Software Effort Estimation using Hybridized Search Based Techniques

Dissertation

Submitted in partial fullfillment of the requirement for award of the degree of

MASTER OF TECHNOLOGY

IN

SOFTWARE TECHNOLOGY

Submitted by

MOHIT PRASAD

(Roll No: 2K13/SWT/08)

MAJOR PROJECT II

(Paper Code: CO 821)

Under the Esteemed Guidance of

Dr Ruchika Malhotra

Associate Head & Assistant Professor,

Department of Computer Science & Engineering

DELHI TECHNOLOGICAL UNIVERSITY

SHAHBAD DAULATPUR, MAIN BAWANA ROAD, DELHI 110042

INDIA

DELHI TECHNOLOGICAL UNIVERSITY
NEW DELHI

DECLARATION

I hereby declare that the project work entitled "**Software Effort Estimation using Hybridized Search Based Techniques**" done by me under the guidance of Dr Ruchika Malhotra, Associate Head & Assistant Professor, Department Of Computer Science & Eng., Delhi Technological University and this project work is submitted in the partial fulfillment of the requirements for the award of the degree of M.Tech in Software Technology. The results embodied in this thesis have not been submitted to any other University or Institute for the award of any degree or diploma.

Date:

Signature:

Mohit Prasad

2K13/SWT/08

DELHI TECHNOLOGICAL UNIVERSITY
NEW DELHI

CERTIFICATE

This is to certify that the thesis entitled, "**Software Effort Estimation using Hybridized Search Based Techniques**", is a bona fide work done by **Mr. MOHIT PRASAD** in partial fulfillment of requirements for the award of Master of Technology Degree in Software Technology at Delhi Technological University (New Delhi) is an authentic work carried out by him under my supervision and guidance. The matter embodied in the thesis has not been submitted to any other University / Institute for the award of any Degree or Diploma to the best of my knowledge.

Date:

Signature:

Dr. Ruchika Malhotra
Department of Computer Science and Engineering
Delhi Technological University

ACKNOWLEDGMENT

I am presenting my work on "**Software Effort Estimation using Hybridized Search Based Techniques**" with a lot of pleasure and satisfaction. I take this opportunity to thank my supervisor, Dr Ruchika Malhotra, for guiding me and providing me with all the facilities, which paved way to the successful completion of this work. This thesis work was enabled and sustained by his vision and ideas. His scholarly guidance and invaluable suggestions motivated me to complete my thesis work successfully. I would like to express my deep gratitude to my parents. Their continuous love and support gave me strength for pursuing my dream. I am thankful to my friends and colleagues who have been a source of encouragement and inspiration throughout the duration of this thesis. I am also thankful to the SAMSUNG who has provided me opportunity to enroll in the M.Tech Program and to gain knowledge through this program. This curriculum provided me knowledge and opportunity to grow in various domains of computer science. Last but not least, I am thankful to all the faculty members who visited the Samsung premises to guide and teach. Their knowledge and efforts helped me to grow and learn in the field of computer science. I feel proud that their contribution helped me to bring out new ideas in my professional life. This project has provided me knowledge in the area of Software Cost Estimation and helped me in understanding the application of Neural Network & Genetic Algorithm Techniques for Software Effort Estimation.

Mohit Prasad

2K13/SWT/08

ABSTRACT

Prediction of resource requirements of a software project is crucial for the timely delivery of quality-assured software within a reasonable timeframe. Software effort estimation is the process of prognosticating the amount of effort required to build a software project**.** Most cost estimation models attempts to generate an effort estimation, which can then be mapped into project duration and cost. Many conventional (model-based) and Artificial Intelligence (AI) oriented (model-free) resource estimators have been proposed in the recent past. In this thesis two search based Effort Estimation techniques are discussed .Firstly we evaluates a genetically trained neural network (NN) predictor trained on historical data. Secondly, Particle Swarm Optimization (PSO) technique which operates on data sets clustered using the K-means clustering algorithm. Hence PSO and Genetic Algorithm (GA) based search techniques are employed to perform optimized search in solution space.

The comparison of this new predictor is established using *n*-fold cross validation and Student's *t*-test. The data is obtained on various partitions of merged COCOMO data set and Kemerer data sets incorporating data from 78 real-life software projects. PSO is employed to generate parameters of the COCOMO (Constructive Cost Model) model for each cluster of data values. The clusters and effort parameters are then trained to a Neural Network by using Back propagation technique, for classification of data. Here we have tested the model on the COCOMO dataset and also compared the obtained values with standard COCOMO model. By making use of the experience from Neural Networks and the efficient tuning of parameters by PSO operating on clusters, the proposed model is able to generate comparable results and it can be applied efficiently to larger data sets.

It goes without saying that a predictor trained on historical data can only be as accurate as the data set itself. Hence, there is a need to continue collection of data on diverse projects with wide range of attributes to construct a sizable historical database for training neural predictors. Using search based techniques to train NN; we are looking to overcome this limitation to possible extent.

Table of Contents

List of Figures

List of Tables

Chapter 1. Introduction

1.1 Introduction

Recently, the most expensive part of system software projects has been software. Human effort is majority of this software development cost, and this aspect of most cost estimation methods is main focus and provides estimates in terms of person-months.

It is very important for both customers and developers to have accurate cost estimates. Contract negotiations, proposals, monitoring, scheduling, and control request can be generated using it. Underestimation of the costs may result in management providing approval to proposed systems that might exceed allocated budgets, with under-developed functions and inferior quality, and it may fail to complete timely. Overestimating the software projects may lead to extra assets allocated to the project, or, during bidding of contract, may results in losing the contract, which can finally lead to job loss.

Accurate cost estimation is important because:

- It supports to prioritize and classify the development projects with respect to an comprehensive business plan.
- It is used to identify the resources to be used for the project and how well these assets will be utilized.
- Impacts of changes are assessed and re-planning is facilitated.
- Control and management of project is easier when resources are compatible to actual needs.
- Customers expect least deviation between actual development costs and estimated costs.
- Software cost estimation involves the identification of one or more of the following parameters:
- Duration of projects in calendar time
- Effort which is usually measured in person-months
- Cost in dollars

Most cost estimation models attempts to generate an effort estimation, which can then be mapped into project duration and cost. Even though effort and cost are closely related, they may not be necessarily mapped by a simple equation. Effort is mostly measured in person-months of

the analysts, developers and project managers. The effort estimate can be mapped into a dollar cost figure by evaluating an average salary per unit time of the staff involved, and then multiplying this by the estimated effort needed.

Practitioners have struggled with three fundamental issues:

- Which software size metric to use – function points (FP), lines of code (LOC), or feature point?
- Which Software cost estimation method or model should be used?
- What is measure of accuracy?

The widely followed cost estimation method is judgment of an expert. For many years, project managers have been dependent on experience and the existing industry trends as a basis to develop cost estimates. However, making estimates on expert judgment is error prone:

- It is tricky to find highly experienced estimators each time for a new project.
- This approach is unrepeatable and the means of deriving an estimate are not very specific.
- Relationship between system size and cost size is non linear. Cost tends to increase exponentially with respect to size. The expert evaluation method is relevant only in case of historical similarity of projects.
- Budget modifications by management targeted at minimizing over-estimates make experience and data from previous projects questionable.

In the last few decades, several quantitative software cost estimation models have been designed. It ranges from A*nalytical* models such as those in [2] [3] to Empirical models such as Boehm's COCOMO models [1]. In *empirical models,* data from previous projects are used to evaluate the current project and basic formulae are derived from analysis of the available database. On the other hand, mathematical formulae based on global assumptions are used by analytical model, like developer's problem solving rate and the number of problems available.

Majorly size estimation is based on size measures such as LOC and FP, which are used to derive cost models. Size estimation accuracy is proportional to cost estimation accuracy. However there are common drawbacks of size measurements, an organization can make good use of any one, as long as the counting method which is used is consistent.

A good software cost estimate should have the following attributes [4]:

- Based on a well-defined and tested software cost model.
- Supported and Conceived by the project manager and the software development team.
- It is based on project experience of relevant databases (similar processes, similar technologies, similar environments, similar people and similar requirements).
- All stakeholders accept it as realizable
- Defined in sufficient detail so that learning of related key risk areas is developed and the possibility of success is objectively evaluated.

Historically, Software cost estimation has been a major challenge in software development. Several points recognized for this difficulty are:

- Lack of cost measurement based historical database
- Software development involving many inter-related factors, which affect development effort and productivity, and whose relationships are not well understood
- Dearth of trained estimators with the required knowledge
- On poor estimate associated penalty is often little

1.2 Motivation of Work

Reasonably accurate prediction of software development effort has a profound effect on all stages of the software development cycle. Underestimates of resource requirements for a software project lead to:

(a) Underestimation of the cost;

(b) Unrealistic time schedule;

(c) Considerable work pressure on the engineers; and

(d) Compromises in development methodology, documentation and testing.

On the other hand, overestimates are likely to cause:

(a) A lost contract due to prohibitive costs;

(b) Over allocation of engineers to the project leading to constraints on other projects;

(c) Low productivity levels of engineers; and

(d) Easy-going work habit in the organization.

Resource requirement prediction for software projects is, therefore, an active research area. Various conventional model-based methods have met with limited success, whereas, intelligent prediction using neuro-computing has proven its worth in many diverse application areas [5]. McCullagh [6] have used neural network (NN) to estimate rainfall in Australia and have reported results superior to conventional model-based approach. As compiled by [7] NN predictors are playing major roles in diverse applications and are being successfully applied to load forecasting, medical diagnosis, communications, robot navigation, software production etc.

Recently, software engineers have started using NNs in various stages of software production with significant success.

1.3 Basic of Work

Two Hybrid Search Based Method of software cost estimation proposed in this thesis are briefly summarized below:

A. This thesis explains a new genetically trained neural network (NN) predictor trained on historical data. We demonstrate substantial improvement in prediction accuracy by the neuro-genetic. The superiority of this new predictor is established using n-fold cross validation and Student's t-test on various partitions of merged COCOMO and Kemerer data sets incorporating data from 78 real-life software projects.

B. In this methodology we have proposed a Particle Swarm Optimization (PSO) technique which operates on data sets clustered using the K-Means clustering algorithm. PSO is employed to generate parameters of the COCOMO model for each cluster of data values. The clusters and effort parameters are then trained to a Neural Network by using Back propagation technique, for classification of data. Here we have tested the model on the COCOMO 81 dataset and also compared the obtained values with standard COCOMO model. By making use of the experience from Neural Networks and the efficient tuning of parameters by PSO operating on clusters, the proposed model is able to generate better results and it can be applied efficiently to larger data sets.

Chapter 2. Literature Survey

2.1 Software Effort Estimation: A Survey of Current Practices

Before 1970, Trial and error thumb rules or some algorithms were used for effort estimation [1]. Construction of Computerized Software estimation tools was done in1970 as it was a crucial period to predict the expenses and plan for software development. When developing large software systems many difficulties were experienced. Around mid 1970's the first automated s/w approximation tool which was developed was flesh. The prototyping composite model is COCOMO (Constructive Cost Model) developed by Barry Boehm and is portrayed in book Software Engineering Economics [1]. In 1975, A new approach based on Function Point Analysis was developed for size estimation and development effort [8], based on five different features named as Inputs, Output, Logical Files, Inquires, Interfaces. Putnam [9] introduced SLIM (Software Life Cycle Model) , based on Norden Rayleigh Curve ,to US-Market in 1979. In 1983, DOD (U.S. Department of Defense) introduced Ada Programming language to build Ada-COCOMO model which reduced developing cost of large systems [10]. In 1981, Dr. Barry Boehm, through his book ―Software Engineering Economics [1], introduced the essential algorithms of Constructive Cost Model (COCOMO).

During the same period, an article on FPA method was published by Albrecht, which emphasized the rules for rating the complexity of software. In 1982, Tom de Marco developed a independent functional metric that inherited few of the features of Albrecht's function point. He published a book, controlling software projects, for introducing this metric. In 1983, British software estimating researcher Symons [11] introduced a Mark II function point metric. In 1984 function point metric was revised majorly by IBM and the revised version became basis of today's function points. 1985, Jones [12] extended the concept of computationally complex Function Point algorithms. In 1986, Fast developing utilization of Function Point Metrics, IFPUG (International Function Point Users Group) was founded in Toronto, Canada. In 1990, Barry Boehm, at college of Southern California started to revise and expand the idea of original COCOMO model. In 1991, Due to expanding need of creating control program advancements, various techniques and tools were created by Genuchten and Koolen [13]. In year 1993, newer COCOMO 2.0 version was introduced which was popularized in 1994 [14]. In 1994, Banker et al.[15], thought software development as a economic production process as that it was helpful for expense estimation and profit assessment purposes. In 1996, from the early system specifications

Cockcroft [16] found accurate size estimations. In 1997, accuracy techniques were focused on and there was review of existing models. In 1998, to give predictions of the resources (effort, time, people and cost) [17], a new model called MARCS was constructed by Chatzoglou. In 1999, Genetic Programming (GP) was explored by Dolado [18] for possible cost functions. In 2001, new methodology was envisaged which was building on analogy based reasoning and it utilized linguistic quantifiers for effort estimation [19].

2.2 New Estimation approaches using Machine learning Techniques

A variety of machine learning (ML) methods have been used to predict software development effort. Good examples include Artificial Neural Nets (ANNs) [20] [21], Case Based Reasoning (CBR) [22, 23] and Rule Induction (RI) [20]. Hybrids are also possible, e.g. Shukla [24] reports better results from using an evolving ANN compared to a Standard back propagation ANN. Dolado and others [18, 25] analyze many aspects of the software effort estimation problem and present encouraging results for a genetic programming (GP) based estimation using a single input variable. Burgess and Lefley [26] also had some success using GP based estimation. One characteristic to all ML methods is the need for training data. However, recent software engineering research has found that shared or public data sets are much less effective than restricting the prediction system to potentially very few local cases [27, 28]. The issue at stake is whether a company can improve prediction accuracy by incorporating results from other companies. Generally the more data available to a learner, the better it can model behavior. However, no matter how good the control, some metrics are likely to be measured differently across companies and the working environments may have a marked difference. Thus there will be some distortion of the models accuracy. The results reported by [27, 28] for non-evolutionary models found the larger data sets to provide less accurate estimates. This suggests that companies should only use their own data, assuming they have sufficient examples close to a new case to make an estimate. Their research used models based on regression, robust regression, Classification & Regression Tree (CART), stepwise Analysis of Variance (ANOVA) and analogy based estimation. Genetic programming offers an evolutionary solution to estimation problems that may better take into account the source of data. For example a variable indicating in house or

external could be used as a multiplier to ignore data from outside sources and so has the potential to build a prediction system at least as accurate as one based on only internal data.

2.2.1 Genetic Algorithm

Darwin's Natural Selection Theory shows that individuals that better adapt to the environment have a greater chance of surviving and passing their genetic characteristics to their offspring. Genetic Programming (GP) is the use of the Natural Selection Theory in computers, to automatically generate programs. It was presented by Koza [29], based on the idea of Genetic Algorithms introduced by Holland [30].

Instead of a population of beings, in GP there is a population of computer programs. The main goal is to naturally select the program that better solves a given problem. In order to do that, the algorithm starts with a random population and, generation after generation, applies genetic operators that simulate the evolution process.

The first step of the evolution process is to randomly generate an initial population. Then, the algorithm enters a loop that is executed, ideally, until a desired solution is found. This loop consists of two major tasks:

- evaluation of each program, by the use of a special heuristic function that shows how close each one is to the ideal solution; and
- creation of a new population by selecting individuals based on their fitness and by applying the basic genetic operators: reproduction, crossover and mutation. Each time this loop is executed, a new generation of computer programs is created.

In real life, the evolution process is never-ending, but in computing, time and resources are limited, making it necessary to establish a termination criterion that will interrupt the process. The following topics focus on each major task of the algorithm.

- Program Structure: A tree is the most straightforward structure for representing programs in GP. Each node within the tree can either be a function or a terminal. A terminal has its own value, while a function has to be evaluated considering its parameters. The user provides functions and terminals set according to the problem.

- Fitness Function and Selection: In nature, individuals are selected based on how well they fit to the environment. In GP the entity that reflects this degree of adaptation is the fitness function. The programs that better solve the problem at hand will receive a better fitness value, and will consequently have a better chance of being selected. The choice of a fitness function and an evaluation method depends on the problem. An adequate choice is essential to provide good results.

- Genetic Operators: Once the individuals are selected, it is time to apply one of the three basic genetic operators:
  1) Reproduction: an individual is replicated to the next generation;
  2) Crossover: two programs are recombined to generate two offspring potentially different; and
  3) Mutation: a new sub-tree replaces a randomly selected part of a program.

- Parameters: The behavior of the algorithm is determined by a set of parameters that, among other things limit and control how the search is performed. Some of them are: genetic operators rates (crossover rate, mutation rate), population size, selection rate (tournament size), maximum depth of the individual, etc.

Application of GP in various types of Cost Estimation

A) Analogy weights Optimization [31] by genetic algorithm for software effort estimation: An analogy-based software effort estimation model is the process of identifying one or more historical projects which are similar to the project to be developed, and then deriving an estimate from them.
B) Dolado [18] applied an extension of GA to the problem of software size estimation to investigate software size functions. Equations derived from this evolutionary method can be considered an alternative to multiple linear regressions.
C) Dolado [32] also applied this method to the problem of software cost estimation for exploring possible software cost functions. The GA explores a larger set of potential equations without

some assumptions about data distribution, instead deriving equations according to fitted values only.

D) Burgess and Lefley [26] extended this idea to richer models, requiring larger populations and much longer learning lifetimes. The proposed method provides significant improvements in accuracy when compared to other methods

E) Shukla [24] applied GA to neural network predictor in order to improve estimation capability. The hybrid method is less sensitive to weight initializations in neural network and has excellent generalization ability when trained with historical data.

2.2.2 Neural Network

Neural networks techniques are a mathematic model inspired in the neural structure of intelligent organisms that acquire knowledge by the experience. The network is composed by processing units, neurons, that are linked by communication paths. This connection is associated with a synaptic strength or weight value. In a neural network, the program is distributed across the network and stored at the synapses of each neuron. During the learning phase, synaptic weights and threshold values are adjusted until they yield the de- sired outputs. At the end, the obtained weight and threshold value of each neuron constitute the network's program and the solution to the problem.

- Topology: a typical topology for structuring the neurons is called as multi-layer neural network. Based on this topology, the layer from which the output response is obtained is the output layer. Intermediate layers are called hidden layers because their outputs are not readily ob- servable. In this kind of net, the information flows from the input layer to the output layer without return cycles (feed-forward topology).
- Learning algorithm: there are different learning algorithms; the most known is the back-propagation learning algorithm. This algorithm has two major steps:

  The input is presented to the input layer and propagated until to the output layer, and the output is compared to the desired one and the error is calculated. The error is then used to update the weights in the output layer.

Then the algorithm continues calculating the error and com-putting new weight values, moving layer by layer backward, toward the input. The steps mentioned are repeated for each

available input and constitute an epoch. Several epochs may be necessary to reach a steady state and to obtain the solution.

- Parameters: the user has to choose the topology of the net, that is, to determine the number of layers and neurons by layer and the function associated to them. Other parameters are related to the learning algorithm and are: a termination criterion, the learning rate, the initial weight values. To start the process a set of training patterns (input and desired outputs) is necessary.

Current work using NN in various types of Effort & Cost Estimation:

A) The proposed approach consists of a hybrid artificial neuron based on framework of mathematical morphology (MM) with algebraic foundations in the complete lattice theory (CLT), referred to as Dilation-Erosion Perceptron (DEP) [33].
B) An evolutionary morphological approach for software development cost estimation by Araújo, Oliveira [33]
C) Neuro-genetic prediction of software development effort by N N Shukla [24].
D) Exploring Machine Learning Techniques for Software Size Estimation by Araújo, Oliveira [34].

2.2.3 Analogy Based Estimation

An analogy-based software effort estimation model is the process of identifying one or more historical projects which are similar to the project to be developed, and then deriving an estimate from them. A study of nearly 600 organizations reported that analogy is the most widely used estimation method in the software industry [35]. Users may be more willing to accept solutions from analogy-based systems since they are derived from a form of reasoning which is more akin to human problem solving as opposed to the somewhat arcane chains of neural nets. This advantage is particularly important if the systems are to not only be deployed but also be analyzed to determine the reasoning processes behind them. Most of the existing analogy-based software effort estimation models adopt unweighted similarity measures for each effort driver [36, 37]. Essentially, the relevant effort drivers should be given significant weights in similarity measures. Nevertheless, applying analogy to estimate software development effort poses a problem in determining what effort drivers are available and which ones are significant. When designing a new analogy-based system, a model builder should first identify the effort drivers of a

software project that are believed to be significant when determining the similarity of software projects [38]. The weight of an effort driver needs to be increased when such an effort driver is significant in the process of determining the similarity between a pair of projects. In this regard, one strategy for specifying significant effort drivers is to build a learning mechanism whose algorithm can derive the optimal importance of various effort drivers.

Genetic algorithm (GA) is a searching technique based on the mechanism of natural evolution of species. It has been used for solving optimization problems in many fields [39]. In the present paper, we evaluate the potential benefits of applying GA to analogy-based software effort estimation models. GA is used in analogy learning processes to derive suitable effort driver weights for similarity measures. The present paper investigates three different weighted analogy methods—the unequally weighted, the linearly weighted and the nonlinearly weighted, to compare the effect on the accuracy of the software effort estimates. The unequally weighted analogy method uses GA in order to derive different values for the similarity measure weights for effort drivers.

The linearly weighted analogy method uses GA to derive the similarity measure weights using various linear equations for effort drivers. The nonlinearly weighted analogy method uses GA to examine the diverse nonlinear equations of the similarity measure weights for effort drivers. These methods are applied to derive the similarity measure weights for effort drivers in analogy-based software effort estimation models.

Related Work using Analogy Based Methodology are:

1. Optimization of analogy weights by genetic algorithm for software effort estimation By Chui & Huang [40].
2. The adjusted analogy-based software effort estimation based on similarity distances By Chui & Huang [31].

.

2.2.4 Case Based Reasoning

A number of research groups have been investigating applying CBR to software project

prediction since the mid 1990s [41, 42]. The basic approach is that each completed project is considered as a separate case and added to a case base. Each case is characterized by *n* features which might be continuous, discrete or categorical. Example features might include the number of interfaces, the level of code reuse and the design method employed. Clearly, a restriction is that these features must be known (or reliably estimated) at the time of prediction. A new project, for which a prediction is required (known as the target case), is also characterized by the same feature set and plotted in standardized *n*-dimensional feature space. Distance, usually a modified form of Euclidean distance is used to identify the most similar cases to the target and these, since they have known values for effort, etc., are used as the basis of the prediction. For a thorough review of CBR the reader is referred to [43]. There have been some differences in approach, for instance Prietula *et al.* make substantial use of adaptation rules whilst our work is closer to a *k* nearest neighbor (*k*-NN) method. We believe our approach to have the advantage of being more flexible since we are not restricted to a particular set of features which is a requirement for adaptation rules. This flexibility enabled us to develop ANGEL, a software estimation CBR tool that has a shell structure that can deal with arbitrary sets and types of features. Features are re-scaled so that the influence of a feature is not related to the choice of unit. This is achieved by normalizing the using the difference between maximum and minimum observed values as a denominator. For more details [42]. In general the results have been sufficiently encouraging — we found that ANGEL performed as well or better than a stepwise regression model across 9 data sets [42] to generate significant interest.

Related Work using CBR Based Methodologies are:

1) Integration of the grey relational analysis with GA for software effort estimation by Chui & Hwang [44]
2) Search Heuristic, CBR & Software Project Effort Prediction by Krissop[45]
3) A study of mutual information based feature selection for case based reasoning in software cost estimation by Y.F. Li *, M. Xie & T.N. Goh [46]

2.2.5 Particle Swarm Optimization

Particle swarm optimization (PSO) belongs to evolutionary computation technology, in 1995, Eberhaet and. Kennedy proposed it [49], and this algorithm was derived from researching

on the predation of bird flock. Besides, PSO is a research based on population, and this population includes lots of particles where each particle represents a solution of an optimization problem. Like other evolution computation technology, these particles are always initialized randomly. This algorithm optimizes the search of swarm intelligence guidance through the collaboration and competition between particles. The equation of motion of particle insides the d dimension is showed below.

Equation (a)

$V[n] = Co * V[n] + C1 * rand * (pbest[n-1] - present[n-1] + C2 * rand(\ ) * (gbest[n-1] - present[n-1]$

(in the original method, c0=1, but many researchers now play with this parameter)

Equation (b)

$present[n] = prsent[n-1] + v[n-1]$

The first term on the right hand side of the equation above is corresponding to the momentum which represents the particles‟ current status. The second and third terms are corresponding to the intensification during the searching process. Particles are moving to the target point by analyzing self-information and group information until it satisfied the terminal condition.

The steps of PSO shows below: first, initialize the particle population randomly and initialize the location and velocity of each particle. Second, define the fitness function based on the goal of optimization problem, evaluate the fitness of all particles and update the optimal location of each particle from the population. Third, update the new population optimal location according to the fitness value. Finally, iterate the location and velocity of each particle, redo the steps above until it satisfy the iteration terminal condition of this algorithm and give the output of group optimal location.

However, when solving some complex optimal problems, PSO also has the phenomenon of premature convergence. Hence scholars try to improve the evolutionary mechanism and the optimal performance of PSO recent years. Naka proposed a HPSO method which can be applied into distribution state estimation. It uses the mechanism of natural selection, replacing the particles whose fitness values are low by using those are high in order to make them enter a more

efficient searching area. This can estimate the state of target system more precisely by comparing with the traditional PSO algorithm, and even if there may still have some measuring errors, this method can estimate the system state better than others. Huang and Mohan [50] proposed a simple and convenient micro particle swarm algorithm (μPSO), it can be competent a kind of PSO algorithm which has a huge group data by only using a small number of population when considering some high-dimensional optimization problems. Experiments indicate that its calculated amount is less than traditional PSO algorithm significantly when calculating the fitness functions. [10] Ren proposed a method which combines PSO algorithm with BP algorithm, that defines elite particles, and trains the neural network by using a hybrid cross method to give higher accuracy and faster convergence speed of network learning process.

Related Work using PSO are:

1) Applying Particle Swarm Optimization to Estimate Software Effort by Multiple Factors Software Project Clustering by Lin & Tzeng [51]
2) CPN-A Hybrid Model for Software Cost Estimation by Hari [52]
3) Improving the Accuracy in Software Effort Estimation Using ANN Model Based on PSO by Zhang Dan [53]
4) Evaluating software cost estimation models using PSO and fuzzy logic for NASA projects: a comparative study by Sheta, Ayesh & Rine

2.2.6 Hybrid Methodology

Recent years, the software industry is growing rapidly and people pay more attention on how to keep high efficiency in the process of software development and management. In the process of software development, time, cost, manpower are all critical factors. At the stage of software project planning, project managers will evaluate these parameters to get an efficient software develop process. Software effort evaluate is an important aspect which includes amount of cost, schedule, and manpower requirement. Hence evaluate the software effort at the early phase will improve the efficiency of the software develop process, and increase the successful rate of software development.

Many models are developed with combination of one of the above ML methods

A) ANN with PSO

Artificial neural network (ANN) prediction model that incorporates with Constructive Cost Model (COCOMO) which is improved by applying particle swarm optimization (PSO), PSO-ANN-COCOMO II, to provide a method which can estimate the software develop effort accurately. The modified model increases the convergence speed of artificial neural network and solves the problem of artificial neural network's learning ability that has a high dependency of the network initial weights.

B) ANN with GP

The idea is to obtain an equation to estimate LOC for each mentioned approach (FP and NOC), using Genetic Programming (GP) and Neural Networks (NN). The main motivation to choose these techniques for this task is their capability of learning from historical data, discovering a solution with different variables and operators, being robust with respect to noisy data.

2.3 Process of estimation

Effort estimation is an important part of the planning process. For example, the project plan is derived using cost estimate in the top-down planning approach:

1. The project manager build a description of the overall functionality, size, process, environment, people, and quality required for the project.
2. Using software cost estimation model, a macro-level estimate of the total effort and schedule is designed.
3. Partitioning of the effort estimate into a top-level work breakdown structure is done by project manager. In order to form a project plan, he also partitions the schedule into major milestone dates and identifies a profile of its manpower.

The real cost estimation process is defined in seven steps [5]:

1. Objectives of cost-estimation is established
2. A project plan is developed for required data and resources
3. Software requirements are freeze

4. All feasible Details about Software system was worked out.
5. Various independent cost estimation techniques were applied collectively to capitalize on their overall strengths
6. Measure the similarity between different estimates and repeatedly iterate the estimation process
7. After the project has started, monitor its actual cost and progress, and feedback results to project management

Irrespective of which estimation model is chosen, attention must be paid by the users to the following points to get the best results:

- coverage of the estimate (some models generate effort for the full life-cycle, while others do not include effort for the requirement stage)
- tuning and assumptions of the model
- Influence of the estimates to the different parameters of models
- deviation of the estimate with respect to the actual cost

2.4 Estimation Methods

Two major types of cost estimation methods are non-algorithmic and algorithmic. Algorithmic models vary extensively in mathematical sophistication. Some are based on simple arithmetic formulas using such summary statistics as standard deviations and means [9]. Others are based on regression models [38] and differential equations [30]. To improve the accuracy of algorithmic models, there is a need to adjust or calibrate the model to local circumstances. These models can't be used over-the-shelf. Even with calibration the accuracy can be mixed quietly.

Non-algorithmic methods are briefed below:

2.4.1 Non-algorithmic Methods

Analogy costing: This method need one or more projects which are completed and are similar to the new project and derives the estimation using the actual costs of previous projects through reasoning by analogy. Analogy based estimation can be done either at the total project level or at each subsystem level. The total project level has the advantage that all cost components of the system will be considered while the subsystem level has the advantage of

providing a more detailed assessment of the differences and similarities between the new project and the completed projects. The key strength of this process is that estimate is based on real project experience. However, it is not very clear up to what extend the previous project is actually effective on the constraints, environment and functions to be performed by the new system. In [33] Positive results and a definition of project similarity in term of features were reported.

Expert judgment: This method includes consulting experts. The experts provide estimates applying their indigenous methods and experience. Expert-consensus mechanisms such as Delphi technique or PERT will be used to resolve the inconsistencies in the estimates.
The Delphi technique works as follow:

1) The coordinator provides each expert with a specification and a form to make estimation records.

2) Form is filled by each expert individually (without discussing with others) and questions are allowed to coordinator only.

3) A summary of all estimates provided by experts is prepared (including mean or median) on a form requesting another round of the experts' estimates and the objective for the estimates.

4) Repeat steps 2)-3) as many rounds as appropriate.

A derivation of the Delphi technique proposed by Boehm and Farquhar [5] seems to be more effective: Before the estimation, a group meeting involving the coordinator and experts is arranged to discuss the estimation issues. In step 3), the experts do not need to give any rationale for the estimates. Instead, after each round of estimation, the coordinator calls a meeting to have experts discussing those points where their estimates varied widely.

Price-to-win: The software cost is calculated to be the best price to get the project. The estimation basis is customer's budget instead of the software functionality. For example, if an approximate estimation for a project costs 100 person-months but the customer can only afford 60person-months, it is norm that the estimator is asked to modify the estimation to fit 60 person-months' effort in order to win the project. This is again not a good practice since it is very likely to will lead to a delay of delivery or cause overtime for the development team.

Parkinson: Using Parkinson's principle "work expands to fill the available volume" [28], Available resources determine the cost (not estimated) instead of an objective assessment. In case software has to be delivered in 11 months and 5 people are allocated, the calculated effort is

estimated to be 55 person-months. Although it sometimes gives good estimation, this method is not recommended as it may provide very unrealistic estimates. Also, this method does not promote good software engineering practice.

Top-down: An overall cost estimate for the system is derived from global properties, using either algorithmic or non-algorithmic methods. The total cost can then be split up among the various components. This approach is more suitable for cost estimation at the early stage.

Bottom-up: This approach is the opposite of the top down method. In this approach, each software system component is estimated separately and the results aggregated to produce an overall system estimate. The requirement for this approach is that an initial design must be in place that indicates how the system is decomposed into different components.

2.4.2 Algorithmic methods

The algorithmic methods are based on mathematical models that develop cost estimate as a function of a number of variables, which are considered to be the major cost factors. Any algorithmic model has the form:

Effort = f(x1, x2, …, xn)

where {x1, x2, …, xn} denote the cost factors.

The existing algorithmic methods differ in two aspects: the form of the function f and the selection of cost factors. Firstly cost factors used in these models are discuss, then characterize the models according to the form of the functions and whether the models are empirical or analytical.

2.4.3 Cost Factors

In addition to software size, there are many other cost factors. The most comprehensive set of cost factors are proposed and used by Boehm et al in the COCOMO II model [14]. These cost factors can be divided into four types:

Product factors: required reliability; database size used; product complexity; required reusability; documentation match to life-cycle needs;

Computer factors: execution time constraint; main storage constraint; computer turnaround constraints; platform volatility;

Personnel factors: analyst capability; programming capability; application experience; platform experience; personnel continuity; language and tool experience;

Project factors: multisite development; use of software tool; required development schedule. The above factors are not necessarily independent, and most of them are hard to quantify. In many models, some of the factors appear in combined form and some are simply ignored. Also, some factors take discrete values, resulting in an estimation function with a piece-wise form.

2.4.4 Multiplicative models

These models use the coefficient values that are best for the completed project data. Following models are considered to be multiplicative model.

Walston & Felix Model This model was developed by Walston and Felix at IBM Federal Systems to measure the rate of production of lines of code. The model estimates the total man-months of effort as a function of the line of code to be produced [58] and also estimates pages of documentation, duration of development in calendar months, average staff size and cost of development with respect to computer time [59]. The model was a result of statistical analysis of historical data derived from a database of 60 different projects that ranged from "…4,000 to 467,000 LOC, and from 12 to 11,758 person-month effort… 28 high-level languages, and 66 computer systems and were classified as small less-complex, medium less-complex, medium complex, and large complex systems"[81].

Based on their collected data they investigated 68 variables that may affect the productivity measures. Out of those 68 variables, they selected most significant 29 factors that are associated with productivity. These factors were used to calculate the productivity index, which was computed in a linearly regression fashion to obtain an equation for estimating productivity of new projects [81][31]. Out of the nine equations used by this model, one relationship of the form

$E = aL^{b}$ was used to estimate effor*t*, where *L* is the number of lines of code, in thousands, and *E* is the total effort required in person-months. The equation obtained after deriving values for parameters *a* and *b* was [11][31]. $\boldsymbol{E = 5.2\, L^{0.91}}$

This model has not provided a distinction between comments and program instructions, consequently the effort for both was assumed to be same. Limited availability of this model has restricted its use or recalibrations across the organizations. The reliability of this model is questioned by different researchers and due to other weaknesses this model is probably not in practice any more.

Doty Model associates with US air force sponsor ship incorporated this manual model in 1976/77 [11] [59]. This model is used to compute total person-months of development effort, development cost, and time, overhead cost of computer time, documentation and travel. Four application areas (i) command and control (ii) scientific (iii) business (iv) and utility are covered with the help of different equations. 14 environmental factors are also proposed in this model [table III in [72]], however their use is optional [59]. The expression used to compute effort in man Months MM for any general application is discussed as [72].

$MM = 5.288\ (KDSI)^{1.047}$ for $KDSI \geq 10$

$MM = 2.060\ (KDSI)^{1.047}$ x (effort Multipliers F_i) for $KDSI < 10$.

2.4.5 Power function models

Power function models have the general form:

Effort = a×S x b ---- Equation 1

where S is the code-size, and a, b are (usually simple) functions of other cost factors. This class contains two of the most popular algorithmic models in use, as follows:

COCOMO (Constructive Cost Model) models

This family of models was proposed by Boehm [4, 5]. The models have been widely accepted in practice. In the COCOMOs, the code-size S is given in thousand LOC (KLOC) and Effort is in person-month.

Basic COCOMO. This model uses three sets of {a, b} depending on the complexity of the software only:

(1) for simple, well-understood applications, a = 2.4, b = 1.05;

(2) for more complex systems, a = 3.0, b = 1.15;

(3) for embedded systems, a = 3.6, b = 1.20.

The basic COCOMO model is simple and easy to use. As many cost factors are not considered, it can only be used as a rough estimate.

Intermediate COCOMO and Detailed COCOMO. In the intermediate COCOMO, a nominal effort estimation is obtained using the power function with three sets of {a, b}, with coefficient a being slightly different from that of the basic COCOMO:

(1) for simple, well-understood applications, a = 3.2, b = 1.05

(2) for more complex systems, a = 3.0, b = 1.15

(3) for embedded systems, a = 2.8, b = 1.20

Then, fifteen cost factors with values ranging from 0.7 to 1.66 (see Table 1) are determined [5]. The overall impact factor M is obtained as the product of all individual factors, and the estimate is obtained by multiplying M to the nominal estimate.

While both basic and intermediate COCOMOs estimate the software cost at the system level, the detailed COCOMO works on each sub-system separately and has an obvious advantage for large systems that contain non-homogeneous subsystems.

COCOMO II. Perhaps the most significant difference from the early COCOMO models is that the exponent b changes according to the following cost factors: precedentedness, development flexibility, architecture or risk resolution, team cohesion, and process maturity. Other differences include newly added cost factors and models for solidifying software architecture and reducing risk.

Putnam's model and SLIM

Putnam derives his model based on Norden /Rayleigh manpower distribution and his finding in analyzing many completed projects [30]. The central part of Putnam's model is called software equation as follows:

$S = C_k * K^{1/3} t_d^{4/3}$

where td is the software delivery time; E is the environment factor that reflects the development capability, which can be derived from historical data using the software equation. The size S is in LOC and the Effort is in person-year. Another important relation found by Putnam is

$Effort = D_0 \times t_d^3$

Where D_o is a parameter called manpower build-up which ranges from 8 (entirely new software with many interfaces) to 27 (rebuilt software).

Putnam's model is also widely used in practice and SLIM is a software tool based on this model for cost estimation and manpower scheduling.

2.4.6 Model calibration using linear regression

A direct application of the above models does not take local circumstances into consideration. However, one can adjust the cost factors using the local data and linear regression method. We illustrate this model calibration using the general power function model: Effort = a×Sb. Take logarithm of both sides and let Y = log(Effort), A = log(a) and X = log(S). The formula is transformed into a linear equation:

$Y = A + b \times X$

Applying the standard least square method to a set of previous project data {Yi, Xi: i =1, …,k}, we obtain the required parameters b and A (and thus a) for the power function.

2.4.7 Discrete models

Discrete models have a tabular form, which usually relates the effort, duration, difficulty and other cost factors. This class of models contains Aron model [3], Wolverton model [39], and Boeing model [4]. These models gained some popularity in the early days of cost estimation, as they were easy to use.

2.4.8 Other models

Many other models exist and the following have been used quite successfully in practice.

Price-S is proprietary software cost estimation model developed and maintained by RCA, New Jersey [27]. Starting from an estimate of project size, type and difficulty, the model computes project cost and schedule.

SoftCost relates size, effort and duration to address risk using a form of the Rayleigh probability distribution [36]. It contains heuristics to guide the estimators in dealing with new technology and complex relations among the parameters involved.

Chapter 3. Research Methodology

3.1 Neural Network Training

For effort prediction we have used a multi-layered feed forward NN with 39 input neurons, each neuron corresponding to one of Boehm's features [1], hidden layers of neurons to develop the desired mapping and 1 output neuron corresponding to the predicted effort in person-months. Our NN model uses a weight vector **W**, indexed over the arcs **A** of the network **N.** .**V**; **A**.; which contains no directed cycles. Here **V** and **A** are the set of vertices and arcs in the graph defining the NN. We consider real-valued weights, thresholds and exemplars, and sigmoid activation function. The NN must learn to compute a vector-to-vector mapping **H** (**W**), defined as follows:

When the immediate predecessors of a node v have computed their outputs, then v computes its output according to the rule:

$$Xv = 1/(1 + e^{-wv}) \qquad Wv = \sum_{u,v\in A} Wuv.Xu \qquad (1)$$

.

We use a pseudo node with output fixed at +1, and weights connecting it to all hidden and output units to model the neuron threshold parameter. With this arrangement, the dimension of the input vector is increased by one, the last component being +1. When the neurons compute the weighted sum of inputs, the threshold effect is modeled by the last element of the weight vector, which is the weight of the link connecting the pseudo node to neuron. Thus, the *augmented* weight vector **W** includes the threshold parameters.

Our notations generally follow Ref. [54] . Given a network **N** .= (**V**;**A)**. and weight vector **W** defined on the arcs **A**, we use **N**(**W**) to denote a network with weight vector **W**, and *N* to denote the family (or all architectures) of possible networks **N**(**W**) indexed over all possible weight vectors **W**. **V** denotes the set of nodes in the input layer. **V**1, **V**2, **V**3,…, etc. denote sets of nodes in higher layers (hidden, or output layers).

Training set T for an architecture N is defined as a finite set of ordered pairs:

$$T = \{(\mathbf{X}t\ ;\mathbf{Y}t.) |\ t\ .\ 1;\ 2;\ 3;\ldots;\ T\} \qquad (2)$$

Here, $\mathbf{X}_t$ is the input vector and $\mathbf{Y}_t$ is the corresponding output vector.

In general, $\mathbf{X}_t$ and $\mathbf{Y}_t$ may have different dimensions. For example, in the present case, software effort prediction input vector has dimension 39, while the output vector has dimension 1, (i.e. it is a scalar). The training set contains the input–output patterns from a historical database of past projects.

Let $\mathbf{Y}t = \mathbf{H}.(\mathbf{X}t\ ;\mathbf{W})$ denote the output actually computed by the network corresponding to the input vector $\mathbf{X}t$. We define a residual vector with respect to t as:

$$\boldsymbol{R(W)} = (\boldsymbol{R}t(\boldsymbol{W})|t = 1; 2; 3;\ldots; T. \qquad (3).$$

$$\boldsymbol{R}t(\boldsymbol{W}) = . \boldsymbol{Y}t - \widehat{\boldsymbol{Y}}t \quad for \quad t = 1; 2; 3;\ldots; T\ . \qquad (4)$$

We define an error function using L2 norm of the residue:

$$E(W) = \frac{1}{T}\sum_{t=1}^{T} ||R_t(W)||_2^2 \qquad (5)$$

This is our performance metric or objective function. The subscript 2 in expression (5) denotes the fact that we are using L_2 or Euclidean norm to formulate the prediction error. The supervised learning problem for our NN model then becomes:

$$\min_{W} E(W) \qquad (6)$$

That is, minimize $E(\mathbf{W})$ with respect to $\mathbf{W}$; or, find an optimum weight vector $\mathbf{W}^*$ that globally minimizes $E(\mathbf{W})$ over the training set. If N contains sufficient number of hidden units to develop the required internal representation, then we say that $\mathbf{W}^*$ is *exact* with respect to the training set *t*. Then, $E(\mathbf{W}) = 0$; or, equivalently,

$$\mathbf{H}.(\mathbf{X}t\ ,\mathbf{W}^*) = \mathbf{Y}t\ ;\ t\ .\ 1; 2; 3;\ldots; T\ . \qquad (7)$$

To efficiently solve the global optimization problem (Eq. (6)), and thereby train the NN to predict development effort accurately, we utilize genetic algorithm (GA), which is a time-tested global optimize so successfully used by nature for the evolution of species. In the next section, the basic

principle of genetics-based optimization algorithm is presented. This is followed by a section describing the use of GAs to train NN to learn the diagnosis problem.

3.2 Genetic algorithm— A global optimizer

GA are based on biological evolutionary theories to solve optimization problems . GA comprises of a set of individual elements (the population) and a set of biologically inspired operators. According to evolutionary theories, only the most suited elements in a population are likely to survive and generate offspring's, and transmit their biological heredity to the new generations . GAs are much superior to conventional search and optimization techniques in high-dimensional problem spaces due their inherent parallelism and directed stochastic search implemented by recombination operators.

GA operates through a simple cycle of three stages:
1. Randomly create an initial population of individuals.
2. Perform the following sub steps iteratively for each generation until a termination condition is fulfilled:
2.1. Evaluate the fitness of each individual in the population and save the best individual of all preceding populations.
2.2. Create a new population by applying the genetic operators:
2.2.1. Selection;
2.2.2. Crossover;
2.2.3. Mutation;
2.3. Replace the current population by the new population.
3. Output the individual with the best fitness as the optimum solution.

Selection is based on fitness, i.e. the fitter an individual the greater the chance for this individual to get selected for reproduction and contribute offspring for the next generation.

Crossover operator takes two chromosomes and swaps part of their genetic information to produce new chromosomes.

Mutation is implemented by occasionally altering a random bit in a string before the offsprings are inserted into the new population.

Control parameters: We can visualize the functioning of GAs as a balanced combination of *exploration* of new regions in the search space and *exploitation* of already sampled regions.
The balance, which critically controls the performance of GAs is determined by the right choice of control parameters: the crossover and mutation probabilities and population sizes. The trade-offs that arise are:

- Increasing the crossover probability increases the recombination of building blocks, but it also increases the disruption of good strings.
- Increasing the mutation probability tends to transform the genetic search into a random search, but it also helps reintroduce lost genetic material.
- Increasing the population size increases its diversity and reduces the probability that the GA will prematurely converge to a local optimum, but it also increases the time required for the population to converge to the optimal regions in the search space.

The reader is referred to Ref. [55] for several aspects of GA implementation details.

3.3 Clustering PSO Based Neural Network

The COnstructive COst MOdel (COCOMO) proposed by Boehm, is the most famous Cost Estimation Model .The COCOMO model defines the relationship between the software effort given in person-months, the size of the project given in thousands of lines of code(KDLOC) and the Effort Adjustment Factor(EAF)(Equation 3).

The mathematical formulation of the COCOMO model is given below:

$$\text{Effort} = a * (\text{size})^{b} * \text{EAF} + c. \quad (8)$$

Here a, b and c are the statistical parameters [56, 57] .

These parameter values were previously estimated by using regression analysis applied on historical data .However to account for the unpredictability in the data values, several soft computing techniques were proposed. Some of these models used fuzzy logic, neural networks, Genetic algorithms, etc. PSO is one such soft computing technique which is based on the

movements of intelligent swarms. It was implemented by observing the movements of agents (particles) in the problem space which was provided for training and then using the experience of the swarms for further testing. K means algorithm has been proposed for clustering of data values into related clusters based on the relationships between them. Neural networks is also a soft computing technique which can be used for classification of data sets. The Back propagation algorithm applied on neural networks can efficiently classify data sets based on previous learning.

In our model-CPN we have implemented a hybrid technique for estimating the COCOMO parameter values. The input data set is clustered using the K-means clustering algorithm and then PSO is applied to each cluster to find the values. The resulting training is given to a Neural Network which is then able to classify testing data into the appropriate cluster to be further evaluated by PSO.

3.4 Performance of estimation models - Fitness functions and accuracy statistics

This section describes the accuracy statistics used to evaluate models in this study, and the fitness functions used to build the models.

3.4.1 Accuracy statistics

These accuracy statistics were used to evaluate estimation models in this study:

R^2: The square of the coefficient of correlation between estimated and actual values.

MSE: Mean squared error — used as an accuracy statistic by [25].

StDev: Standard deviation of error. Recommended by as an accuracy statistic; almost identical to root mean squared error, used as an accuracy statistic in [58].

MMRE: Mean magnitude of relative error. Widely used as an accuracy statistic since its description in [59]. Median MRE is also reported here.

MMER: Mean magnitude of error relative to the estimate. Proposed in [60], argued as intuitively preferable to MMRE since at the time of estimation one wants to know what errors to expect relative to the estimate. Median MER is also reported here.

Mean: Kitchenham et al [60] recommended that accuracy should be measured in terms of *z=(estimate/actual)*. It is asymmetric and favours models that minimize overestimates (which could be a problem since over- estimates are usually less serious than underestimates). The full

distribution of z should be considered when comparing prediction systems. Median of is also reported here, as well as the mean of

1/z=(actual/estimate).

Pred(l): The proportion of estimates that are within a given percentage of the actual value. Widely used as an ac-curacy statistic since its description in [59], particularly in conjunction with MMRE. Pred(0.25) and Pred(0.50) are reported here.

Means and medians of absolute errors are also reported. Foss et al [58] recommend LSD and RSD as accuracy statistics. These are intended as estimates of the standard deviation of the error term. Neither could be used here. LSD involves logarithms of estimated values, so cannot be computed for any model that generates any negative estimates; several such models emerge in this study. RSD can only be used if estimates are based o a single predictor variable.

3.4.2 Fitness functions

The fitness functions chosen for study were selected either because they are commonly used in building estimation models, or they are commonly used in evaluating estimation models. The functions chosen were:

MSE: since it underpins ordinary regression, and is related to and SD.

LAD (least absolute deviation): an obvious alternative to ordinary regression, proposed centuries ago but overshadowed by least-squares regression because it was harder to compute and lacked theoretical under- pinning's — both addressed now [1].

MRE: because it is so common as an accuracy statistic, and has also been used elsewhere as a fitness function [26].

MER and Z: since they are proposed as better alternatives to MRE.

Pred(l), since it is so widely used as an accuracy statistic.

Writing a fitness function consists of specifying how the error term is calculated for a single data point, given its known and estimated values. The GP software used here always tries to minimize the overall fitness value, so the fitness function needs to be written so that smaller values are better.

Functions are easily written for differences between estimated and actual values:

MSE: error = (estimate - actual)

LAD: error = abs (estimate - actual)

MRE: error = abs ((estimate - actual) / actual)

MER: error = abs ((estimate - actual) / estimate)

Z is trickier, since its optimum value is 1, not zero. Computing and subtracting 1 does not work, since negative values are possible and minimization will do the wrong thing. *abs(z-1)* is no help, as that is MRE. As an approximation, the following function was used:

Z: error = max(estimate, actual)/min(estimate, actual)

This is always at least 1, and it makes sense to minimize it: the closer it is to 1, the closer the estimate and actual values are to each other.

Chapter 4. Proposed Work

4.1 Methodology I – Genetically Trained Neural Network

Application of GA to NN training requires binary representation of synaptic weight parameters so that genetic operators can be readily applied. Here, two broad possibilities exist. One can use a strong representation that specifies exactly each connection, neuron etc. This will result in a large search space. On the other hand a weak representation uses an abstract description of an artificial NN, which must be translated to yield a network phenotype. This result in a much smaller search space and the NN can be restructured according to the application requirements. Strong representation schemes are good at capturing connectivity patterns within small networks and are more useful for evolution of compact architectures.

Extensive simulation results show that NN training using GAs is very sensitive to the representation schemes used. If chromosomes represent individual weights of an NN, the crossover operator may cause several good weight values to be destroyed due to string exchanges. On the other hand, if a chromosome represents *all* the weights in the NN then the crossover operator may exchange bits between weights belonging to different layers, which will again destroy some desirable computation properties learnt by a layer previously. We have introduced a new multipoint restricted crossover operator (MRX) to alleviate this problem. The details are given in the below section.

The software development effort prediction problem involves making an NN learn the underlying correlations between the 39 project features identified by Boehm and the person-month requirement of the project, as well as the correlation that may exist among the predictor variables.

Using historical data to form the training set T , the fitness function is taken as:

$$f = \frac{1}{E(W)} \tag{9}$$

Where, $E(\mathbf{W})$ is given by Eq. (5).

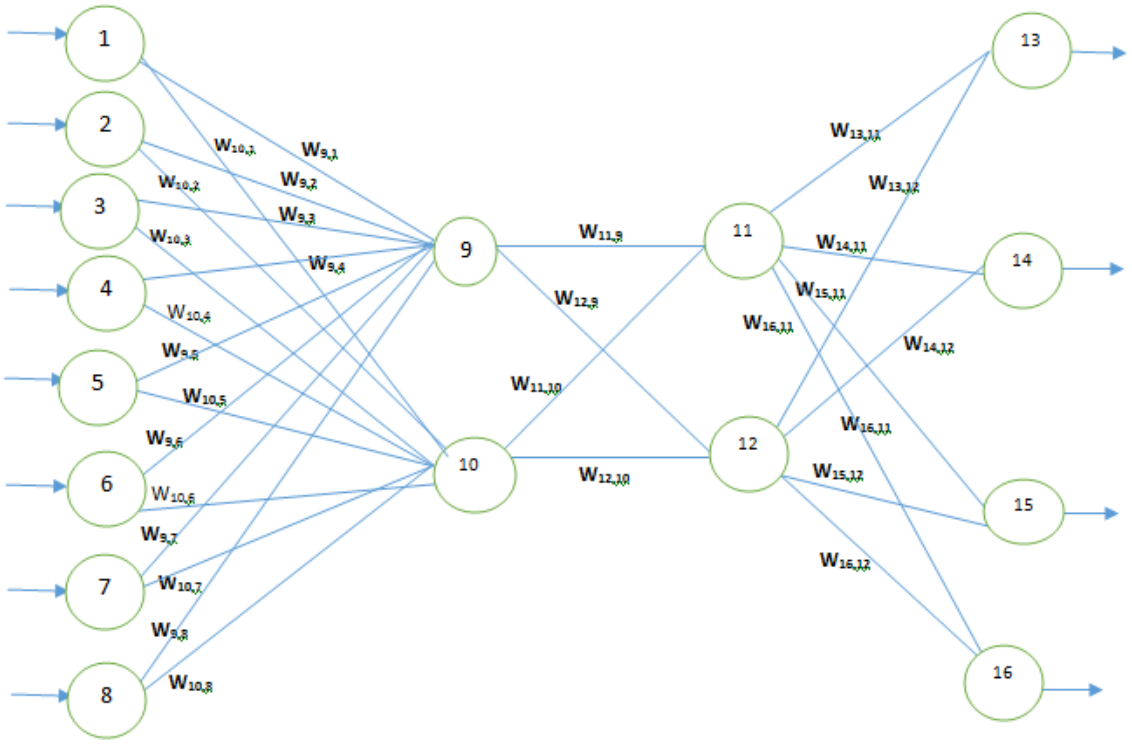

Figure 4.1 A neural network to be trained genetically

The GA proceeds to maximize the above fitness function resulting in minimization of mean square output error. A suitable range of GA parameters like population size, crossover probability and mutation probability, has been found by multiple simulation runs for the software development effort prediction problem. In simulations we add a small positive constant to the denominator of the fitness function (8) to avoid overflow as the error approaches zero.

NN to be trained genetically is shown in Fig. 1. This NN consists of 30 weights connecting various layers. The bias weights have not been shown for clarity. These 30 weights are encoded in a chromosomal string shown in Fig. 2.

A) **Encoding**: A resolution of around 10 bits per weight is necessary for acceptable performance. We have also obtained encouraging results with 10 bits per weight resolution. Each weight w_{ij} in the example of **Fig. 4.2** is encoded as a 10-bit string,

representing weights in the range [-5,5] thus giving a total chromosome length of 300 bits for this example.

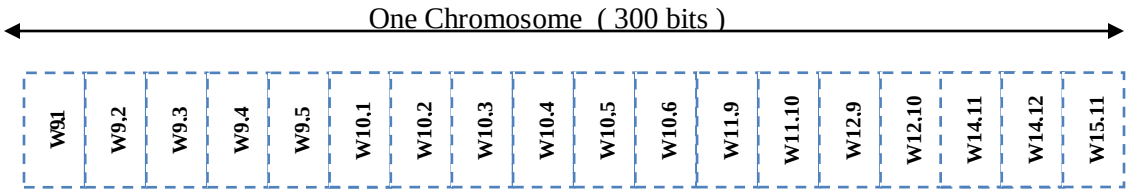

Figure 4.2 Chromosome representation of neural network in Fig 4-1

B) **Crossover**: randomly selected multiple crossover sites equal to the number of weights in the chromosomes are used but *bit exchange is restricted to corresponding weights only.* Same is explained in Fig 4.3.

C) ***Mutation:*** The crossover is followed by multi-bit mutation equal to the number of weights. As usual, the mutation probability is kept very low in comparison to crossover probability

D) **Selection**: Hybridization of ELITIST and ROULETTE WHEEL selection strategies. The elitist policy encourages preserving the best chromosomes in the subsequent generations. The first two places in the next generation were *reserved* for the fittest string of the current generation. The rest of the strings were generated using weighted roulette wheel selection scheme. Each individual of the population receives a part of the roulette wheel proportional to its fitness. Selecting an individual means spinning the roulette wheel. Individuals that occupy a bigger part of the roulette wheel have a greater chance to get selected. While the elitist component increases the selective pressure so that the search focuses on the top ranking individuals, the roulette wheel policy, being probabilistic in nature, provides population diversity.

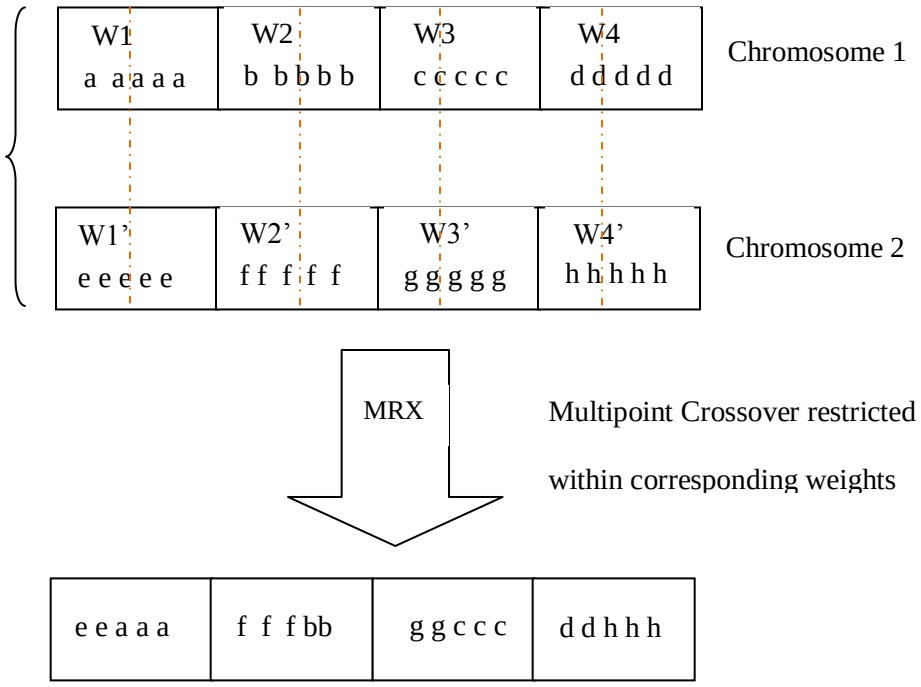

Figure 4.3 The MRX genetic operator used for evolving neural predictor

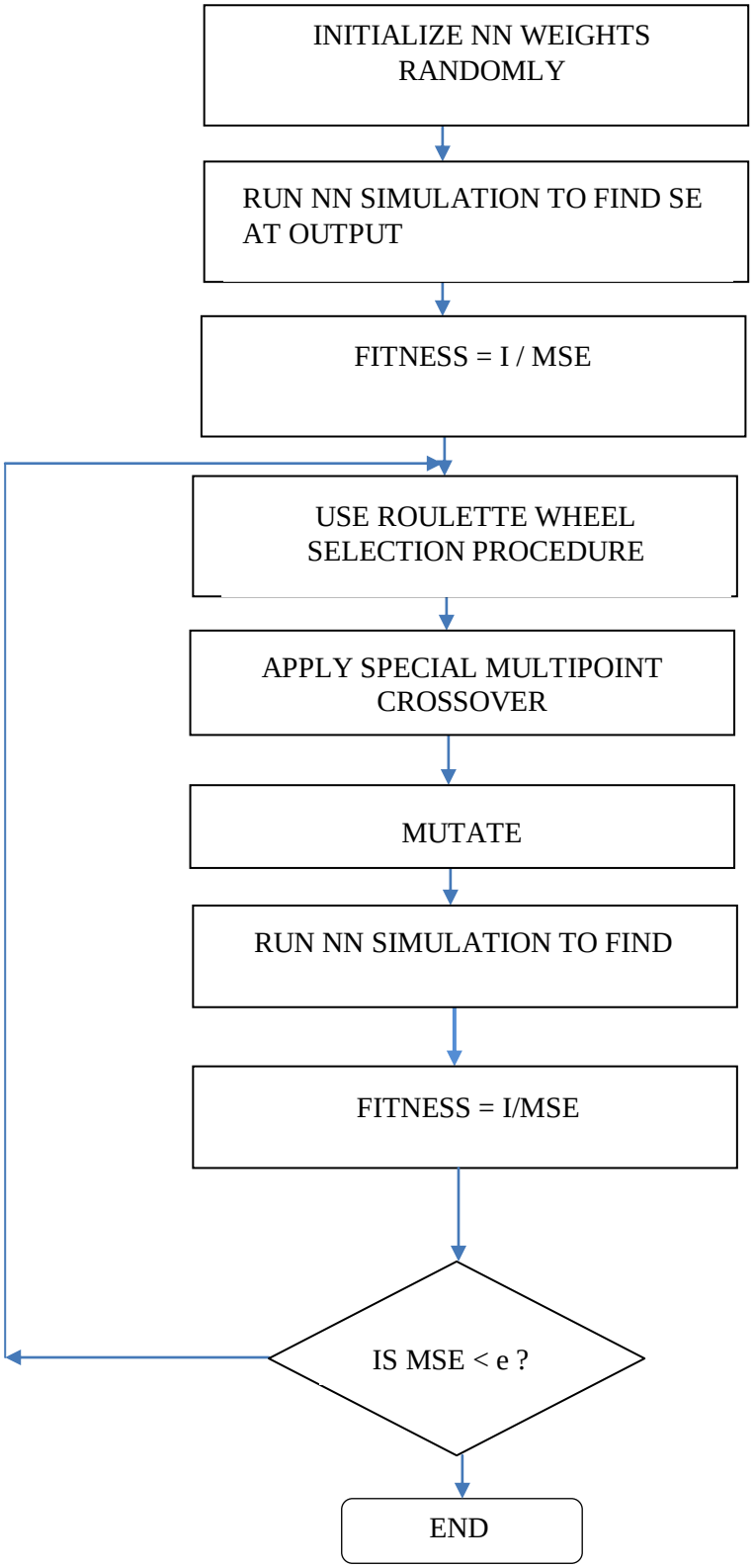

Figure 4.4: Flow Chart showing genetic evolution of NN predictor for development effort

4.2 Methodology II - Clustering PSO Neural Network

Here the methodology employed for the CPN model is described. The model uses K- means algorithm for clustering of input data set. It then implements PSO on these clusters to obtain the parameter values of the COCOMO model. The description follows:

Algorithm CAK( ):The steps described below lists out the implementation of the K-means clustering algorithm.

INPUT: Data sets consisting of N values of size, measured effort and EAF.

OUTPUT: K clusters of data values.

Step 1: Choose K values out the given N values to be the initial centroids.

Step 2: Assign each value of the data set to the cluster for which the distance between the value and the corresponding centroid is minimum. Euclidian distance formula is used for the distance evaluation.

The distance is given by the formula:

$$D(size_i, EAF_i) = \sqrt{(| size_i - size_c|^2 + | EAF_i - EAFc |^2)} \quad \text{--- (10)}$$

Here $size_i$ and EAF_i denote the value being evaluated and $size_c$ and EAF_c denote the centroids of cluster c.

Step 3:Calculate the new centroid of the cluster by finding the mean of all the data values in the cluster as:

$$\text{Centroid}(\text{sizec}, \text{EAFi}) = (\sum_{i \epsilon cluster} \frac{\text{SIZEi}}{\text{N}}, \sum_{i \in cluster} \frac{EAFi}{N}) \quad \text{- (11)}$$

.

Step 4: Repeat steps 2- 4 until the values obtained denote stable cluster values .These clusters are the obtained clusters.

Step 5: Stop.

Algorithm PSOIW( ):The steps described below represent the implementation of PSO on the clusters of data values obtained from the above K means algorithm.

INPUT: K clusters of data values containing the software project size, EAF and the measured effort.

OUTPUT: Optimized COCOMO parameter values for each cluster.

Step 1: *Initialization:* Initialize particles with random positions and velocity vectors of tuning parameters. Specify the range of velocity between [- Vmax, Vmax].

Step 2: *Evaluation of Fitness Function:* For each particle position evaluate the fitness function. The fitness function here is Mean Absolute Relative Error (MARE). The objective in this method is to minimize the MARE by selecting appropriate values from the ranges specified in step 1.

Step 3: *Finding the Pbest* – Personal best: If fitness (x) better than fitness (Pbest) then: Pbest = x. Here the P best is determined for each particle by evaluating and comparing measured and estimated effort values of the current and previous parameters values.

Step 4: *Finding the G_{best} (global best):* Set the best of 'P_{best}' as global best – G_{best}. The particle value for which the fitness function shows the best values is chosen as the G_{best} particle.

Step 5: *Update values:* Update the velocity and positions of the tuning parameters with equations (a) & (b) in Section 2.2.5.

Step 6: Repeat steps 2 to step 5 until "particles exhaust".

Step 7: Give the G_{best} values as the optimal solution.

Step 8: Stop.

Chapter 5. Results & Analysis

Results of both methodologies are explained below and also comparison of GANN & CPN methodology is described:

5.1 Results of Clustering PSO Neural Network:

For the purpose of experimentation initially 35 values were used for training. These values are clustered by using the algorithm CAK(). These training data values are shown in Table 1.Once the clusters were generated the PSOIW() was applied to each of the cluster to obtain the effort estimation parameters a, b and c. The cluster wise parameter values obtained are given below.

Cluster 0: a=1.145552; b=1.468810 ; c= -9.406167.

Cluster 1: a= 0.058581; b=2.214696; c= -2.090077.

Cluster 2: a= 4.297179 ; b=0.956393 ; c= -1.595891.

From these parameter values, the estimated efforts were calculated by using the equation (**3**). The estimated effort values(CPN-EE), standard COCOMO effort values (COCOMO) ,size(S),Cluster number (Cl.No.) ,Measured Effort(ME) and EAF values are listed out in **Table 5.1**.

The training is applied to the Neural Network as described in CPN-SCE algorithm in Chapter 4.2. The testing is now carried out by using the 9 values as given in **Table 5.2**.

Neural Networks determine the cluster to which the values belong to and the corresponding parameters give the estimated effort. These values are given in **Table 5.2**.

Cl.No	S	EAF	ME	COCOMO	CPN-EE
0	16	0.66	33	39	34.973383
0	18	2.38	321	214	180.854615
0	20	2.38	218	243	212.698845
0	24	0.85	79	108	94.275643
0	22	1.76	230	201	179.519985
0	13	2.63	82	161	120.953225
0	12	0.68	55	33	20.560369
0	15	0.35	12	20	11.999961
0	19.5	0.63	45	46	47.240168
0	24	1.52	176	193	176.001306
0	15	3.32	237	239	193.646248
1	46	1.17	240	212	327.856469
1	30	2.39	423	327	259.443218
1	50	3.14	1063	962	1063
1	40	2.26	605	529	465.579244
1	34	0.34	47	44	47
1	28	0.96	83	102	88.075523
1	30	1.14	87	130	122.658022
1	32	0.82	106	100	101.428441
1	37	1.12	201	238	192.922529
2	4	2.22	43	30	34.324587
2	3	5.86	73	60	70.414683
2	9.4	2.04	88	89	73.135978
2	2.14	1	7.3	7	7.299985
2	1.98	0.91	5.9	5.9	5.919528
2	6.2	0.39	8	8.4	8.000005
2	2.5	0.96	8	8.1	8.313376
2	5.3	0.25	6	4.7	3.698494
2	10	3.18	122	114	121.999739
2	8.2	1.9	41	55	59.4845
2	5.3	1.15	14	22	22.758279
2	4.4	0.93	20	14	14.888
2	6.3	0.34	18	7.5	6.898763
2	6.7	2.53	57	60	65.447466
2	3.9	3.63	61	52	55.733833
2	6.7	2.53	57	60	65.447466
2	3.9	3.63	61	52	55.733833

Table 5.1 : Measured Effort (ME) and Estimated Effort(CPN-EE) (Training)

Cl.No	S	EAF	ME	COCOMO	CPN-EE
0	25	1.09	130	145	131.7661
0	21	0.87	70	68	77.8152
0	23	0.38	36	33	34.1368
0	13	2.81	98	133	129.8749
1	28	0.45	50	47	40.1753
1	48	1.16	387	239	357.3719
2	9.1	1.15	38	42	39.2456
2	6.9	0.4	8	9.8	9.3063
2	3.7	2.81	40	38	40.6043

Table 5.2: Measured Effort (ME) and Estimated Effort(CPN-EE) (Testing)

5.2 Results of Genetically Trained Neural Network

We conducted a simulation experiment in Matlab with feed forward NN model described in the previous sections using, Back Propagation (BP) and genetically trained NNs (GANN). The COCOMO data set comprising 63 projects and Kemerer data set comprising 15 projects were randomly merged to form a single dataset of 78 projects. These two data sets are, in fact, disjoint, and we can always obtain a higher prediction performance by using them separately to train two different NNs. However, in this work it was decided to merge them for two reasons:

(a) to form a common basis for comparison against the latest results reported in Ref. [7];

(b) application of *n*-fold cross validation using the merged data is a more stringent test for the neural estimator.

For *n*-fold cross validation, 80% of this historical data was used for training and the remaining for testing the person-month prediction accuracy. The experiments were repeated with different draws or folds of 80–20% partition to assess the generalization ability of the NN.

Topology of NN is 39 input neurons, 1 hidden layer of 10 neurons and 1 output neuron interconnected in a feed forward architecture, with weights from pseudo nodes modeling biases at hidden and output layer neurons as shown below in **Fig. 5.1**.

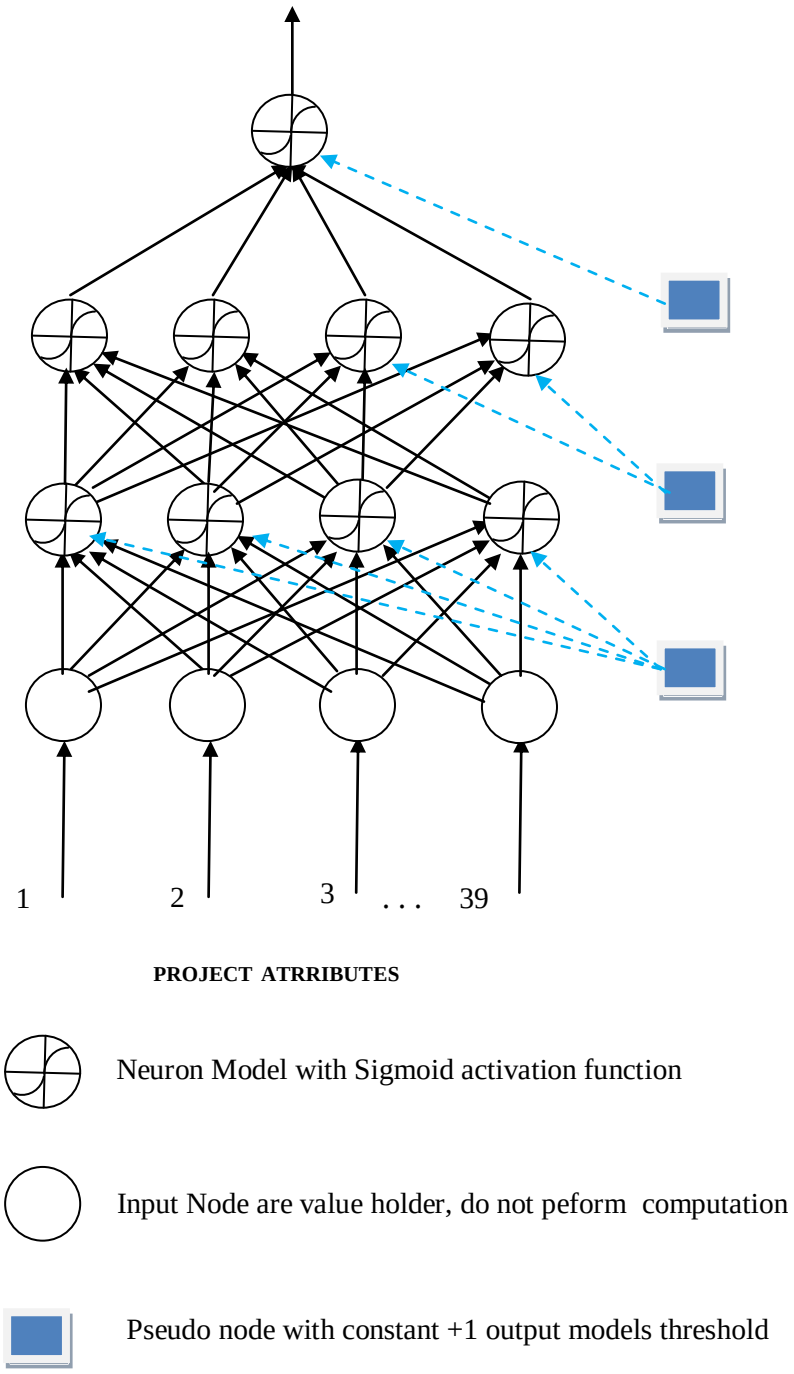

Figure 5.1: The evolutionary neural network architecture used for prediction of man-months using Boehem's attributes.

The experiments were repeated with different folds of data. A fold consists of a random draw of 63 data for training, and the remaining 15 data for testing the prediction accuracy. The comparison results for one such fold with CPN are given below in **Table 5.3**

Actual	Predicted	
	GANN	CPN
287	235.85	234.8
82.5	77.42	70.5
1107.31	1000.21	1200
86.9	88.02	87
336.3	341.23	325.6
84	81.33	82.3
23.2	41.24	30.7
130.3	122.43	150.3
116	125.03	130
72	69.56	80.2
258.7	261.64	300.5
230.7	222.79	250.8
157	168.22	190.4
246.9	202.37	180.1
69.9	62.5	80

Table 5.3: Comparison of actual and predicted effort using CPN & GANN methods

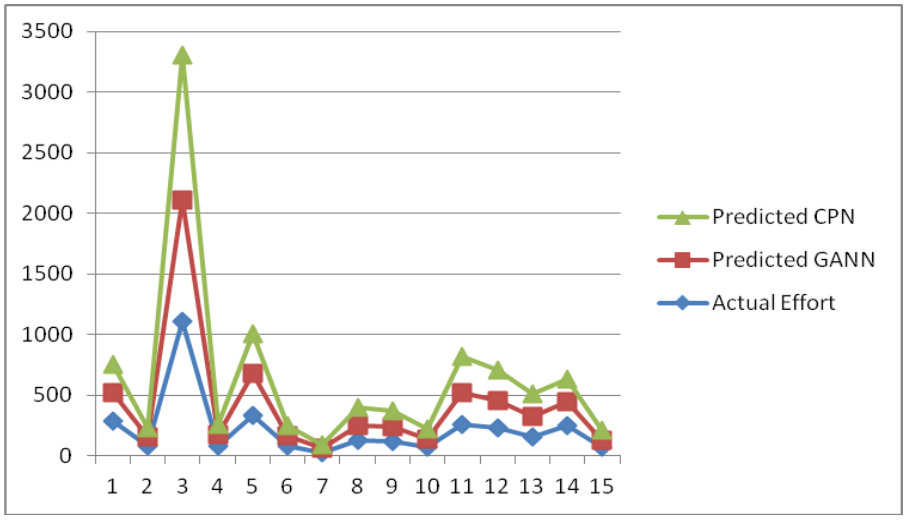

Figure 5.2: CPN Vs GANN

Here it is observed that the values obtained exhibit lesser error even though the MARE in the training phase was high. Hence the predicted GANN model produces more accurate results than CPN on given data set when given enough training. A comparison of the estimated efforts is depicted in the above graph. The **Fig. 5.2** depicts the proximity of the measured effort and the estimated efforts using the GANN model and its comparison with CPN Model.

Chapter 6. Conclusion & Future Scope

The importance of accurately predicting the software development effort (person-month) requirement of a software project cannot be overemphasized. Through a large number of simulation experiments, it has been amply demonstrated that GANNs can be excellent predictors of software development effort when trained on historical data. Neuro-genetic predictors are less sensitive to weight initializations and have excellent generalization ability as established through *n*-fold cross validation. In this study we have kept the NN topology fixed at 39-10-1 for having a common platform for comparison of different learning algorithms. In the near future we plan to evolve both the topology and weight matrix of the NN to arrive at a structurally and parametrically optimum neuro-genetic predictor. It is obvious that a predictor trained on historical data can only be as accurate as the data set itself. Hence, there is a need to continue collection of data on diverse projects with wide range of attributes to construct a sizable historical database for training neural predictors—since it not the predictor technology, but the lack of historical data that will limit the design of accurate software development effort predictors. Further research is required to assess which attributes and combinations of attributes are more influential in determining the resource requirement.

In addition we proposed a cluster based PSO technique for software cost estimation and trained the values to a neural network for classification of values encountered during testing .The CPN model proposed here can be applied effectively to large data sets and PSO tuning is more accurate if the data sets contain projects belonging to similar genres. PSO is a probabilistic model and hence cannot generate exact values. However if enough historical data is provided for training, efficient results can be obtained. Given sufficient amount of data from an organization this model can be useful to make accurate estimations. In future, Using the proposed model we are planning to develop a social web portal in which the user can enter the available data for training and then can use the software based on one of these CPN models for estimating the software costs from the input parameters. The user can train the model and following sufficient training the software could be used for future projects.

References

[1] B. Boehm, Software Engineering Economics, New Jersey: Prentice-Hall, 1981.

[2] L. H. Putnam, "A general empirical solution to the macro software sizing and estimating," *IEEE Trans. Soft. Eng*, pp. 345-361, July 1978.

[3] A. C. a. U. D. C. G. Cantone, "A comparison of models for software cost estimation and management of software projects," *Elisevier Science Publishers B.V*, 1986.

[4] W. Royce, Software project management: a unified framework, Addison Wesley, 1998.

[5] R. Schalkoff, Artificial Neural Networks, New York: McGraw-Hill, 1997.

[6] K. B. E. E. J. McCullagh, "A neural network model for rainfall estimation," *IEEE Computer Society Press*, p. 389–392, 1995.

[7] E. P.K. Simpson, Neural Network Applications, IEEE Technology, 1996.

[8] A. J. A. a. J. E. Gaffhey, Software Function, Source Lines of Code and Development Effort Prediction, IEEE transactions on Software Engineering, 1983.

[9] L. H. Putnam, "A General Empirical Solution to the Macro Software Sizing and Estimating Problem," *IEEE transactions on Software Engineering*, Vols. SE-4, pp. 345-361, 1978.

[10] R. C. Tausworthe, "Deep Space Network Estimation Model," 1981.

[11] C. Symons, Software Sizing and Estimation Mark II function Points (Function Point Analysis),, Wiley , 1991.

[12] C. Jones, How Software Estimation Tools Work. Version 5, Software Productivity Research LLC, 2005.

[13] B. W. a. P. N. P. Boehm, "Understanding and controlling software costs," *IEEE Transactions on Software Engineering*, vol. 14, no. 10, pp. 1462-1477, 1988.

[14] B. d. E. H. C. W. R. M. a. R. S. Barry W. Boehm, "Cost Models for Future Software Lifecycle Processes: COCOMO 2.0 Annals of Software Engineering," in *Ninth International*

COCOMO Estimation Meeting, L.A, 1995.

[15] R. D. H. C. e. a. Banker, "Evidence on economies of scale in software development," *Information and Software Technology,* vol. 36, no. 5, pp. 275-282, 1994.

[16] S. Cockcroft, "Estimating CASE development size from outline specifications," *Information and Software Technology,* vol. 38, no. 6, pp. 391-399, 1996.

[17] P. D. a. L. A. M. Chatzoglou, "A rule-based approach to developing software development AA prediction models.," *Automated Software Engineering 5(2): 211-243,* vol. 5, no. 2, pp. 211-243, 1998.

[18] J. J. Dolado, "A validation of the component-based method for software size estimation," *IEEE Transactions on Software Engineering,* vol. 26, no. 10, pp. 1006-1021, 2000.

[19] A. A. T. M. K. (. Ali Idri, "Fuzzy Analogy- A New Approach for Software Cost Estimation.," in *IWSM¨01*, 2001.

[20] J. Bode, "Neural networks for cost estimation," *Cost Engineering,* vol. 40, pp. 25-30, 1998.

[21] G. W. a. G. Finnie, "Estimating software development effort with connectionists," *Information & Software Technology,* vol. 39, p. 469–476, 1997.

[22] G. E. W. a. J.-M. D. G. R. Finnie, "Estimating software development effort with case-based reasoning," in *2nd Intl. Conf. on Case-Based Reasoning*, 1997.

[23] C. S. a. B. A. K. M. J. Shepperd, "Effort estimation using analogy," in *18th Intl. Conf. on Softw. Eng*, Berlin, 1996.

[24] K. Shukla, "Neuro-genetic prediction of software development effort," *Information and Software Technology*,vol 42, p.701–713, 1999.

[25] R. M. C. L. D. E. Y. Shan, "Software Project Effort Estimation Using Genetic Programming," in *School of Computer Science*, UC, University of New South Wales, 1999.

[26] M. L. Colin, J. Burgess, "Can GP improve Software Effort Estimation ? A Comparative Evaluation," *Information and Software Technology,* vol. 43, pp. 863-873, 2001.

[27] M. R. a. I. W. R. Jeffery, "Using public domain metrics to estimate software," in *7th IEEE Intl. Metrics Symp*, London, 2001.

[28] L. V. W. a. S. D. K. Maxwell, "Performance evaluation of general and company specific models in software development effort estimation," *Management Science*, vol. 45, p. 787–803, 1999.

[29] J. Koza, Genetic Programming: On the Programming of Computers by Natural Selection., MIT Press, 1992.

[30] J. Holland, Adaptation in Natural and Artificial Systems, Ann Arbor, MI: University of Michigan Press, 1975.

[31] N.-H. C. Sun-Jen Huang, "The adjusted analogy-based software effort estimation based on similarity distances," *The Journal of Systems and Software 80 (2007)* , no. 80, p. 628–640, 2007.

[32] J. Dolado, "On the problem of the software cost function," *Information and Software Technology*, vol. 43, p. 61–72, 2001.

[33] A. L. O. S. S. S. M. Ricardo de A. Araújo ∗, "An evolutionary morphological approach for software development cost estimation," *Elsevier*, vol. 32, pp. 285-291, 2012.

[34] G. A. d. S. R. T. P. R. V. Evandro N. Regolin, "Exploring Machine Learning Techniques for Software Size Estimation," in *International Conference of the Chilean Computer Science Society*, Curitiba, Brazil, 2003.

[35] F. Heemstra, "Software cost estimation," *Information and Software Technology*, vol. 34, no. 10, p. 627–639, 1992.

[36] M. Jorgenson, "A review of studies on expert estimation of software development effort," *Journal of Systems and Software*, no. 70, pp. 37-60, 2004.

[37] C. S. M. Shepperd, "Estimating software project effort using," *IEEE Transactions on Software Engineering*, vol. 23, no. 12, p. 736–743, 1997.

[38] K. K. E. Rich, Artificial Intelligence, New York,: McGraw-Hill, 1995.

[39] D. Goldberg, Genetic Algorithms in Search Optimization and Machine Learning, Reading, MA: Addison-Wesley, 1989.

[40] N.-H. C. Sun-Jen Huang *, "Optimization of analogy weights by genetic algorithm for software effort estimation," *Information and Software Technology,* no. 48, p. 1034–1045, 2006.

[41] S. V. M. P. T. Mukhopadhyay, "Examining the feasibility of a case-based reasoning model for software effort estimation," *MIS Quarterly,* vol. 16, pp. 155-171, 1992.

[42] C. S. M. Shepperd, "Estimating software project effort using analogies," *IEEE Transactions on Software Engineering,* vol. 23, no. 12, p. 736–743, 1997.

[43] J. L. Kolodner, Case-Based Reasoning, Morgan-Kaufmann, 1993.

[44] N.-H. C. b. L.-W. C. Sun-Jen Huang a, "Integration of the grey relational analysis with genetic algorithm for software effort estimation," *European Journal of Operational Research,* vol. 188, pp. 898-909, 2008.

[45] M. S. J. H. Colin Kirsopp, "Search Heuristics, Case-Based Reasoning and Software Project Effort Prediction," in *Empirical Software Engineering Research Group,* 2006.

[46] M. X. T. G. Y.F. Li *, "A study of mutual information based feature selection for case based reasoning in software cost estimation," *Expert Systems with Applications,* vol. 36, p. 5921–5931, 2009.

[47] A. L. I. O. R. L. M. Petronio L. Braga, "A GAbased Feature Selection and Parameters Optimization for Support Vector Regression Applied to Software Effort Estimation," in *AC'08*, Fortaleza, Ceara', Brazil, 2008,.

[48] A. S. a. B. SchÄolkopf, "A tutorial on support vector regressioN," *Statistics and Computing,* vol. 14, no. 3, p. 199{222, 2004.

[49] J. Kennedy and R. Eberhart, "Particle Swarm Optimization," *Proceedings of IEEE International Conference on Neural Networks,* pp. 1942-1948, 1995.

[50] T. H. a. A. S. Mohan, "Micro–particle swarm optimizer for solving high dimensional

optimization problems," *Applied Mathematics and Computation,* vol. 181, no. 2, pp. 1148-1156, 2006.

[51] H.-Y. T. Jin-Cherng Lin, "Applying Particle Swarm Optimization to Estimate Software Effort by Multiple Factors Software Project Clustering," *IEEE,* pp. 1039-1044, 2010.

[52] S.-J. H. Nan-Hsing Chiu, "The adjusted analogy-based software effort estimation," *The Journal of Systems and Software,* vol. 80, p. 628–640, 2007.

[53] CH.V.M.K.Hari, "CPN-A Hybrid Model for Software Cost Estimation," *IEEE,* pp. 902-906, 2011.

[54] Z. Dan, "Improving the Accuracy in Software Effort Estimation Using Artificial Neural Network Model Based on Particle Swarm Optimization," *IEEE,* pp. 180-184, 2013.

[55] M. S. E. Y. N. Chalapati, "Maximally fault tolerant neural networks," *IEEE Transactions on Neural Networks,* vol. 3, pp. 14-23, 1992.

[56] L. Davis, Handbook of Genetic Algorithms, New York: Van Nostrand Reinhold, 1991.

[57] A. F. Sheta, "Estimation of the COCOMO Model Parameters Using Genetic Algorithms for NASA Software Projects," *Journal of Computer Science,* vol. 2, pp. 118-123, 2006.

[58] D. R. a. A. A. A. Sheta, "Development of Software Effort and Schedule Estimation Models Using Soft Computing Techniques," in *IEEE Congress on Evolutionary Computation,* 2008.

[59] E. S. B. K. a. I. M. T. Foss, "A simulation study of the model evaluation criterion MMRE," *IEEE Transactions on Software Engineering,* vol. 29, no. 11, pp. 985-995, 2003.

[60] H. D. a. V. S. S. Conte, Software Engineering Metrics and Models., Benjamin-Cummings, 1986.

[61] L. P. a. S. M. B. Kitchenham, "What accuracy statistics really measure," *IEE Proceedings – Software,* vol. 148, pp. 81-85, 2001.