improvements is the model tried to satisfy all the criteria, which resulted in only obtaining local minimum error points for each criteria.

Table 5 – SD, RSD, LSD.

	Improvement (%)		
Experiment	**SD**	**RSD**	**LSD**
1	-0.79	-0.11	3.87
2	-0.96	-4.57	2.38
3	-1.10	-3.34	3.93
4	0.68	1.21	4.50
5	4.38	-0.04	3.39
6	0.65	2.80	1.46
7	2.22	5.43	2.46
8	-3.44	-10.00	0.04
Overall Average	**0.21**	**-1.08**	**2.75**

Table 6 – PRED Validation Results.

	MRE (%)		MER (%)	
Experiment	**PRED < 25%**	**PRED < 50%**	**PRED < 25%**	**PRED < 50%**
1	-18.18	-8.20	-9.52	-1.47
2	15.00	-9.80	16.67	5.77
3	3.57	-7.84	24.00	9.80
4	0.00	-1.64	-5.71	2.90
5	13.79	-12.96	3.45	1.67
6	0.00	1.82	0.00	3.39
7	-2.78	3.70	-5.88	3.33
8	2.86	1.61	-3.13	-1.54
Overall Average	**1.78**	**-4.16**	**2.49**	**2.98**

6. Threats to Validity

There were threats to validity in this experiment. Certain language levels contained limited project data. 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 experiments, 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 38.

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[17] that in different application domain, the size and cost of the applications differ. Angelis *et al.*[18] showed a software cost estimation model based on attributes such as organization type, business type, development platform and development type. Furthermore, Jones[19] suggested that factors, such as the development environment, affect productivity. Therefore, these factors may have also affected the source lines of code per function point.

7. Conclusion

Backfiring had been used for many years for sizing projects which use 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 the empirical results:

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

- The model overall showed small improvements in MRE and MER when tested against the ISBSG data set, however, there was no clear and consistent evidence that it improved for SD, RSD and PRED criteria.
- The models had no bias towards underestimating (MRE bias) and overestimating (MER bias).
- As new data became available, the model was flexible in training and modifying fuzzy sets.

The calibration of backfiring conversion ratios would improve the accuracy of estimating the size of information systems and may result in more successful projects; however, there are still risks as the conversion ratios failed to improve size estimates that are already accurate.

Acknowledgements

Justin Wong would like to thank his supervisors and referees for their helpful comments. Furthermore, he would also like to thank SPR and ISBSG for providing research data.

References

1. R. D. Stutzke, *Estimating Software-Intensive Systems – Projects, Products, and Processes*, (Addison-Wesley, Upper Saddle River NJ, 2005).
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* **9** (1983) 639-648.
3. Function Point Counting Practices Manual 4.2.1, *International Function Point Users Group* (2005) www.ifpug.org.
4. C. Jones, Backfiring: converting lines of code to function points, *Computer* **28** (1995) 87-8.
5. SPR, Programming Languages Table (PLT2006b), *Software Productivity Research Incorporated* (2006) http://www.spr.com.
6. J. M. Mendel, Fuzzy Logic Systems for Engineering: a Tutorial, *Proceedings of the IEEE* **83** (1995) 345-347.
7. J. P. Lewis, Large Limits to Software Estimation, *ACM Software Engineering Notes* **26** (2001) 54-59.
8. R. P. Lippmann, Introduction to Computing Neural Nets, *IEEE ASSP Magazine* **4** (1987) 4-22.
9. X. Huang, D. Ho, J. Ren, and L.F. Capretz, A Soft Computing Framework for Software Effort Estimation, *Soft Computing* **10** (2006) 170-177.
10. T. Foss, E. Stensrud, B. Kitchenham, I. Myrtveit. A simulation study of the model evaluation criterion MMRE, *IEEE Transactions on Software Engineering* **29** (2003) 985-995.
11. B. A. Kitchenham, S. G. MacDonell, L.M. Pickad, M. J. Shepperd, What Accuracy Statistics Really Measure, *IEE Proceedings: Software* **148** (2001) 81-85.
12. 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* **1** (2005) 505-509.

13. D. R. Jeffery, G. Low, Calibrating estimation tools for software development, *Software Engineering Journal* **5** (1990) 215-221.
14. A. Idri, T. M. Khoshgoftaar, A. Abran, Can Neural Networks be easily Interpreted in Software Cost Estimation?, *IEEE International Conference on Plasma Science* **2** (Honolulu, 2002) 1162-1167.
15. K. Srinivasan, D. Fisher, Machine learning approaches to estimating software development effort, *IEEE Transactions on Software Engineering* **21** (1995) 126-136.
16. Data CD R10 Demographics, *International Software Benchmarking Standards Group* (2004) www.isbsg.org.
17. D. J. Reifer, Let the Numbers Do the Talking, *CrossTalk* (Mar. 2002) 4-8.
18. L. Angelis, I. Stamelos, M. Morisio, Building a software cost estimation model based on categorical data, *Proceedings Seventh International Software Metrics Symposium* (London, 2001) 4-15.
19. C. Jones, *Programming Productivity*, (McGraw-Hill, New York, 1986).