Improving the Performance of Neuro-Fuzzy Function Point Backfiring Model with Additional Environmental Factors

Justin Wong
SAP, justinplwong@gmail.com

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

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

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

Part of the Computer Engineering Commons, Electrical and Computer Engineering Commons, and the Software Engineering Commons

Citation of this paper:
Improving the Performance of Neuro–Fuzzy Function Point Backfiring Model with Additional Environmental Factors

Justin Wong¹, Danny Ho² & Luiz Fernando Capretz¹
¹University of Western Ontario, Canada
²NFA Estimation Inc., Canada

ABSTRACT
Backfiring is a technique used for estimating the size of source code based on function points and programming. In this study, additional software environmental parameters such as Function Point count standard, development environment, problem domain and size are applied to the Neuro-Fuzzy Function Point Backfiring (NFFPB) model. The neural network and fuzzy logic designs are introduced for both models. Both estimation models are compared against the same data source of software projects. It was found that the original NFFPB model outperforms the extended model. The results were investigated and explained to why the extended model performed worse.

INTRODUCTION
Software Estimation Background
Software estimation is important in software development. Project estimation is used to determine the necessary time and budget of projects. Many estimation techniques have been developed for software estimation such as: Constructive Cost Model (COCOMO), Putnam’s Software Lifecycle Management (SLIM), and
Function Point (Stutzke, 2005). Source lines of code (SLOC) and function points are two popular metrics used today.

SLOC metric is used to measure the amount of source code in software. SLOC can also be used to determine the cost and effort needed to develop a software application. SLOC can be referred to as physical SLOC or logical SLOC. Physical SLOC is the number of lines in an application. Problems with physical SLOC are that formatting and style affect the size of an application. Logical SLOC, on the other hand, is unaffected by formatting and style because it measures the number of statements. The SLOC metric still has shortcomings. It is sensitive to programming language and technology (Galorath & Evans, 2006). Despite these problems, SLOC is still a popular metric used in the software industry today.

Function point is a unit of measurement for determining the functional size of an information system introduced by Albrecht in the 1970s (Albrecht & Gaffney, 1983). In the 1990s, the popularity of function point grew. It became a major tool in software sizing. Function points are used today for sizing software in both industry and academia. As the usage of function points grew, the International Function Point User Group (IFPUG) was formed (International Function Point Users Group, 2007).

Function point analysis is a process of classifying major system components as ‘simple’, ‘average’, or ‘complex’. Unadjusted Function Points (UFP) is a measure obtained from identifying system components. The function components are Internal Logic Files (ILF), External Interface Files (EIF), External Inputs (EI), External Outputs (EO), and External Inquiries (EQ). The resulting function point count or Adjusted Function Points (AFP) is obtained by multiplying the UFP by the Value Adjustment Factor (VAF). There are 14 General System Characteristics (GSC) that defines VAF (International Function Point Users Group, 2005).

**Backfiring Technique**

Backfiring is a technique used to size source code by converting function points to logical SLOC statements (Jones, 1995). Backfiring can be accomplished by multiplying the function point with the conversion ratios to obtain the SLOC. Similarly, backfiring can also be used for calculating function points by dividing the SLOC by the conversion ratios (Stutzke, 2005). Jones classified that the conversion ratios are defined based on the number of statements required to implement one function point based on the programming language (Jones, 1995). Based on these classifications, “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 (Jones, 1995). Software Productivity Research (SPR)
annually publishes the conversion ratios for many programming languages (Software Productivity Research Incorporated, 2006). Table 1 illustrates SPR's programming language's SLOC/FP and language levels.

<table>
<thead>
<tr>
<th>Language</th>
<th>Language Level</th>
<th>SLOC/FP</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Low</td>
</tr>
<tr>
<td>Basic Assembly</td>
<td>1.0</td>
<td>213</td>
</tr>
<tr>
<td>C</td>
<td>2.5</td>
<td>21</td>
</tr>
<tr>
<td>Fortran</td>
<td>3.0</td>
<td>75</td>
</tr>
<tr>
<td>Cobol</td>
<td>3.0</td>
<td>65</td>
</tr>
<tr>
<td>C++</td>
<td>6.0</td>
<td>30</td>
</tr>
<tr>
<td>Java</td>
<td>9.0</td>
<td>20</td>
</tr>
<tr>
<td>SQL</td>
<td>25.0</td>
<td>8</td>
</tr>
<tr>
<td>Spreadsheet</td>
<td>50.0</td>
<td>1</td>
</tr>
<tr>
<td>MATHCAD</td>
<td>60.0</td>
<td>1</td>
</tr>
</tbody>
</table>

*Table 1. SPR's programming language level and SLOC/FP.*

The equation for converting function point to SLOC is defined in equation 1. In equation 1, FP is the function point input and conversion is the SLOC/FP. The SLOC equation can be rearranged to find the number of function points if the SLOC is known (Jones, 1995).

\[
SLOC = FP \times Conversion
\]

*Equation 1. SLOC Equation.*

**Problems with Backfiring Technique**

Backfiring is a simple and quick method to convert between function point and SLOC because of its simplistic formula shown in equation 1. However, there are major problems with backfiring.

Conversion ratios have a large range and it is difficult to determine the correct conversion ratios because of the large difference between the low and high SLOC/FP. For example, third generation languages have a range of 80 SLOC/FP (Software Productivity Research Incorporated, 2006). Selecting the wrong conversion ratio would result in a high error estimate. Furthermore, there are no instructions and method on choosing the correct conversion ratio. Another problem is conversion ratios are generic, therefore it does not reflect accurately to specific organizations and environments. There are so many different types of organizations and different software practices, making conversion ratios
unsuitable for most organizations. Furthermore, there is no documentation on what the mean conversion ratios represent.

Backfiring only uses function point count and programming language level as the input. It does not use any other inputs which can affect the conversion ratio value. Other factors such as business domain, type of development and team size can affect the size.

Backfiring fails to estimate enhancement projects because of reuse and changes in code. Furthermore, there is no consistent correlation between changed SLOC and changed function point because reusing code and program designs depend on the capabilities of the programming language.

Software implemented with an excessive amount of code may not mean it has a lot of functionality. Similarly, software implemented using less SLOC does not necessarily indicate that the software has less functionality.

Research Objective
A Neuro-Fuzzy Function Point Backfiring (NFFPB) model was developed to address the major problems of the backfiring technique (Wong & Ho & Capretz, 2008). The NFFPB model was shown to have a small improvement over the existing backfiring technique with its calibrated conversion ratios. However, the model only uses the function point count, conversion ratio and programming language level as inputs. It does not address other environmental factors that could affect software size.

In this study, an alternative extended NFFPB model is presented. The extended NFFPB model uses additional input parameters such as function point count standard, development environment, domain and team size. The extended NFFPB model is compared against the original NFFPB model to see if additional environmental input parameters would improve the software size estimates.

NEURO-FUZZY BACKFIRING MODEL DESIGN
The Neuro-Fuzzy Function Point Backfiring (NFFPB) model is a composite software size estimation model. It uses fuzzy logic, neural network, and a backfiring algorithmic model (Wong & Ho & Capretz, 2008).

Technical Overview
The NFFPB model is a method that calibrates the programming language level to improve backfiring estimations. Fuzzy logic is used to model the programming
language level curve by grouping the programming language levels into fuzzy sets. The fuzzy sets are used for the input programming language level. A neural network is used to calibrate the fuzzy sets' conversion ratios. The calibrated SLOC/FP replaces the original backfiring conversion ratios (Wong & Ho & Capretz, 2008). Figure 1 shows a technical view of the NFFPB model.

![Figure 1. Technical view of the NFFPB Model.](image)

**Design**

The objective of the NFFPB model is to improve the accuracy of the backfiring size estimates by calibrating the programming language's conversion ratio to a SPR's project dataset. Calibrated conversion ratios would produce more accurate estimates for specific organizations. The model consists of an input layer, processing layer, and output layer, which are shown in Figure 2.
Figure 2. NFFPB layer model.

The input layer is the preprocessing layer. In the preprocessing layer, each programming language from SPR are grouped into programming language levels. The group average SLOC/FP are then passed into the neural network in the processing layer. The input layer passes the historical function point data and SLOC project data from ISBSG. Furthermore, the programming language levels and SLOC/FP from SPR are inputted into the Fuzzy Programming Language Levels System (FPLLS). The FPLLS converts the programming language into a fuzzy programming language. The FPLLS includes fuzzy programming language level sets, fuzzy membership functions, and fuzzy language rules.

The function point data are the inputs into the neural network and the SLOC data is used for training data for the neural network. Within the processing layer, the conversion ratios are calibrated by a neural network by learning from historical data. The neural network takes the inputs from the input layer and produces a size estimate, which is compared with the actual result. The error difference is then minimized by having conversion ratios calibrated. By calibrating the conversion ratios, the trained neural network can be reused in similar project environments, without requiring further training.
The calibrated conversion ratios are passed to the output layer. Afterwards, the calibrated conversion ratios are applied to the FPLLS. The fuzzy membership functions are adjusted according to the conversion ratios; thus, the FPLLS is tuned to the new conversion ratios. Furthermore, the FPLLS can be used for simulating new size estimates (Wong & Ho & Capretz, 2008).

**Neural Network Training Data Source**

A data source is used to train the neural network in order to obtain calibrated conversion ratios. Furthermore, it is used for validation and comparison against the original conversion ratios.

The International Software Benchmarking Standards Group (ISBSG) is a non-profit organization made up of a group of national software metrics associations. The goal of the ISBSG organization is to collect and maintain software project data for research. In addition, the ISBSG project data repository contains metrics on software development, enhancement, maintenance, and support (International Software Benchmarking Standards Group, 2004).

In the NFFPB model, the UFP, function points, programming language and SLOC are used from the ISBSG Release 9 (International Software Benchmarking Standards Group, 2004). In the Release 9 repository, there are 3,024 projects from twenty countries, with 70% of the project data being less than 6 years old. Furthermore, there are over 70 programming languages used in development environments and reported in the data repository. The data fields that are needed for the NFFPB model are lines of code, primary programming language and UFP. All project data missing any of those three fields are filtered out—a total of 260 projects were used out of 3,024.

**Fuzzy Programming Language Levels System**

The FPLLS is a fuzzy rule-based inference system which is used in the input and output layers of the NFFPB model. In the input layer, it converts the input programming language levels into fuzzy language levels which is used as inputs into the neural network. After calibrating the conversion ratios with the neural network, the conversion ratios are used to adjust the fuzzy membership functions of the output SLOC for each fuzzy programming level. The tuned conversion ratios for each fuzzy language levels are used as the output. The main components of the FPLLS are the fuzzy language level, input and output membership functions, fuzzy rules, and tuned fuzzy system (Wong & Ho & Capretz, 2008).
By using the FPLLS, the margin of size estimate error by the backfiring technique is minimized. The FPLLS calibrates the generic conversion ratios into specific conversion ratios that are geared towards a certain software environment to improve the accuracy of size estimates.

**Fuzzy Language Level**

Figure 3 shows the relationship between the language levels to the mean SLOC/FP based on SPR’s project data (Software Productivity Research Incorporated, 2006) that was shown in table 1. Furthermore, the equation of this relationship is defined in equation 2.

The objective is to model project data from ISBSG into a curve similar to figure 1. However, the project data from ISBSG does not contain sufficient data for certain programming languages; therefore, fuzzy language levels are proposed to solve this problem. A fuzzy language level is an abstract level which contains various programming languages that are similar in programming language level. The programming languages were broken into language levels, which translated to the number of SLOC/FP. For example, from table 1, Fortran and Cobol have both have programming language level of 3.0 so they are grouped as a fuzzy language level 3. In another example, in the ISBSG project data, there was a limited data for programming languages with language levels of 27 to 50, therefore these programming languages were grouped together shown in table 2.
Figure 3. Mean inverse curve of language levels versus SLOC/FP.

\[ y = 319.4x^{-0.997} \]

where:

- \( y \) is the SLOC/FP,
- \( x \) is the language level.

Equation 2. Inverse curve equation.

The inverse relationship is modeled by grouping the language levels based on similar SLOC/FP into various fuzzy levels which are fuzzy sets (Mendel, 1995). Table 2 shows the language levels being grouped into various fuzzy levels to model the curve. A programming language level belongs in more than one fuzzy level. Based on the ISBSG project data, there were 260 project's SLOC/FP data points distributed into 19 fuzzy sets which represent language levels from 0 to 50 ISBSG (International Software Benchmarking Standards Group, 2004). In certain language levels such as programming language level 8 to 9.5, there were a lot of data available; therefore, the data in each of those programming language levels were grouped into more fine-grained fuzzy levels. At the higher programming language levels, there were less data available so each fuzzy level had a larger programming language range. The high programming languages also had similar low SLOC/FP values. For example, programming language Spreadsheet and MATHCAD from table 1 would be grouped together as fuzzy language level 19.

The average SLOC/FP was calculated by averaging all the backfiring conversion values within a fuzzy level shown in table 3. Moreover, these average values were used as initial weights in the neural network and the initial peak of the fuzzy membership functions. Figure 4 illustrates how the two sample programming languages are processed into the neural network (Wong & Ho & Capretz, 2008).
<table>
<thead>
<tr>
<th>Fuzzy Programming Language Level (1-10)</th>
<th>Fuzzy Programming Language Level (11-19)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fuzzy Level</td>
<td>Programming Language Level Range</td>
</tr>
<tr>
<td>1</td>
<td>[0, 2.5, 3]</td>
</tr>
<tr>
<td>2</td>
<td>[2.5, 3, 3.5]</td>
</tr>
<tr>
<td>3</td>
<td>[3, 3.5, 4]</td>
</tr>
<tr>
<td>4</td>
<td>[3.5, 4, 5]</td>
</tr>
<tr>
<td>5</td>
<td>[4, 5, 6]</td>
</tr>
<tr>
<td>6</td>
<td>[5, 6, 7]</td>
</tr>
<tr>
<td>7</td>
<td>[6, 7, 8]</td>
</tr>
<tr>
<td>8</td>
<td>[7, 8, 8.5]</td>
</tr>
<tr>
<td>9</td>
<td>[8, 8.5, 9]</td>
</tr>
<tr>
<td>10</td>
<td>[8.5, 9, 9.5]</td>
</tr>
</tbody>
</table>

Table 2. Fuzzy input language level membership function.

<table>
<thead>
<tr>
<th>Fuzzy Programming Language Level (1-10)</th>
<th>Fuzzy Programming Language Level (11-19)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fuzzy Level</td>
<td>Average SLOC/FP</td>
</tr>
<tr>
<td>1</td>
<td>[150, 128, 107]</td>
</tr>
<tr>
<td>2</td>
<td>[128, 107, 91]</td>
</tr>
<tr>
<td>3</td>
<td>[107, 91, 81]</td>
</tr>
<tr>
<td>4</td>
<td>[91, 81, 67]</td>
</tr>
<tr>
<td>5</td>
<td>[81, 67, 53]</td>
</tr>
<tr>
<td>6</td>
<td>[67, 53, 46]</td>
</tr>
<tr>
<td>7</td>
<td>[53, 46, 40]</td>
</tr>
<tr>
<td>8</td>
<td>[46, 40, 38]</td>
</tr>
<tr>
<td>9</td>
<td>[40, 38, 36]</td>
</tr>
<tr>
<td>10</td>
<td>[38, 36, 34]</td>
</tr>
</tbody>
</table>

Table 3. Fuzzy output SLOC/FP membership function.
Input and Output Membership Functions

Figure 5 and Figure 6 illustrate the input and output fuzzy membership functions. The input and output membership functions both use a triangular function (Mendel, 1995). For the input membership function, each fuzzy level is represented as a triangular function. The triangles are based on Table 2. In table 2, the programming language ranges are shown as $[a, b, c]$, a and c represent the left and right base of the triangle and b represent the peak.

For the output membership functions, the peak of each membership function represents the average SLOC/FP per fuzzy level (Wong & Ho & Capretz, 2008). The output triangles from figure 6 is based on table 3. The average SLOC/FP range are represented as $[a, b, c]$ notation. The a and c represent the left and right base of the triangle and b represent the peak.
Fuzzy Rules and Inference

Each fuzzy level was directly referenced to a fuzzy output. For example, \( f_1 \) referenced \( o_1 \), \( f_2 \) referenced \( o_2 \) and so on. Figure 7 shows the fuzzy rule block. For example, when programming language level 4 is the input, only RULE 3 and RULE 4 are activated and have a membership value greater than 0. The fuzzy membership values are then used for defuzzification into SLOC/FP. For defuzzification, the maximum accumulation method and center of gravity method were used (Wong & Ho & Capretz, 2008).
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 18 : IF inputLevel IS f18 THEN conversion IS o18;
RULE 19 : IF inputLevel IS f19 THEN conversion IS o19;

Figure 7. Fuzzy rule block.

CALIBRATING USING NEURAL NETWORK

The neural network is used to calibrate the generic conversion ratios within the processing layer. Afterwards, the newly calibrated conversion ratios are used by the fuzzy language levels in order to tune the FPLLs.

Neural Network Architecture

The neural network was used to calibrate the average source statements per function point for each fuzzy level. The SLOC, the UFP and the language level are the neural network inputs. 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. These grouped language levels were fed into the network shown in Figure 8, which shows the network’s design. The neural network was designed to be easily interpreted so that it avoids being a “black-box” model.

Figure 8. Neural network design.
The L₁ to Lₙ were binary grouped language level inputs. When a language level was fed into the network, the input was in the 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] based on the proposed programming language level groups.

The result is obtained from the activation function by multiplying the UFP with the weight of the programming language level. The weight's initial values are from Table 1. During training, the output was obtained and compared with the target result. The error between the actual result and the predicted result was propagated back to the input layer. Additionally, the weights were then adjusted based on the error using back-propagation (Wong & Ho & Capretz, 2008).

**Learning Process**

A learning algorithm was used to minimize the difference between the size estimated and the actual SLOC during training. The error equation is defined in equation 3.

\[
E = \text{actual} - \text{prediction}
\]

where:

\[E\] is the error signal.

*Equation 3. Error equation.*

The error was back-propagated from the output node to the input nodes of the neural network. In addition, the conversion weights were calibrated based on the error difference and the learning rate. The weight change equation is defined in equation 4.

\[
W'_i = W_i + nE
\]

where:

\[W_i\] is the \(i\)th initial conversion weight, \(W'_i\) is the \(i\)th modified conversion weight, \[n\] is the learning rate, \[E\] is the error signal.

*Equation 4. Weight change equation.*

During initial training, the weights were set to the original conversion ratio. The function point count and language level were inputted into the neural network. The size estimate was the output. Moreover, the estimate was compared with the actual SLOC to determine the error. Following the adjustment, the error was propagated back to the input nodes and the conversion weights were adjusted. This process continued until the specified epoch or error goal was reached.
Post-Tuned Fuzzy Programming Language Level System

The FPLLS is used in the input and output layer of the NFFPB model. In the input layer, it is used to convert programming language levels into a fuzzy language level. After the neural network calibrations of the conversion ratios are completed, the calibrated values are applied to the FPLLS. The calibrated conversion ratios are used to adjust the fuzzy output membership functions by adjusting the peak triangular functions based on the new conversion ratios. The new tuned fuzzy language levels are used in the output layer. The main components of the FPLLS are the fuzzy language level, input and output membership functions, fuzzy rules, and tuned fuzzy system.

The newly tuned FPLLS could then be used to perform size estimates more accurately. When more data becomes available, the neural network would be reused again to update the conversion values. The changes would then be applied back to the FPLLS (Wong & Ho & Capretz, 2008).

NEURO-FUZZY BACKFIRING MODEL WITH ENVIRONMENTAL FACTORS

In this research, external environmental factors were investigated because the backfiring technique only takes the function point count and programming language level as inputs. An extension was added to the NFFPB model to determine if the performance improves with additional inputs.

The extension to the NFFPB model uses similar environmental factors as Angelis et al.’s software cost estimation model based on ISBSG's environmental factor data (Angelis & Stamelos & Morisio, 2000). The new inputs used were development type, development platform, organization type, business area type, application type, maximum team size, function point standard, and architecture.

Similarly to Angelis et al.’s approach, the data for each factor were sorted into low and high levels and the averages were taken (Angelis & Stamelos & Morisio, 2000). For each different type of value in a factor, the average error between the actual SLOC and the estimated SLOC was observed. For example, for the environmental factor development type, the values were new development, redevelopment and enhancements. For each of these values, the average error, defined in equation 3, was taken. If the average error is negative, that value would be classified as LOW. Moreover, if the average error is positive, the value is classified as HIGH.
**Function Point Count Standard**

The function point metrics used in this model are NESMA (NESMA, 2006) and IFPUG (International Function Point User Group, 2007). The older function point metrics were NESMA 1.0 and IFPUG 4.0. The up-to-date function point counts are NESMA 2.0 and IFPUG 4.1, which were very similar to one another (NESMA, 2006). However, when an error average test was performed, it was found that projects using the NESMA count had a negative error, while the IFPUG counts had a positive error; therefore, NESMA was grouped as HIGH and IFPUG was grouped as LOW. The high level indicates that when NESMA count is used, the SLOC is going to be overestimated. For the low level IFPUG, it indicates that the SLOC would be underestimated.

**Development Environment**

The development project type, development platform, and architecture categories were sorted as low and high based on each of the factors, where low represented low complexity and high represented high complexity. The result of the classifications was the same as Angelis et al. classification of ISBSG's categories (Angelis & Stamelos & Morisio, 2000).

New development, redevelopment and enhancement projects were investigated and classified. New development types were considered high complexity and redevelopment and enhancement were considered low complexity because of code reuse and previous knowledge of the project development. The error average test confirmed the high and low levels because new development had a positive error, while redevelopment and enhancement had a negative error.

For development platforms, the ISBSG repository contained three types of platforms, which were personal computer (PC), mainframe (MF), and mid-range (MR). PCs were considered to be low complexity because of their simplicity. MF and MR were high complexity because they were more difficult in terms of development. These classifications were confirmed from the error average test because MF and MR had a positive error and PC had a negative error.

In the ISBSG architecture category, there were two types of architectures, which were client-server and standalone architecture. Client-server architecture was high because of its complexity, while standalone was low. The error average test confirmed that client-server had a large positive error and standalone had a negative error.
Domain
The domain categories were investigated to see how they affect size estimate. It was shown by Reifer that the size and cost of applications differ in different application domains (Reifer, 2002). Therefore, the organization type, business area type, and application type categories were investigated.

The organization types in the ISBSG repository were services, communications, finance, manufacturing, and operations. An error average test was performed for each organization type. Finance and communication were low complexity, while services, manufacturing, and operations were high complexity.

In the ISBSG project repository, the business area types were accounting, banking, engineering, insurance, inventory, manufacturing, marketing, sales and telecommunications. From the average error test, the low complexity business area types were accounting, banking, insurance, inventory, and sales. In addition, the business area types that were high complexity were engineering, manufacturing, marketing, and telecommunications.

The application types in the ISBSG repository were business, electronic data interchange, process control, network management, management information system, office information system, stock order/order processing system, workflow support/management, and transaction/production system. After conducting the average error test for the application types, the high complexity types found were business, management information system, transaction/production system, and process control. The low complexity types were electronic data exchange, network management, office information systems, stock order/order processing system, and workflow support/management.

Team Size
Maximum team size was used in Aggarwal et al.’s size estimation model (Aggarwal & Singh & Chandra & Puri, 2005) and Angelis et al.’s cost estimation model (Angelis & Stamelos & Morisio, 2000). The maximum team size was another input added to the extended NFFPB model. The team size could affect the software size because of large teams, source code management software and software programming standards.

Extended Model Design
The extended NFFPB model uses additional environmental inputs, which were organization type (OT), business area type (BT), application type (AT),
development project type (DT), development platform (DP), architecture (AR), function point standard count (count) and maximum team size (TEAM). The low complexity inputs were treated as -1 and high complexity is +1 in the neural network. For project data which had missing data fields, the neural network inputs are 0. Figure 9 illustrates the additional inputs being passed through an activation function. The environmental factor's weights were initially set to 1 and are adjusted during training. The activation function used was a sigmoid function because the input parameters were patterns. Furthermore, the results of the sigmoid function were used to calibrate the conversion ratio. This was done by multiplying the sigmoid function by difference of the maximum and minimum SLOC/FP for each programming language from SPR's conversion ratios. This difference is called the change value, which is defined in equation 5. The sigmoid function is used because it's output ranges from 0 to 1. The sigmoid function represents the adjustment between the minimum and maximum conversion ratio value of the programming language. For example, the sigmoid function can adjust the conversion ratio of the programming language Java between the minimum conversion ratio of 20 SLOC/FP to the maximum conversion ratio of 50 SLOC/FP, shown in table 1.

![Figure 9. Development Environmental Input Parameters.](image)
\[ \text{Change} = \max(C_i) - \min(C_i) \]

where:

- \( \text{Change} \) is the change value,
- \( \max \) is the maximum SLOC/FP,
- \( \min \) is the minimum SLOC/FP,
- \( C_i \) is the \( i \)th programming language level conversion ratio range.

*Equation 5. Difference equation.*

Figure 10 shows the extended NFFPB model. The output node’s activation function, shown in equation 6, is the sum of the environmental factors multiplied and the change factor, the original UFP multiplied by conversion ratio, and the maximum team factor. The learning process was the same as the original model, except the error also propagates back to the environment inputs and team weights. Afterwards, the weights were adjusted.
Figure 10. Extended NFFP model.
Output = (Team \times W_{Team}) + (Y \times W_{ENV}) + Z

where:
Team is the maximum team size, \(W_{Team}\) is the team weight,
Y is the result of the sigmoid function multiplied by the Change,
\(W_{ENV}\) is the weight of the environment factors,
Z is the result of the \(W_{i}\) multiplied by the UFP.


EVALUATING THE MODELS

Performance Evaluation Criteria

Magnitude of Relative Error (MRE) should not be used when evaluating and comparing prediction models because the results were misleading (Foss & Stensrud & Kitchenham & Myrtveit, 2003; Kitchenham & MacDonell & Pickad & Shepperd, 2001). MRE favored underestimation and performed worse in small sized projects. The equation for MRE is defined in equation 7. Despite the misleading results, this method of evaluation is still commonly used in industry; thus, it was used to evaluate the experimental model.

\[
MRE = \frac{|actual - predicted|}{actual}
\]

Equation 7. MRE.

Another method used for comparing prediction models was Magnitude of error Relative to the Estimate (MER) (Kitchenham & MacDonell & Pickad & Shepperd, 2001). The MER is defined in equation 8. MER has shortcomings similar to MRE. It favors overestimation because the estimation is a divisor; therefore larger estimates tend to perform much better than smaller estimates.

\[
MER = \frac{|actual - predicted|}{predicted}
\]

Equation 8. MER.

Standard Deviation (SD), Residual Error Standard Deviation (RSD) and Logarithmic Standard Deviation (LSD) were shown to be good and consistent criteria (Foss & Stensrud & Kitchenham & Myrtveit, 2003). The equation for SD is shown in equation 9, where \(y_{i}\) represents the actual SLOC, \(\hat{y}_{i}\) represents the predicted SLOC, and \(n\) is the total number of project points. Whereas in the RSD equation defined in equation 10, \(y_{i}\) represents the actual SLOC, \(\hat{y}_{i}\) represents the
predicted SLOC, \( x_i \) represents the number function points and \( n \) represents the total number of projects. The equation for LSD, shown in equation 11, \( y_i \) represents the actual SLOC, \( \hat{y}_i \) represents the predicted SLOC, \( n \) is the total number of projects, and the \( s^2 \) is the variance of \( \ln y_i - \ln \hat{y}_i \). In addition to MRE and MER, SD, RSD and LSD were used for evaluation.

\[
SD = \sqrt{\frac{\sum (y_i - \hat{y}_i)^2}{n - 1}}
\]

where:
\( y_i \) is actual SLOC, \( \hat{y}_i \) is predicted SLOC, \( n \) is number of projects.

Equation 9. SD.

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

where:
\( y_i \) is actual SLOC, \( \hat{y}_i \) is predicted SLOC, \( x_i \) is number of Function Points, \( n \) is number of projects.

Equation 10. RSD.

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

where:
\( y_i \) is actual SLOC, \( \hat{y}_i \) is predicted SLOC, \( n \) is number of projects, \( s^2 \) is the variance of \( \ln y_i - \ln \hat{y}_i \).

Equation 11. LSD.

Prediction at Level (PRED) was another criteria used to evaluate the prediction models. PRED(n) is the number of projects with a MRE lower than the n%. MRE less than 25% and 50% was utilized for PRED because other models have used these criteria for evaluation. PRED(25%) measures the number of projects with MRE less than 25% error and PRED(50%) measures the number of projects with MRE less than 50% error.
The goal of the NFFPB models is to achieve lower MRE, MER, SD, RSD and LSD values than the original conversion ratios. In the PRED criteria, the objective of the NFFPB models is to have a larger value than the original conversion ratios.

Experiment Methodology

The tuned NFFPB model’s calibrated conversion ratios were benchmarked against the original conversion ratios from Software Productivity Research (SPR) (Software Productivity Research Incorporated, 2006). There were seven different experiments conducted for the original NFFPB and the extended NFFPB model. In each experiment, the MMRE, MMER, SD, LSD, and RSD were compared with the original and calibrated ratios.

The dataset was divided in half for training and evaluation in the first two experiments. The data points were randomly selected for each programming language level in each of the experiments.

In the next two experiments, the dataset was sorted based on the size of the projects. The effect on the performance was investigated when the NFFPB tool was trained with projects that have small function point counts and tested against projects with large function point counts. In the second size experiment, the opposite was performed. Large function point count projects were used for training and the small function point counts were used for simulation. A larger training set was used to see if the performance improves. 75% of the dataset was used for training in two of the experiments. The data points were randomly selected for each programming level.

RESULTS

50% Random Test

In Experiment 1, the extended model was trained with the same data as the original experiment. The results were very similar to the results of the original model. Both models outperformed the original conversion ratios. Table 4 shows the comparison of the extended model against the original model’s conversion ratios and the traditional backfiring conversion ratios.

<table>
<thead>
<tr>
<th>Evaluation Criteria</th>
<th>Original Conversion Ratio</th>
<th>Calibrated Conversion Ratio</th>
<th>Extended Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>MMRE</td>
<td>1.26</td>
<td>1.21</td>
<td>1.20</td>
</tr>
<tr>
<td>MMER</td>
<td>1.30</td>
<td>1.20</td>
<td>1.20</td>
</tr>
<tr>
<td>PRED(MRE&lt;25%)</td>
<td>31</td>
<td>31</td>
<td>31</td>
</tr>
</tbody>
</table>
Table 4. Experiment 1.

In Experiment 2, the same experiment was conducted on both models except using a different dataset. The results were very similar to the calibrated conversion ratios shown in Table 5.

<table>
<thead>
<tr>
<th>Evaluation Criteria</th>
<th>Original Conversion Ratio</th>
<th>Calibrated Conversion Ratio</th>
<th>Extended Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>MMRE</td>
<td>4.40</td>
<td>4.36</td>
<td>4.38</td>
</tr>
<tr>
<td>MMER</td>
<td>1.16</td>
<td>1.02</td>
<td>1.02</td>
</tr>
<tr>
<td>PRED(MRE&lt;25%)</td>
<td>39</td>
<td>40</td>
<td>39</td>
</tr>
<tr>
<td>PRED(MRE&lt;50%)</td>
<td>60</td>
<td>60</td>
<td>59</td>
</tr>
<tr>
<td>SD</td>
<td>26133.26</td>
<td>26038.71</td>
<td>26052</td>
</tr>
<tr>
<td>LSD</td>
<td>1.12</td>
<td>1.08</td>
<td>1.08</td>
</tr>
<tr>
<td>RSD</td>
<td>85.97</td>
<td>85.99</td>
<td>85.99</td>
</tr>
</tbody>
</table>

Table 5. Experiment 2.

Size Test

In Experiment 3, the extended model demonstrated the same result as calibrated conversion ratio. Table 6 presents the result that the extended model produced the same improvement.

<table>
<thead>
<tr>
<th>Evaluation Criteria</th>
<th>Original Conversion Ratio</th>
<th>Calibrated Conversion Ratio</th>
<th>Extended Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>MMRE</td>
<td>0.92</td>
<td><strong>0.82</strong></td>
<td><strong>0.82</strong></td>
</tr>
<tr>
<td>MMER</td>
<td>1.07</td>
<td><strong>0.99</strong></td>
<td><strong>0.99</strong></td>
</tr>
<tr>
<td>PRED(MRE&lt;25%)</td>
<td>36</td>
<td>36</td>
<td>37</td>
</tr>
<tr>
<td>PRED(MRE&lt;50%)</td>
<td>54</td>
<td>58</td>
<td>58</td>
</tr>
<tr>
<td>SD</td>
<td>10347.88</td>
<td>10117.83</td>
<td>10120.42</td>
</tr>
<tr>
<td>LSD</td>
<td>0.92</td>
<td>0.87</td>
<td>0.87</td>
</tr>
<tr>
<td>RSD</td>
<td>70.85</td>
<td>69.11</td>
<td>69.13</td>
</tr>
</tbody>
</table>

Table 6. Experiment 3.

In Experiment 4, the extended model had a negative MMRE improvement. The evaluation results are shown in Table 7. The calibrated conversion ratios continue to outperform the original conversion ratio and the extended model.
<table>
<thead>
<tr>
<th>Evaluation Criteria</th>
<th>Original Conversion Ratio</th>
<th>Calibrated Conversion Ratio</th>
<th>Extended Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>MMRE</td>
<td><strong>4.74</strong></td>
<td><strong>4.56</strong></td>
<td><strong>4.84</strong></td>
</tr>
<tr>
<td>MMER</td>
<td>1.32</td>
<td>1.28</td>
<td>1.27</td>
</tr>
<tr>
<td>PRED(MRE&lt;25%)</td>
<td>35</td>
<td>34</td>
<td>36</td>
</tr>
<tr>
<td>PRED(MRE&lt;50%)</td>
<td>62</td>
<td>63</td>
<td>62</td>
</tr>
<tr>
<td>SD</td>
<td>31947</td>
<td>32080.82</td>
<td>31964.92</td>
</tr>
<tr>
<td>LSD</td>
<td>1.22</td>
<td>1.21</td>
<td>1.21</td>
</tr>
<tr>
<td>RSD</td>
<td>68.87</td>
<td>68.81</td>
<td>68.76</td>
</tr>
</tbody>
</table>

Table 7. Experiment 4.

**75% Random Test**

In the random test experiments, where more training data was provided, the extended model produced results similar to the calibrated conversion ratio. In Experiment 5 and experiment 6, different datasets were used. Table 8 and 9 illustrate the similarities. Both models outperformed the original conversion ratios.

<table>
<thead>
<tr>
<th>Evaluation Criteria</th>
<th>Original Conversion Ratio</th>
<th>Calibrated Conversion Ratio</th>
<th>Extended Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>MMRE</td>
<td><strong>1.07</strong></td>
<td><strong>0.99</strong></td>
<td><strong>0.99</strong></td>
</tr>
<tr>
<td>MMER</td>
<td>1.30</td>
<td><strong>1.18</strong></td>
<td><strong>1.18</strong></td>
</tr>
<tr>
<td>PRED(MRE&lt;25%)</td>
<td>17</td>
<td>17</td>
<td>17</td>
</tr>
<tr>
<td>PRED(MRE&lt;50%)</td>
<td>31</td>
<td>30</td>
<td>30</td>
</tr>
<tr>
<td>SD</td>
<td>26134.51</td>
<td>24891.99</td>
<td>24920.66</td>
</tr>
<tr>
<td>LSD</td>
<td>1.04</td>
<td>0.99</td>
<td>0.99</td>
</tr>
<tr>
<td>RSD</td>
<td>75.48</td>
<td>73.83</td>
<td>73.84</td>
</tr>
</tbody>
</table>

Table 8. Experiment 5.

<table>
<thead>
<tr>
<th>Evaluation Criteria</th>
<th>Original Conversion Ratio</th>
<th>Calibrated Conversion Ratio</th>
<th>Extended Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>MMRE</td>
<td>1.25</td>
<td><strong>1.14</strong></td>
<td><strong>1.13</strong></td>
</tr>
<tr>
<td>MMER</td>
<td>0.77</td>
<td><strong>0.72</strong></td>
<td><strong>0.73</strong></td>
</tr>
<tr>
<td>PRED(MRE&lt;25%)</td>
<td>19</td>
<td>21</td>
<td>21</td>
</tr>
<tr>
<td>PRED(MRE&lt;50%)</td>
<td>30</td>
<td>31</td>
<td>31</td>
</tr>
<tr>
<td>SD</td>
<td>18107.94</td>
<td>17514.07</td>
<td>17513.33</td>
</tr>
<tr>
<td>LSD</td>
<td>0.94</td>
<td>0.90</td>
<td>0.90</td>
</tr>
<tr>
<td>RSD</td>
<td>51.10</td>
<td>48.25</td>
<td>48.01</td>
</tr>
</tbody>
</table>

Table 9. Experiment 6.
DISCUSSIONS

NFFPB Models vs Original Conversion Ratios
The results, from the experiments, have shown that the NFFPB models have small improvement over the SPR’s backfiring conversion ratios. Despite the small improvement, it was shown that with a larger dataset, a better improvement could be obtained.

The improvements were small in each experiment due to a limited number of data available. Because certain fuzzy language levels had insufficient data points, there was limited calibration for those language levels, which resulted in smaller improvements. However, regardless of the setback, in the 75% random test results, the improvements in MMRE and MMER were greater than in the 50% random test results. Another reason for the small improvements is that the model tried to satisfy the MMRE, MMER, PRED(MRE<25%) and PRED(MRE<50%) performance criteria, which resulted in only obtaining local minimum error points for each criteria.

NFFPB Model vs Extended NFFPB Model
The extended model produced similar to worse results when compared to the original model due to various reasons. The dataset showed limited consistency for the categorical parameters. Moreover, while categorical parameters were shown to have impact on effort, it does not necessarily mean that they have a direct impact on size. More investigation in environmental variables that affect software size is needed.

Decreasing the size of a model and reducing the number of variables was shown to improve effort estimation (Chen & Menzies & Port & Boehm, 2005; Kirsopp & Shepperd, 2002; Miller, 2002). Based on the studies, the extended model would be worse than the original model. In addition, the results have shown that there was no improvement.

Threats to Validity
Certain language levels contained limited project data. If more data was available in those language levels, the results may have been different. However, other languages that had sufficient data points show improvement and have similar behavior. Limited data in certain languages may have resulted in a less accurate model of the SLOC/FP versus language level curve.

Study has also shown that using different historical project data sets or experimental designs have different estimation accuracies when performed on the
same model (Wen & Li & Lin & Hu & Huang, 2011). Even though different types of experimental designs and randomly selected project data were used, if a different project data set was used, there may be a possibility that the NFFPB model may have a much more different result.

Another threat to validity was that other environmental parameters may exist, which could have a direct effect on the programming language's SLOC/FP. For example, the general system characteristics for function point analysis may affect the estimate in lines of code. Studies have shown that the size and costs are affected by different application domains (Reifer, 2002). Angelis et al. showed a software cost estimation model based on attributes such as organization type, business type, development platform and development type; thus, these factors may have also affected the source lines of code per function point (Angelis & Stamelos & Morisio, 2000). Guyon et al. presented a variable selection method that could be used to identify which factors influence the size of the software (Guyon & Elisseeff, 2003). By using the variable selection method, the number of factors could be reduced which could improve the performance of the NFFPB model.

**FUTURE RESEARCH DIRECTIONS**

**Research in factors and variables that affect software size**

The factors used in the extended model did not improve performance over the original calibrated model because they were proven to affect software effort but not size. Some other factors should be investigated such as reusability, security, system age (legacy and new), and system performance which may have more impact on software size.

**Combine calibrated function points with calibrated conversion ratios**

Xia developed a neuro-fuzzy function point technique which calibrates the function point complexity values (Xia & Ho & Capretz, 2008). In the study, the estimation model’s size estimations showed improvements over the traditional complexity values. Others (Huang & Ho & Ren & Capretz, 2004; Du & Ho & Capretz, 2010) found improvements in estimation accuracy when a neuro-fuzzy approach is combined with an algorithmic model. Using the calibrated complexity values with the NFFPB model could potentially enhance the NFFPB model’s size estimation.

**CONCLUSION**

The backfiring estimation technique had some major shortcomings such as the conversion ratios had a large range and were generic. Moreover, it only used the
programming language and the function point count as inputs. Furthermore, this estimation technique did not work for enhancement projects.

The NFFPB was introduced to solve these problems by calibrating the conversion ratios. The NFFPB model used both fuzzy logic and neural network. Fuzzy logic was used to model the relationship between programming language level and source lines of code per function point (SLOC/FP). The neural network was used to calibrate the conversion ratios.

An extended NFFPB model was developed which used other inputs such as business area, organization type, development platform, function point count type, and team size. An investigation was conducted to determine if there was an improvement over the original NFFPB model. The NFFPB models were benchmarked against SPR’s conversion ratios. The ISBSG release 9 project data repository was used for training and evaluation. The extended model took additional input parameters such as environmental factors and team size.

The NFFPB models were shown to have an improvement in the Mean Magnitude of Relative Error (MMRE), Magnitude of error Relative to the Estimate (MER), Standard Deviation (SD), Logarithmic Standard Deviation (LSD), and Residual Standard Deviation (RSD). In the experiment where the training data set was smaller, the improvements were smaller. However, when a larger training set of data was used, the improvements increased.

The size of the data set available for training and evaluation weakened the conclusions being drawn because if more data became available, a greater improvement would show. Furthermore, there were only small improvements in the neural network because during training, the neural network was minimizing error based on both the MMRE and MMER criteria, which resulted in only obtaining local minimum error points (Angelis & Stamelos & Morisio, 2000).

The extended model was tested against the original NFFPB model and was shown to produce the same or worse results. A reason why there was no improvement over the original NFFPB model was that these additional factors have only been shown to affect software effort. There was no indication whether these factors would affect the size of a software system. Furthermore, it was shown that adding more variables into the model does not improve its performance estimation (Chen & Menzies & Port & Boehm, 2005; Kirsopp & Shepperd, 2002; Miller, 2002).

REFERENCES


Reifer, D. J. (2002). Let the numbers do the talking. *CrossTalk*, 4-8.


