The Optimization of Success Probability for Software Projects using Genetic Algorithms

Francisco Reyes, Narciso Cerpa, Alfredo Candia, Matthew Bardeen
[freyes,ncerpa,acandia,mbardeen]@utalca.cl

Abstract

The software development process is usually affected by many risk factors that may cause the loss of control and failure, thus which need to be identified and mitigated by project managers. Software development companies are currently improving their process by adopting internationally accepted practices, with the aim of avoiding risks and demonstrating the quality of their work.

This paper aims to develop a method to identify which risk factors are more influential in determining project outcome. This method must also propose a cost effective investment of project resources to improve the probability of project success.

To achieve these aims, we use the probability of success relative to cost to calculate the efficiency of the probable project outcome. The definition of efficiency used in this paper was proposed by researchers in the field of education. We then use this efficiency as the fitness function in an optimization technique based on genetic algorithms. This method maximizes the success probability output of a prediction model relative to cost.

The optimization method was tested with several software risk prediction models that have been developed based on the literature and using data from a survey which collected information from in-house and outsourced software development projects in the Chilean software industry. These models predict the probability of success of a project based on the activities undertaken by the project manager and development team. The results show that the proposed method is very useful to identify those activities needing greater allocation of resources, and which of these will have a higher impact on the projects success probability.

Therefore using the measure of efficiency has allowed a modular approach to identifying those activities in software development on which to focus the project’s limited resources to improve its probability of success. The genetic algorithm and the measure of efficiency presented in this paper permits model independence, in both prediction of success and cost evaluation.

Keywords: Software project outcome, prediction model, optimization, software project success, genetic algorithm, efficiency

1. Introduction

The software development process is complex and expensive, with a high rate of failure. A previous research study found that 20% of software projects fail and 40% present problems in cost, planning or functionality [22]. Other studies suggest that the failure rate could reach up to 85% [14, 18, 20, 27]. Recently Charette estimated that the cost of the software project failures in the U.S. economy is between $25 and $75 billion [7], even without considering projects that exceed their budgets or finish late, which happens in the majority of cases.

To face the risks involved in software development and gain a greater advantage, many software development companies have undertaken an improvement campaign which includes adopting accepted software development processes and practices (For example, CMMI-Dev [11]). This allows them to demonstrate the quality of their development work [15]. It is important for both the outsourcing organizations and the purchasers of the software development services to receive a satisfactory product at the right price while meeting the expectations of all project stakeholders.

Overall, our research is aimed toward developing software project success/risk analysis models which can aid project managers in identifying, analyzing and controlling potential risks during software development.

The software development process is affected by many risk factors which can not be ignored, such as project management incompetence, inappropriate planning which affects project cost and scheduling, and developers’ lack of motivation [2, 4, 49].

We can define risk as the possibility of an adverse or unfortunate event that can be pronounced as producing a loss [16, 38]. Risk Management in software engineering is a set of techniques whose central objective is to avoid,
as much as possible, the negative effect that the risks can have on the project. To achieve that goal, those risk factors that may cause the loss of control must be identified and mitigated [29, 30].

Some risk factors that may induce software projects to failure are as follows [7, 22, 32]:

- **Failure in activity estimation and scheduling**: restricted resources or inaccurate estimates of needed resources (personnel and time), tight schedules, large and complex projects for resources and team experience.
- **Failure in Requirements Capture**: poor requirement specification; poor scope definition; inappropriate goals.
- **Failure in communication**: bad communication between developer and client/user; poor organizational structure, lack of leadership; lack of support from top management; lack of effort and personality clashing.
- **Failure in Process**: not meeting specifications, use of immature technologies, ineffective use of software development methods, inappropriate business processes; inappropriate assignment of resources; inadequate project management and ineffective control procedures and tools.

Additionally, previous studies have identified seven risk categories in software projects [34, 48, 49], which are as follows: (1) management, (2) clients and users, (3) requirements, (4) estimation and activity scheduling, (5) project manager, (6) software development process and (7) software development personnel.

One way to manage the risk level of a project is: (1) identifying risk factors; (2) the way these affect the project and; (3) to mitigate them. Previous research shows that it is possible to build models that can predict the success probability of a software project [1, 5, 42]. These models receive the states of a set of risk factors as input and return the success probability of the project. From a point of view, we consider that the failure probability of a project is closely correlated with its risk level and that its complement is the success probability of the project. For a positive perspective we mostly refer to the success probability.

However, risk management requires more information than the success probability to make appropriate decisions. Therefore it is helpful to know which factors have more effect on the risk level of the project. This information will allow project managers to address those factors with the aim of reducing risk and improving the success probability.

The problem with many prediction models is that it is not always clear which inputs have the most influence on project success. Cross-correlation and Principal Components Analysis have been used in previous research to identify the important factors within a similar data set for use in a logistic regression prediction model [5], but these tools do not address the relative importance of one variable over another in predicting project success. Other techniques such as automatic relevance determination [56] serve the same role as principal components analysis – mapping the original input space onto a reduced space that should make the given task easier to learn, but suffer the same problems. Furthermore, different predictive models may select different combinations of input variables for use in their predictions, further complicating the task.

An additional concern is that in practice, the cost of addressing risk factors means an investment of a project’s limited resources. We consider two main approaches that could be used to address the issue of cost, one where the cost is considered by the prediction model in determining project outcome, and one where project outcome is predicted irrespective of cost.

Most projects are considered failures due to either time or budget overruns [2]. However these are the effects of unknown issues in the development process [6]. If we consider the cost as a variable to determine project outcome, it is almost certain that it will influence the prediction of the model. This makes reducing the cost of the project the most important factor in increasing the likelihood of project success.

On the contrary if we consider cost separately from the prediction of project outcome, we will be able the identify the real causes of budget and time overruns, and thus, software project failure. Also this approach permits the incorporation of different cost estimation models and, as seen later, allows for the identification of areas of a project which represent cost effective improvements to the probability of project success.

The problem with many prediction models is that it is not always clear which inputs have the most influence on project success. Cross-correlation and Principal Components Analysis have been used in previous research to identify the important factors within a similar data set for use in a logistic regression prediction model [5], but these tools do not address the relative importance of one variable over another in predicting project success. Other techniques such as automatic relevance determination [56] serve the same role as principal components analysis – mapping the original input space onto a reduced space that should make the given task easier to learn, but suffer the same problems. Furthermore, different predictive models may select different combinations of input variables for use in their predictions, further complicating the task.

An additional concern is that in practice, the cost of addressing risk factors means an investment of a project’s limited resources. We consider two main approaches that could be used to address the issue of cost, one where the cost is considered by the prediction model in determining project outcome, and one where project outcome is predicted irrespective of cost.

Most projects are considered failures due to either time or budget overruns [2]. However these are the effects of unknown issues in the development process [6]. If we consider the cost as a variable to determine project outcome, it is almost certain that it will influence the prediction of the model. This makes reducing the cost of the project the most important factor in increasing the likelihood of project success.

On the contrary if we consider cost separately from the prediction of project outcome, we will be able the identify the real causes of budget and time overruns, and thus, software project failure. Also this approach permits the incorporation of different cost estimation models and, as seen later, allows for the identification of areas of a project which represent cost effective improvements to the probability of project success.

The problem with many prediction models is that it is not always clear which inputs have the most influence on project success. Cross-correlation and Principal Components Analysis have been used in previous research to identify the important factors within a similar data set for use in a logistic regression prediction model [5], but these tools do not address the relative importance of one variable over another in predicting project success. Other techniques such as automatic relevance determination [56] serve the same role as principal components analysis – mapping the original input space onto a reduced space that should make the given task easier to learn, but suffer the same problems. Furthermore, different predictive models may select different combinations of input variables for use in their predictions, further complicating the task.

An additional concern is that in practice, the cost of addressing risk factors means an investment of a project’s limited resources. We consider two main approaches that could be used to address the issue of cost, one where the cost is considered by the prediction model in determining project outcome, and one where project outcome is predicted irrespective of cost.

Most projects are considered failures due to either time or budget overruns [2]. However these are the effects of unknown issues in the development process [6]. If we consider the cost as a variable to determine project outcome, it is almost certain that it will influence the prediction of the model. This makes reducing the cost of the project the most important factor in increasing the likelihood of project success.

On the contrary if we consider cost separately from the prediction of project outcome, we will be able the identify the real causes of budget and time overruns, and thus, software project failure. Also this approach permits the incorporation of different cost estimation models and, as seen later, allows for the identification of areas of a project which represent cost effective improvements to the probability of project success.

We describe the following process which identifies the most influential factors to project outcome and recommends cost effective allocation of resources to improve project outcome.

1. Identify a base set of projects to use in assessing new ones, i.e. collect training data.
2. Create a prediction model that can determine the probability of success of a project. This model should determine which factors (and their associated values) contribute to the project success or failure.
3. To generate the recommendations for allocation of project resources:
   (a) Create a cost model describing how the project cost changes in relation to varying the value of each factor of the model defined in (2) above.
   (b) Create an initial description of the project in terms of the values corresponding to each factor of the model defined in (2) above.
   (c) Use an optimization method to change the value of each factor to optimize the efficiency of the solution.
4. The factors that have been changed from the initial project description represent the recommendations for better allocating project resources.
OB1: Identifying which risk factors are more influential in determining project outcome regardless of prediction model.

OB2: Identifying a cost effective investment of project resources to improve the probability of project success.

To meet these objectives, we will use the probability of success relative to cost to calculate the efficiency of the probable project outcome. We then use this efficiency as the fitness function in an optimization technique based on genetic algorithms. This method maximizes the success probability output of the prediction model relative to cost.

For testing the technique, we use previously developed prediction models to address points 1 and 2. These models were developed using data from a survey which collected information from in-house and outsourced software development projects in Chile [37, 42].

To give a brief overview of the organization of the paper: in the next section we discuss previous work in software project success prediction and cost estimation, in section 3 we describe the prediction models and the data that was used to build them. In section 4 we describe the optimization problem and the genetic algorithm approach used. In section 5 we show and analyze the results of the genetic algorithm, and finally, in section 6 we summarize the paper and provide suggestions for future work in this area.

2. Related Work

Previous research on software project outcome prediction has focused on the identification of software characteristics which indicate success [1] or the probability of occurrence of threats [39]. It seems essential to identify the characteristics of successful projects in order to determine similarities [57]. In order to have reliable predictions of project success we need to have an appropriate method to measure characteristics and identify similar projects.

To increase the probability of success in future projects, we must learn from past projects. A basic definition of a successful software project is that the associated costs and development time are within the estimates, the functionality is sufficient and the software quality is satisfactory for the client. However, this definition is post-hoc; by the time these problems are visible the software project has already failed. Instead researchers have focused on identifying those aspects that serve as predictors or causal agents for project failure or success [5].

Abe et al. collected twenty-nine metrics classified into categories such as development process, project management, company organization, human factors, and external factors to build a software project prediction model using a Bayesian classifier [1]. In a different scope, Wang used a k-means clustering algorithm to predict the success of open source software projects. The clustering was based on those groups of characteristics or success measures that describe open source software outcome, such as project outdegree (interaction between actors), active developer count change trend, bugs fixing speed and project releases [51].

Cheng and Wu proposed an evolutionary support vector machine inference model to dynamically predict project success, which was built based on a hybrid approach that includes a support vector machine and a fast messy genetic algorithm. This model integrates the process of continuous assessment of project performance to dynamically select continuous factors that influence and have the ability for predicting project success [10].

Some project characteristics or metrics are more important than others in determining project outcome and to identify similarities among projects. Some approaches based on similarity identification have been used in the past (e.g. nearest neighbor, friends, key success driver analysis). The nearest neighbor uses a distance measure, a simple approach used in case-based reasoning to identify similar projects. The friends approach uses all available information to identify a set of similar projects. The third method consists of analyzing and identifying the key success drivers to find similar projects [57]. A comparison of these methods shows that using key drivers of project success gives better software project outcome predictions. Therefore, it is important for software developers to identify the key project characteristics to improve their control over project outcome. Another approach to predict software project success which was mentioned above is the identification of project threats and their probability of occurrences [39]. Two probabilities are usually used, the probability of a threat occurring; and the probability of the threat causing a negative impact or consequence on some project success criteria.

This previous work does not consider the costs associated with the identified factors that influence project outcome. However, there are many other models that are used for estimating the effort and cost of a project [3, 21]. But, as McConnell says in relation to the estimation of cost: “The primary purpose of software estimation is not to predict a project’s outcome” [24].

Obviously, if both cost and outcome are not considered together, optimizing the efficiency of a project is impossible. If we predict that the probable outcome of a project is not good, it would be ideal if we could steer the project to a greater probability of success, but within the budget restrictions. Theoretically, it would be possible to build prediction models that take into account the associated cost of a project outcome. However this would restrict the flexibility of the prediction model. For example, when trying to use a prediction model in a company with a different cost overhead, or using different cost estimation models, the prediction model would need to be rebuilt and revalidated. Having a modular approach which separates the cost estimation model from the project outcome prediction model allows the optimization technique presented to
function with a variety of different cost and outcome prediction models.

In summary, cost estimation and prediction of software success are well researched, but a method has not yet been developed which helps project managers determine the largest risk factors and whether it is cost effective to address them. In the next section we will describe how the prediction models used in this paper were built.

3. Prediction Models

In this research we used existing prediction models [37, 42] which were trained on data collected from survey given to Chilean software developers. This survey was based on the seven risk categories from the software engineering literature mentioned above, and extensive discussions with over 90 software practitioners who develop software [34]. In this section we will briefly describe the data and the models built using the data.

3.1. Survey

The survey contained 88 questions organized into the seven risk categories. It also asked respondents if they considered the project they referenced when answering the survey was a success or a failure. The definition of software project success or failure may be different depending on stakeholders’ perceptions [31, 52, 53], but current studies have shown that developers’ perceptions have important commonalities across cultures [30, 33, 35]. The surveys did not include a definition project success in the questionnaire as the researchers felt that respondents had enough knowledge to decide whether the project they referenced in the questionnaire was successful or not [5].

The same research survey was applied in Australia and the United States. Subsets of the collected data was used in previous articles to address issues such as cost, effort, and schedule estimation [46, 48], US project management practices [43, 47], predicting good requirements [44, 45], and exploring the application of case based reasoning techniques to projects considered successful by developers and management [54, 55]. In [5], logistic regression prediction models were built and then evaluated using ROC curve analysis.

Only the projects classified as failures from Chilean, Australian, and US software developers were used in [6] and [50] to identify factors leading to project failures. In [29] part of the data collected from Chilean software developers was described. The Chilean data has only been used to build prediction models in [37, 42].

3.2. Project demographics

The questionnaire was distributed to practitioners from different Chilean organizations and each answered once. Questionnaires describing 140 projects were received. One-hundred and six projects were considered a success by the developers, and thirty-four were considered a failure. Seventy-five of the projects were in-house developments and sixty-five were outsourced projects. One hundred and twenty-seven were development projects and thirteen were maintenance projects. Eighty percent of the projects had a team size of fewer than ten developers. Two percent of projects had more than thirty developers. The largest project had a team of fifty developers. The demographics for the number of IT Contractors/Full-Time employees working on a project can be found in Table 1. For example, based on this table we can see that one project did not have company employees and was handled by IT contractors alone. Forty-five of the projects do not utilize IT contractors.

3.3. Bayesian Models

Prediction models based on Bayesian classifiers were developed for two separate target variables [37, 42] using the Necessary Path Condition learning algorithm [41]. The first target variable was the client organization’s perception of the success of a project (outcome_client) and the second was the development leader’s perception (outcome_leader). Three different techniques were used to construct the models of both these perceptions of project outcome. The first model was built using a Bayesian belief network (BBN) [8, 9], the second one was constructed based on Naive Bayesian Classifier (NBC) [8], and the third model was based on a Selective Bayesian Classifier (SBC) [36]. Therefore, a total of six models will be analyzed and compared.

The BBN approach allows the generation of a model without a target variable, so any of the variables can be used as output. This method is not purely a classifier because it can predict the state of any unknown variable based on the states of known variables (those with evidence). The BBN model has a directed acyclic graph (DAG) type of structure.

On the other hand, the NBC has a hierarchical structure and is composed by a set of input variables that can take a finite set of values called states. The output of the classifier is the most probably state for the output variable. The belief is calculated according to the states of input variables based on the knowledge of previous classified cases [9]. The SBC has the same structure as the

Table 1: Number of projects by employee type and number of employees

<table>
<thead>
<tr>
<th>Full-time Employees</th>
<th>Projects</th>
<th>IT Contractors</th>
<th>Projects</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>1</td>
<td>None</td>
<td>45</td>
</tr>
<tr>
<td>1-4</td>
<td>97</td>
<td>1-4</td>
<td>85</td>
</tr>
<tr>
<td>5-9</td>
<td>28</td>
<td>5-9</td>
<td>6</td>
</tr>
<tr>
<td>10-19</td>
<td>7</td>
<td>10-19</td>
<td>2</td>
</tr>
<tr>
<td>20-29</td>
<td>1</td>
<td>20-29</td>
<td>0</td>
</tr>
<tr>
<td>&gt;30</td>
<td>6</td>
<td>&gt;30</td>
<td>2</td>
</tr>
</tbody>
</table>
NBC but it is created using only those attributes that decision trees would use when learning [36]. The difference in structure and input variable selection method implies that these models were constructed with a different set of input variables. This may have an effect on the output of each model e.g. they could have different predictions.

The output variables from these models (outcome_client, outcome_leader) can take only two states called success and failure. The prediction model assigns the probability of occurrence of each state. In this paper will only refer to probability of success or \( P(\text{success}) \) as the output of the models. These states are disjoint, that is, \( P(\text{success}) = 1 - P(\text{failure}) \). The classification of success or failure will depend on the threshold used by practitioners and academicians to define success in a particular project [6].

These prediction models achieve up to an 85.7% prediction accuracy on holdout data. In the next section, we will describe how they will be used in our method as a means of determining the probability of success of a given project state.

4. Methodology

As stated previously, the main objectives of this paper are to develop a method which will identify which risk factors are more influential in determining project outcome regardless of prediction model, and to identify a cost effective investment of project resources to improve the probability of project success.

To this end, we define and implement a genetic algorithm (GA) to find a state of the project which represents an efficient improvement in the success probability. The GA will take the current state of the project, defined as the current states of the risk factors used in the predictive model, then algorithmically change the states to see if these changes improve the probability of success given by the predictive model. It will weigh the change in the probability of success (if there is one) with the cost of changing the states of the risk factors to determine if the proposed project state is efficient. This method requires three basic components:

1. A predictive model of project success which takes as an input the states of the risk variables of the project and returns its probability of success (Section 4.1.1).
2. A cost model which gives the cost associated with changing the states of any given risk variable (Section 4.1.2).
3. A function which measures the efficiency of the probability of success relative to the cost (Section 4.1.3).

To fill these requirements, we use the Bayesian models developed in [37, 42] and described in the previous section as our predictive models, define a very basic cost model, and adapt a measure of efficiency from the field of education [28].

4.1. Optimization problem

We would like to optimize the probability of success for a given project while keeping the costs associated with the project under control. The Bayesian models represent the learned knowledge of the initial data set (the projects). Other prediction models could be developed to similarly represent knowledge of the initial data set [5]. We wish to use the learned knowledge inherent in these predictive models to find the combination of states for all the variables which represent the best probability of success for the least cost. We now formally define the optimization problem.

4.1.1. Probability of Success

For the purpose of the optimization problem we will define the probability of success as the function:

\[
P(X') = p \tag{1}
\]

which receives an instance of the set of variables, \( X' \) and returns a value \( p \) between 0 and 1, which represents the success probability of the project.

In this paper, \( P(X') \) is taken as each one of the Bayesian models discussed in section 3.3. Each model has its own particular set of input variables, denoted by:

\[
X = \{x_1, x_2, x_3, ..., x_n\} \tag{2}
\]

where \( n \) denotes the number of input variables of the model. Which particular variables are used depends highly on the learning algorithm used when developing the model. It is likely that no two models use the same set of input variables.

Each variable \( x_i \) for \( i \) between 1 and \( n \), can take a different number of states. We will say that \( x_j \) has a value \( j \) when it is known that \( x_j \) is in the \( j \)th state. We denote \( |x_i| \) as the number of states that \( x_i \) can take. Initially all input variables have a value 0, representing a lack of knowledge of the state of the variable. A value \( j \) of \( x_i \) different to 0 is called a hard evidence of the occurrence of the \( j \)th state of \( x_i \). An instance of the set of variables \( X \) is a set of specific states for each variable, and it can be evaluated by the prediction model. We will denote an instance \( X' \) as follows:

\[
X' = \{x'_1, x'_2, x'_3, ..., x'_n\} \tag{3}
\]

For example, a model with eight risk variables (\( n = 8 \)) can have an instance representing the state of a specific project as follows:

\[
X' = \{1, 0, 3, 1, 2, 0, 1, 3\} \tag{4}
\]

In this case, we can infer \( x_1 \) is in the first state, the state of \( x_2 \) is unknown, \( x_3 \) is in its third state, \( x_4 \) is in its first state, \( x_5 \) is in the second state, \( x_6 \) is in an unknown state, \( x_7 \) is in the first state, and finally \( x_8 \) is in its third state.
4.1.2. Associated Cost

The second item we wish to consider during the optimization process is the overall cost for the project. We associate a cost to the state of each variable. For the purposes of this study, these costs are assigned manually by the user. We attempt to generate realistic values for the cost, but we are more interested in the process than an actual case study. Therefore these values remain unsubstantiated.

Thus, we consider the set of costs established for each variable \( i \) and each state \( j \), denoted \( c_{ij} \). These values are interpreted as a cost associated with the action of changing a variable from one state to another. Therefore, the total cost of an instance \( X' \) depends on the state or value of each variable and it can be calculated as follows:

\[
C(X') = \sum_{i=1}^{n} c_{ij}
\]

(5)

Initially the user defines an instance to represent a software project, this instance is denoted \( X'_0 \). Additionally the user defines the costs associated to each of the different states of the all variables.

4.1.3. Efficiency

The efficiency of a project state can be defined as the relationship of cost to its success probability, and the action of optimizing this relationship (i.e., minimizing the cost while maximizing the success probability) is equivalent to a multi-objective problem and we are looking for a solution that lies along the Pareto optimal front [13].

The method used was proposed by Paas and van Merriënboer [28] in the educational field to determine the efficiency of an instructional method defined by the relation between mental effort and performance. We generalize this efficiency as the relation between a positive and a negative attribute, in this case the success probability and cost. As far as we know, this is first use of this approach as a fitness function to a genetic algorithm.

There exist many other methods for solving multi-objective problems [13], but we choose the method described by Paas and van Merriënboer because it has the advantage of being simple to calculate and easy to understand graphically. However, as further discussed below, it is subject to the restriction of being able to specify the maximum values for each of the two attributes in order to normalize the data and prevent domination by one of the attributes over the other. It is certainly possible that other methods of determining fitness could be used.

We represent each pair (success, cost) over two Cartesian axes as shown in Figure 1. In this figure, the line \( E = 0 \) indicates an efficiency of zero and all the points on an imaginary line parallel to this have the same efficiency. Increases in efficiency are represented by those lines shifted to the upper left of the diagram (higher success probability in relation to cost). Conversely lines with lower efficiency are those shifted to the lower right (lower success probability in relation to cost).

The success probability and cost associated to an instance must be standardized with a mean equal to 0 and a standard deviation equal to 1. To calculate this value we make use of the maximal cost of a project based on the cost specification defined by the user. To generate the maximal cost of a project, we add the maximum costs associated to each software development activity (i.e., for each risk variable of the project), which is necessary for its success. These activities were obtained from the literature and confirmed by the survey applied to software development companies and in-house development in Chile.

A caveat must be stated here: if the maximum cost of a project is high, a change in cost could be translated into small change in efficiency after the standardization. This means that the absolute change in cost could be relatively small in comparison to the maximal cost of the project. Care must be taken in interpreting the results of this fitness function, large changes in absolute cost could be interpreted by the fitness function as being efficient in terms of the maximal cost of the project. For this reason a constraint in cost may be needed to ensure that the genetic algorithm finds a solution within the project budget.

With the success and cost values standardized, the efficiency associated with an instance is calculated as the perpendicular distance to a line \( C = P = 0 \) with efficiency \( E = 0 \). We calculate the efficiency as follows:

\[
E = \frac{P - C}{\sqrt{2}}
\]

(6)

4.2. Formal definition of the optimization problem

We are interested in the instance that maximizes the efficiency of a given project, that is the success probability restricted by a maximum cost. Therefore, the optimization problem can formally be described as:

\[
\max z = \frac{P(X') - C(X')}{\sqrt{2}}
\]

(7)

restricted by \( C(X') \leq C_{\text{max}} \), where \( C_{\text{max}} \) is a cost restriction that the user must determine based on the available project resources.

Therefore, the solution to the problem consists in a value or state for each risk factor (input variables) that maximizes the success probability (output variable) and the cost associated to this instance is less or equal than the maximum cost \( C_{\text{max}} \).

4.3. Problem Size

To measure the size of the problem, we can count the number of states combinations or distinct instances, which depends of the number of variables in the model and the number of states of each variable. It can be calculated by the following expression:

\[
\sum_{k=1}^{n} \prod_{i=1}^{k} |x_i|.
\]

(8)
Figure 1: Representation of the efficiency using standardized success probability and cost. In this case, \( E=0.58 \) represents a project with a change in cost of -0.2 units and a change in the probability of success equal to 0.62. This represents a large increase in efficiency in comparison with the base project (\( E=0 \)).
It is clear that the size of the problem search space increases exponentially as a function of $k$ and $n$. For large search spaces an exhaustive search is impractical, so to find a good solution we utilize a meta-heuristic technique based on genetic algorithms. This technique has been used extensively for optimization problems such as this ([19], for example) and often finds a good solution within a reasonable running time [25, 26]. The description of this technique is presented in the next section.

### 4.4. Genetic Algorithm (GA)

The general idea behind genetic algorithms is as follows [26]: a population of abstract representations (denominated chromosomes) of possible solutions (individuals) to an optimization problem evolves to produce better solutions. The evolution process begins with a population of randomly produced individuals. In each generation all individuals are evaluated by a fitness function, randomly selected and modified (by mutation and crossover) to form a new population that replaces the previous. This process is repeated until satisfying a termination criterion. The sequence of steps of the genetic algorithm can be observed as a flowchart in Figure 2, which represents the generic approach [25, 26] with specific values.

Below, we describe the basic procedures, functions and parameters values used in our implementation of the genetic algorithm.

#### 4.4.1. Encoding

Each individual of the population represents a possible solution to the problem. In this case an individual is considered as an instance for the model. The individual must be represented by a chromosome consisting of a string. In this implementation, each element of the string, called a gene, represents a risk variable $x_i$ of the project. The value assumed by a gene, called the allele, represents the state of the corresponding risk variable (Figure 3). Thus a chromosome is a direct representation of Equation 3.

#### 4.4.2. Initialization

The initial population is formed of possible solutions to the problem. These solutions can be generated randomly or by using heuristic methods [17]. In this research the initial population is formed of one chromosome with the states given by $X_0$ and the rest of the population are random modifications of the value of one gene respect to $X_0$. The population size $M$ is set to be 80 in this research [26].

#### 4.4.3. Fitness Function

The fitness function evaluates the chromosomes taking as input an individual and giving a number proportional to its optimality as output. In this case, we use the efficiency as given in section 4.1.3 as our fitness function.

#### 4.4.4. Constraints

To comply with the constraints defined in the optimization problem, we ensure that a given chromosome satisfies the constraints. To accomplish this, we penalize the fitness value when the chromosome is not feasible [12]. In this case, the fitness for these chromosomes is halved to decrease the chance that the algorithm will converge on a local maximum while also avoiding their selection in the evolution process [25].

#### 4.4.5. Selection and reproduction

The reproduction of the population to the next generation must privilege the high fitness chromosomes, but is also important to preserve the population diversity. There are times when individuals with lower fitness help to explore new solutions and avoid the population converging on a local optimum.

To obtain a new generation, parents are selected using the Roulette wheel method, a probabilistic method in which the probability of an individual of being selected for inclusion in the next generation is proportional to its fitness value [26].

The crossover process generates new individuals from the selected members of the population. The occurrence of crossover is characterized by a crossover rate, which indicates the chance of two individuals exchanging gene information. This rate can be selected using a sensitivity analysis. In this research the rate of crossover was 0.8 [26].

The crossover method used was a one-point crossover, where a random position in the chromosome is selected as the crossover point. Two offspring are then created from the two selected parents. The parents are both split at the same point and one offspring is created from the head of the first parent and tail of the second. The second offspring is formed inversely – from tail of the first parent and head of the second.

#### 4.4.6. Mutation

The process of selection acts to reduce the diversity of the population, so a mutation rate is used to maintain a degree of heterogeneity in the solutions helping to avoid premature population convergence. If selected for mutation, the value of a randomly selected gene on a chromosome is modified slightly. In this work, the probability of mutation for a chromosome was 0.1%. When such a mutation occurs, the new state for the gene is selected randomly from the available states.

#### 4.4.7. Termination

The evolutionary process can be repeated infinitely, but the idea is to obtain a solution in a reasonable time. The first termination criteria is a limit in the number of generations. The maximum generation number can be set very large but extra computation could be performed after the solution has already been found. A convergence measure can provide a early termination of the execution.
Figure 2: Standard Genetic Algorithm Procedure (Figure adapted from [19], supported by [25, 26]).
Figure 3: Example chromosome with eight genes. The specific value of a gene is its allele. In this case, a gene represents a specific risk variable and the allele is the associated state or value of the variable. This chromosome is a representation of the instance depicted in expression 4.

(Automated Accelerated Termination). This research uses a convergence measure based on the evolution of the best and the average fitness of the population. If the difference between the best and the average fitness is less than some threshold after a given number of generations, we can assume that the population has converged. In this case we set a threshold of 0.01 over 20 generations [23].

4.5. Execution of the algorithm

The execution of the entire algorithm can be summarized as the following:

1. Build a model capable of generating an estimate of the probability of success of a given project. In this case, we use models generated by Bayesian methods from previous research [37, 42].

2. Once the model is built, use the variables (attributes) identified as the chromosome structure for the GA.

3. Build the initial population of the GA.

4. Repeat the following until the termination criteria are met.

   (a) Calculate the success probability of all individuals in the population by inputting the values associated with a specific individual into the model.

   (b) Calculate the cost of all individuals in the population by calculating the cost associated with all the states of the alleles for a specific individual.

   (c) Calculate the efficiency/fitness of all the individuals using their respective cost and probabilities of success as found previously.

   (d) Based on the fitness calculated above, select and reproduce the appropriate number of individuals for the next generation of the genetic algorithm.

5. When the termination criteria are met, choose an individual from the population with the highest fitness. This instance represents an efficient, ideally successful, project based on the constraint of cost. Those variables that have changed from the initial state of the project are those where project managers should focus their attention.

5. Results

We ran the genetic algorithm using the six prediction models created with data from past software projects in Chile. These models were described in detail in section 2.1. For both types of outcome (outcome_client and outcome_leader) we had a model created with each Bayesian method (BBN, NBC, SBC).

We defined a unique example case for each model. Each of these cases represents a software project with a low success probability and were based on projects reported in the survey used to build the prediction models. The initial value for each variable was manually set to obtain a success probability under 0.5. The cost associated with each state of each variable is set according to expertise knowledge or from the literature. These costs include the effect that each state has on the risk factor evaluated. For example in all cases to have better requirements documentation implies a higher cost than to not have it. It is necessary to point out that not all projects consider the same magnitude of cost, because some may represent major projects and others may represent smaller development. For all models the maximum cost is set as a 20% increase from the initial cost of the project. All runs were terminated at generation 300 if they had not already converged.

In each execution the GA encounters a satisfactory solution to the problem, given by the chromosome with the highest fitness value found in the population. The chromosome information indicates which variables or risk factors must be managed and controlled to improve the project success at an appropriate cost. This suggests a strategy of where given resources should be allocated to achieve the highest gains in success with the least cost.
Table 2 shows examples of the variables’ labels and states proposed after the execution of the genetic algorithm.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Proposed State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Has the Project Manager control over the project?</td>
<td>Yes</td>
</tr>
<tr>
<td>Will change control be monitored and dealt with effectively?</td>
<td>Yes</td>
</tr>
<tr>
<td>Will the customer’s expectations be managed throughout the project?</td>
<td>Yes</td>
</tr>
<tr>
<td>How well do the team members work together?</td>
<td>Very well</td>
</tr>
<tr>
<td>Is working on the project a pleasant experience?</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Table 3 describes the search space associated with each model. The search space is the total number of possible combinations considering the variables of the model and their potential states. It is calculated using expression (8).

<table>
<thead>
<tr>
<th>Model</th>
<th>Variables</th>
<th>States</th>
<th>Search Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>BBN outcome_client</td>
<td>28</td>
<td>73</td>
<td>$5.73 \times 10^{11}$</td>
</tr>
<tr>
<td>BBN outcome_leader</td>
<td>4</td>
<td>10</td>
<td>560</td>
</tr>
<tr>
<td>NBC outcome_client</td>
<td>10</td>
<td>33</td>
<td>$5.35 \times 10^{9}$</td>
</tr>
<tr>
<td>NBC outcome_leader</td>
<td>5</td>
<td>14</td>
<td>8,820</td>
</tr>
<tr>
<td>SBC outcome_client</td>
<td>9</td>
<td>19</td>
<td>$2.99 \times 10^{9}$</td>
</tr>
<tr>
<td>SBC outcome_leader</td>
<td>7</td>
<td>16</td>
<td>$7.98 \times 10^{9}$</td>
</tr>
</tbody>
</table>

Figure 4 presents the evolution of the success probability and cost of the six models. It can be appreciated that the fitness function does not optimize solely one of the axes, but instead optimizes across both at the same time. A decrease in cost could also be mirrored with a decrease in success probability, such as in Figure 4(a). Likewise, an increase in cost could be offset by an increase in success probability as in Figure 4(f). As in all the cases, these two specific cases represent an overall gain in the efficiency of the risk management approach implemented by the solutions.

5.1. BBN models

The execution with the client’s perception of success and the BBN approach (Table 4) starts with an instance which has 44.49% success probability and 37 cost units. The maximum cost allowed in this case is 44 units.

The execution is terminated at 300 generations at which point the fitness of the best individual is 0.887. The probability of success of the best solution is 96.13% with a cost of 25 units, an improvement of 51.64% and decrease of 32.43% respectively.

In this case the GA finds an instance in which the success probability is greater than original probability, and does it at a lower cost – improving the efficiency of the solution. Figure 4(a) shows an interesting result – the GA finds a solution with a higher probability of success (97.56%), but then abandons it in favor of a solution with a slightly lower probability but much lower cost (i.e. higher efficiency). The number of generations it takes for this model to converge is acceptable, given that it has the largest search space, as we can see in Table 3.

This situation might represent an instance in which the probability of success of the project could be improved by shifting the resource allocation to different aspects of the software development process. Overspending on one aspect (e.g. managing the customer’s expectations throughout the project) might lead to other aspects being neglected (e.g. a less pleasant work experience or bad team dynamics).

The evolution of the GA over the BBN approach associated to leader’s success perception can be seen in Table 4. The case defined has an initial success probability 20.64% and the initial cost is 30 units.

In this case, the population converges in 43 generations and the fitness reaches 0.707. The success probability is of 100.00% at a cost difference of 0 units.

This execution finds an instance that improves the efficiency immediately due to the small search space associated with this model. This case needed only a change of state in just one variable to acquire the highest probability of success, thus an instance with the highest efficiency was found among the initial population of mutated individuals.

This situation represents a software project in which the project manager has not taken the most efficient decisions and improving the success of the project depends on just simple corrections which do not require any additional expense.

Lifting the cost restrictions and then optimizing the models again shows no change in the final values, as seen in Table 5.

5.2. NBC models

The results of GA with NBC model and client’s success perception as output variable can be seen in Table 4. The client’s instance, as defined, has a success probability value of 1.17% with 216 units of cost.

The GA was cut off at 300 generations, acquiring a fitness value of 0.584. The success probability reached 84.85% with 241 units as cost, as can be seen in Figure 4(c).

This model has a large number of possible state combinations, for that reason, the GA execution takes a high number of generations. The initial value for success probability is very low and the improvement of the fitness value is slower than in previous models. However, acceptable
Figure 4: Evolution of Success Probability and Cost of the fittest individual for all models (With cost restrictions)
values for the success and cost are acquired after only 122 generations. The solution encountered has a success probability that is very high compared to the initial value, with an additional cost as well. Nevertheless, the cost increase is not high in terms of the maximal cost of the project and thus, the efficiency increases.

Additional gains in the probability of success can be made with more resource expenditure – an extra 100 units of cost nets a 97.55% probability of success for the project, as shown by running the GA without any cost restrictions (Table 5).

The situation described by this execution represents the most common software project. The assurance of success involves some resource compromise. Allocating resource spending to maximize the success probability can contribute to the good management by the software project leader. This type of information permits the project leader to make informed decisions on how best to manage the project.

In the case of the NBC model with the leader’s perception of success, the behavior is presented in Figure 4(d). Initially, the value for the success probability is 2.03% and the cost associated to this instance is 21 units. The execution ends at 53 generations and the final fitness value is 0.218. The success probability reaches 89.82% with 34 units of cost. The results of this execution shows a large improvement in terms of success probability, with only a slight (in comparison to the other cases) increase in cost. The GA only runs for 53 generations because of the model’s small search space.

### Table 4: Results of Genetic Algorithm Optimization (With Cost Restrictions)

<table>
<thead>
<tr>
<th>Model</th>
<th># of Generations</th>
<th>Best Fitness</th>
<th>Initial Probability of Success (%)</th>
<th>Ending Probability of Success (%)</th>
<th>Initial Cost</th>
<th>End Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>BBN outcome_client</td>
<td>300</td>
<td>0.887</td>
<td>44.49</td>
<td>96.13</td>
<td>37</td>
<td>25</td>
</tr>
<tr>
<td>BBN outcome_leader</td>
<td>43</td>
<td>0.707</td>
<td>20.64</td>
<td>100</td>
<td>30</td>
<td>30</td>
</tr>
<tr>
<td>NBC outcome_client</td>
<td>300</td>
<td>0.584</td>
<td>1.17</td>
<td>84.85</td>
<td>216</td>
<td>241</td>
</tr>
<tr>
<td>NBC outcome_leader</td>
<td>43</td>
<td>0.333</td>
<td>2.03</td>
<td>48.26</td>
<td>411</td>
<td>414</td>
</tr>
<tr>
<td>SBC outcome_client</td>
<td>300</td>
<td>0.566</td>
<td>5.79</td>
<td>81.45</td>
<td>25</td>
<td>30</td>
</tr>
<tr>
<td>SBC outcome_leader</td>
<td>53</td>
<td>0.218</td>
<td>2.67</td>
<td>89.82</td>
<td>21</td>
<td>34</td>
</tr>
</tbody>
</table>

### Table 5: Results of Genetic Algorithm Optimization (Without Cost Restrictions)

<table>
<thead>
<tr>
<th>Model</th>
<th># of Generations</th>
<th>Best Fitness</th>
<th>Initial Probability of Success (%)</th>
<th>Ending Probability of Success (%)</th>
<th>Initial Cost</th>
<th>End Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>BBN outcome_client</td>
<td>289</td>
<td>0.887</td>
<td>44.49</td>
<td>96.13</td>
<td>37</td>
<td>25</td>
</tr>
<tr>
<td>BBN outcome_leader</td>
<td>45</td>
<td>0.707</td>
<td>20.64</td>
<td>100</td>
<td>30</td>
<td>30</td>
</tr>
<tr>
<td>NBC outcome_client</td>
<td>257</td>
<td>0.647</td>
<td>1.17</td>
<td>97.55</td>
<td>216</td>
<td>341</td>
</tr>
<tr>
<td>NBC outcome_leader</td>
<td>52</td>
<td>0.613</td>
<td>2.03</td>
<td>98.82</td>
<td>411</td>
<td>1814</td>
</tr>
<tr>
<td>SBC outcome_client</td>
<td>52</td>
<td>0.674</td>
<td>5.79</td>
<td>97.36</td>
<td>25</td>
<td>59</td>
</tr>
<tr>
<td>SBC outcome_leader</td>
<td>51</td>
<td>0.218</td>
<td>2.67</td>
<td>89.82</td>
<td>21</td>
<td>34</td>
</tr>
</tbody>
</table>

5.3. SBC models

The behavior of the GA over SBC model using the client’s perception of success as the output variable is presented in Figure 4(e). Initially the value for the success probability is 5.79% and the cost associated to the case is 25 units. The execution in this case is cut off at 300 generations, and the final fitness value is 0.566. The probability of success for the best solution reaches 81.45% with 30 units as cost.

The success probability improvement is high and the cost associated is not significant, thus the solution proposed is very efficient. After removing the cost restriction it can be seen in Table 5 that with an additional outlay of resources (59 units of cost, nearly 100% increase over the constrained solution) the project is almost guaranteed success (97.36% probability of success).

Finally, the execution of the GA over SBC model with the leader’s perception of success is presented in Figure 4(f). Initially the value for the success probability is 2.67% and the cost associated to this instance is 21 units. The execution ends at 53 generations and the final fitness value is 0.218. The success probability reaches 89.82% with 34 units of cost. The results of this execution shows a large improvement in terms of success probability, with only a slight (in comparison to the other cases) increase in cost. The GA only runs for 53 generations because of the model’s small search space.

13
This case represents a software project which will most probably fail, but with a relatively small outlay of resources in the correct areas it has a much higher probability of success. Additional spending is not likely to provide an efficient increase in the probability of success, as is seen after the cost restrictions are lifted (Table 5).

In summary, we have seen that this method can identify areas where a software project could be improved. This information can aid the project manager make more informed decisions on how to allocate resources. We have seen that the method identifies instances where shifts in the project resource allocation, with or without additional cost, results in a greatly improved probability of success. We have also seen an example in which the method suggests that the current resources are not enough to improve the probability of success, and the project manager would need to secure more resources. We also have seen a case in which the method shows that neither a shift in the allocation of resources, nor the securing of additional resources would be enough to change the probability failure outcome of the project.

During the process of software development, this information would be invaluable in aiding the decision making process of the project manager. As seen in [40], there are two problems with making decisions: too much information (information overload), and too little information. The method we have proposed here diminishes the risk of making incorrect decisions due to these two problems.

6. Discussions and Conclusions

In this paper we have shown that by strategic allocation of resources, a project manager can help ensure the success of a project, often without a large increase in cost. When additional resources are needed, it is helpful to know where they should be allocated and what return on investment they should bring. This will ultimately help to diminish the risk of the project.

In accordance with our objectives listed in Section 1, we have successfully developed a method to help identify those factors in the software development process that are more influential in determining project outcome, regardless of the specific implementation details of the predictive model (OB1). Meeting this previous objective allows us to identify cost effective investments of project resources to improve the probability of project success (OB2).

Using the measure of efficiency from [28] has been useful in meeting the objectives, as it has allowed a modular approach to attacking the basic problem in software development of where to focus limited resources to improve success probability. This modularity permits model independence, in both prediction of success and evaluation of associated cost.

It should be noted that the approach taken here is to optimize the probability of success as given by the Bayesian models. A major assumption is that the models return predictions that accurately represent the real world probability of success for the project. Thus, the modularity of the method is very advantageous, as models built with different techniques which perhaps offer greater predictive accuracy could easily be substituted for the Bayesian models used in this research.

Likewise, since the cost is not included in the prediction model, but rather a component of the efficiency measure, it is trivial to adopt any software development cost model to be used in this method.

An important but time-consuming extension of this work would be to come full circle and study the effectiveness of implementing the changes suggested by this approach in an ongoing project. This extension would help to confirm the Bayesian prediction models, but it would not affect the methodology described in this paper.

By using a fitness function based on the definition of efficiency proposed by Paas and van Merriënboer in the field of education [28], we have developed an aid to the software development process that uses the power of a genetic algorithm to quickly identify those factors that influence success in the most cost effective way.

Acknowledgments

This research is fully supported by the project FONDECYT 1030785, of the Chilean Government.

References


