International Journal of Computational Intelligence Systems, Vol. 6, No. 5 (September, 2013), 796-821

Multi-step Generation of Bayesian Networks Models for Software Projects Estimations

Raquel Fuentetaja [1] Daniel Borrajo [2] Carlos Linares López [3] Jorge Ocón [4]

[1] *Universidad Carlos III de Madrid, Spain,*
Email: rfuentet@inf.uc3m.es
[2] *Universidad Carlos III de Madrid, Spain,*
Email: daniel.borrajo@uc3m.es
[3] *Universidad Carlos III de Madrid, Spain,*
Email: clinares@inf.uc3m.es
[4] *GMV, S.A., Spain*
Email: jocon@gmv.com

Received 22 February 2012
Accepted 21 March 2013

Abstract

Software projects estimations are a crucial component of successful software development. There have been many approaches that deal with this problem by using different kinds of techniques. Most of the successful techniques rely on one shot prediction of some variables, as cost, quality or risk, taking into account some metrics. However, these techniques usually are not able to deal with uncertainty on the data, the relationships among metrics or the temporal aspect of projects. During the last decade, some researchers have proposed the use of Bayesian Belief Networks (BBNs) to perform better estimations, by explicitly taking into account the previous shortcomings. But, these approaches were based on manually defining those BBNs and handling only one of the estimation variables (cost, quality or risk). In this paper, we present an approach for semi-automatically building BBNs by using machine learning techniques. We describe two algorithms to generate such BBNs. The first one generates one-shot BBNs, while the second one generates BBNs that take into account the temporal aspect of project development. We performed experiments on real data coming from two software companies, obtaining a 63% of accuracy on multi-class classification. Our main interest was to find a semantically correct model that can be trained with future projects to increase its accuracy. In this sense, we introduce a well-balanced approach to make good predictions with strong explanatory power.

Keywords: Software estimation, Bayesian Belief Networks

1. Introduction

Project estimations have always been a major challenge in software development. [18,7] Current estimation methods are mainly based on models for project characterization. External attributes of interest (cost, schedule, effort, budgeting, quality) are related with a variety of models to *internal* system metrics (structure, behaviour, data management). The most used external attributes, that lead to different types of models for estimation, have been quality, cost, and risk. These models are based on quantifiable metrics

Copyright: the authors

of the project, and their corresponding relationships and dependencies. Among others, the most important difficulties when generating estimations from the start of the project are:

- Software development processes are crowded of various forms of uncertainty. For example, the same action (such as a design decision) can produce different outcomes and, equivalently, the same sets of observations can lead to different states. Uncertainty also comes from the consideration of human factors such as the ability and expertise of both the development team and the managers, among others.
- On the other hand, most previous approaches only focus on one of the main variables (cost, quality and/or risk). [7,3,5] Some works consider the diversity of methods, [25] but it is also interesting to consider the diversity of factors and their quantitative contribution.
- Besides, many of the most used variables to estimate projects are only known at the end of the project. Thus, there is a need of estimation models that are able to predict values with the highest possible accuracy for cost, quality and/or risk from the beginning of the project, in the presence of uncertainty.

Bayesian Belief Networks (BBN) represent a new method for generating software estimations. [3,17,42] BBNs are well defined and provide advanced analysis techniques based on probability calculus. The main advantage of using BBNs for making predictions is that they allow users to obtain the estimation in presence of uncertainty and incompleteness of the input parameters. In addition, BBNs allow users to declaratively represent the causal relationships among the attributes. Also, in software development, the time component is known to be critical. [17,42] While some of the variables are known at the beginning of the software development cycle, others are only known at intermediate steps. BBNs provide a simple way to qualitatively model time, since software engineers can refine the predictions further when more variables are known, simply by introducing their values.

In this paper we propose an approach to semi-automatically generate estimation models based on BBNs for project metrics taking into account the uncertainty aspects of software development. Specifically, we will focus on the estimation of parameters such as effort, quality, or risk, to mention the most prominent ones. By effort estimation we understand those methods and techniques oriented towards the *a priori* determination of cost and schedule parameters for a given project. Quality estimations are intended to predict the number of failures that will be produced in the development. And risk estimations tend to predict the probability that a given parameter of the project (mainly costs or schedule) could be underestimated.

The main contribution of the paper is the definition of a multi-step process for semi-automatic generation of BBNs for software estimations, using an off-the-shelf data mining tool. This process considers that different variables are known at different time steps of the life-cycle of a software project. To the best of our knowledge there is no other work using this method for generating BBNs neither for software estimations nor for other processes in which variables are known at different moments. One of the direct advantages for software engineers is that as project phases conclude, more variable values are known. Thus, estimations can be refined.

A second contribution of the paper is a case study on the use of the previous process to obtain estimation models for two software companies. Those models can later be used as input models for project estimation tools. In order to test the effectiveness of the proposed approach, we have gathered data from 30 software projects of two medium-sized companies, GMV and Skysoft, and generated BBN models for estimating cost, quality and risk. We present experimental results that show the advantages and disadvantages of this approach.

The rest of the paper presents the task of software project estimation in Section 2. A brief introduction on BBNs is discussed in Section 3. The proposed methods for generating BBNs are presented in Section 4. The paper ends with some experiments and results in Section 5, a review of related work in Sec-

Copyright: the authors

tion 6. and some conclusions in Section 7.

2. Software Project Estimations

Making software project estimations is a difficult task. There have already been some reviews that defined this process. For instance, Molokken and Jorgesen reviewed different surveys on project software estimation. [32] The review covers 10 different surveys, which span through the years 1984 to 2002. The conclusions of the study for project overruns are clear: although the surveys analyzed in this paper provide different results, all of them identify deviations in projects in the majority of the cases. To summarize the main results in the literature on software engineering, it seems as if most projects (60-80%) are completed over budget and/or schedule. Most results also indicate that the percentage and magnitude of overruns increase as projects grow in size, and the magnitude of average effort and cost overrun is between 30% and 40%.

Thus, it is crucial for software engineering efforts to correctly estimate from the start of software projects several aspects, such as cost, quality or risk. Since the start of the formal analysis of software development, there have been many approaches to estimate several of these parameters taking into account data coming from different sources (project attributes, software metrics, etc.) and phases (analysis, design, coding and testing) of software engineering. [18,7] In fact, software metrics is a collective term used to describe the very wide range of activities concerned with measurement in software engineering. These activities range from producing numbers that characterize properties of software code (these are the classic software metrics such as the ones used in COCOMO) to models that help predict software resource requirements and software quality. [7,17,40]

Formally, a simplified description of the problem of prediction in Software Engineering consists of: given the results of a set of projects, each described by a set of input, intermediate, and output attributes (metrics), obtain a model that can predict the values of the output attributes as a function of the values of the input and intermediate attributes. This model will be used in future project estimation processes, by providing the values of the input (and possibly intermediate) attributes, to obtain the values of the output attributes. So, usual questions to use a model like this are:

- What are the relevant metrics - attributes?
- What is the complexity of projects and where does it come from?
- How do the values of some attributes affect others?
- What is the impact of changes of values on some attributes?
- How do we represent those attributes?

We will now discuss several approaches that address some of these questions. We have used the term attributes, though there have been other names given to the same concept. Examples are: decisions (usually represented by input or intermediate features), variables, features, factors, or parameters. Also, some of these attributes come from project measures and are usually referred to as metrics. In some papers, those metrics are used as the desired output. However, they are usually considered to infer the values of the project measurement estimates as intermediate or even input attributes. On the other hand, numerical metrics provide a classification of the software metrics that is commonly accepted. [34] It divides relevant variables into what are traditionally considered "internal" (input variables for the project estimation task) and "external" —output variables for the project estimation task.

The first known metric, and most commonly used, is the Lines of Code (LOC or KLOC for thousands of lines of code) or size metric. Though it is a weak metric to be used in isolation, it was and still is the basis for the measurement of programming productivity (LOC per programmer/month). [36,7] However, this metric provides *a posteriori* estimations, given that its actual value is only known at the end of the project. And, using an estimation of LOC for estimating software cost makes estimating a metric (cost) depend on the estimation of another one (LOC), which is not the best way to proceed. Further studies proposed regression-based models for module defect density (number of defects per KLOC) in terms of module size measured

Copyright: the authors

in KLOC. Other classical metrics and related features were effort, schedule (milestones, achieved goals, slacks, percentage of schedule overdue, etc.), quality (total number of errors opened/closed, number of errors opened/closed since last report, type of errors, etc.), documentation, or rework (number of open/closed Software Change Orders). The need for more discriminating measures was evident during the 70s with the increasing diversity of different programming languages. At this time, measurements for software complexity and measurements of functional size were developed. Thus, usual metrics refer to functional complexity, such as number of functions, McCabe cyclomatic complexity, [30] function points, [2] or information flow complexity. [22]

Other metrics have been defined for the Object-Oriented (OO) paradigm. [12] The authors propose metrics such as: weighted methods per class, depth of inheritance tree, number of children, coupling between object classes (classes interdependence), response for a class, and lack of cohesion in methods. Simpler metrics in OO are number of classes, number of methods, number of properties, or similarity between classes —also known as cohesion.

Most of the early work on project estimation went into cost estimation, mainly using complexity as the main driver. Boehm *et al.* survey the main approaches for project estimations and define six kinds of techniques [6]:

- *Model-based*: they are parametric techniques, as SLIM, [36] or COCOMO. [7] They rely on models represented in a variety of formalisms (as functions, distributions, or knowledge bases) that depend on some parameters and are able to produce project estimations.
- *Expertise-based*: they are based on experts judgements. Examples are the Delphi approach or the hierarchical decomposition of Work Breakdown Structure (WBS). [38,7] They have the advantage of incorporating the knowledge of experts, and the disadvantages that they are biased by the experts that defined them (thus, sometimes, they are domain dependent), and also the estimation models are usually hard to obtain.
- *Learning-Oriented Techniques*: the creation of the estimation model is posed as an inductive task, and machine-learning techniques are used to automatically generate the models from data. Examples of these techniques are analogy (Case-Based Reasoning), [21] or neural networks. [19] The advantage of these techniques is that they alleviate the knowledge acquisition task, and the main disadvantage is that many instances of correct (little noise, no missing data, etc.) pairs ⟨project, output features values⟩ are needed while usually very few examples are available.
- *Dynamics-Based Techniques*: they assume that software project estimations change over the software development cycle. Thus, estimations can be defined in terms of formal models such as differential equations. They are good for planning and control, but particularly difficult to define and calibrate.
- *Regression-Based Techniques*: they have been the most widely used ones and pose the task as the learning-oriented ones (in fact, one could merge them together): starting from data of ⟨project, output attributes values⟩ they generate a regression model (usually as a linear function of the known variables). They obtain good results when there are lots of data or not much data is missing, there are no outliers, and variables are uncorrelated. However, these conditions are seldom met. When the understandability of the model is important, these techniques are a good option.
- *Composite Techniques*: they combine two or more of the previous techniques. For instance, the BBN approach uses a causal model defined by the experts that can be initially injected with estimations on conditional probabilities generated from previous projects.

There are many surveys that compare different project estimation methods and techniques. [19,24] Berlin *et al.* revise regression methods with neural networks [4]; and Mair *et al.* review methods that belong to only one kind, [28] machine learning techniques. Even if most papers focus on cost estimation, the same techniques can be (or have been) used for other kinds of metrics, such as quality and risk. The main metrics used to compute cost have been software complexity (measured, in turn, by differ-

Copyright: the authors

ent metrics as explained before), experience of the design and development team, or even cultural aspects. [21] In that paper, the authors defined an ontology to represent different aspects affecting cost estimation.

Another potential classification criteria could have been the kind of software estimation a given approach focuses on. Thus, there are approaches for cost estimation, quality estimation, or risk estimation. For instance, there is a book that reviews a collection of papers on software quality and presents a survey of quality models in Software Engineering, [16] including quality estimation. Many of the approaches for estimating quality are based on a hierarchy of features. Cho *et al.* redefine some OO metrics applied to quality estimation, [13] in the context of component-based software development. They define up to four different metrics for complexity, and others for customizability, and reusability. The case of risk measurement and management, [5] is usually connected to quality.

In summary, software estimation is an error prone and difficult problem to address. Among others, one of the most important difficulties is related to its intrinsic uncertainty and, indeed, software development processes are crowded of various forms of uncertainty. For example, the same action (such as a design decision) can generate different outcomes and, equivalently, the same sets of observations can lead to different states. Uncertainty also comes from the consideration of human factors such as the ability and expertise of both the development team and the managers. Thus, there have been several approaches to software estimation that use models of uncertainty, such as Bayesian Belief Networks (BBNs), [40] a direct use of Bayes theorem, [14] or fuzzy logic. [1] In this paper we also propose the use of BBNs. Specifically, we introduce an approach for building BBN models semi-automatically using machine learning techniques. Section 6 discusses further related work on the application of BBNs for making software estimations and the main differences with our work.

3. Bayesian Belief Networks

Bayesian Belief Networks (BBNs) are acyclic directed graphs with nodes X_i that stand for different variables which can take one among several values from a domain D_i. Each node can be connected to an arbitrary number of neighbours. This relation is characterized with a conditional probability distribution and can be used to define causal dependencies. For example, if node X is connected to node Y, then X can be interpreted as a cause of Y (see Fig. 1). Moreover, if Y is also connected to Z, X affects Y and the ultimate values of Y do also propagate to Z. However, Z and X are *conditionally independent*, which means that:*

$$P(Z/Y,X) = P(Z/Y)$$

In fact, in BBNs, the joint probability distribution P for the net variables satisfies the *Markov condition*: each variable X_i is conditionally independent of the set of all its non-descendents given its parents. [33] Thus, this distribution can be factorized as:

$$P(X_1,\ldots X_n) = \prod P(X_i|Pa(X_i))$$

Hence, BBNs are efficient models for taking into account a large number of causes simultaneously and measuring the probability of effects to take place. In short, BBNs consist of:

- A number of direct links represented as arcs between nodes which stand for concepts, metrics in the case of projects estimations.
- A number of Conditional Probability Tables (CPTs) which state its likelihood for every pair ⟨causes, effect⟩

For example, Fig. 2 shows a small BBN, where nodes are the number of old and new use cases, the number of new classes, and the cost of the project. All variables can take values *high* and *low*. Each variable has an associated probability distribution. For root variables (as Old and New use cases), this distribution reflects the *a priori* probability of each value. In the case of non-root variables (New classes and Cost), it reflects the conditional probability of having each value depending on the values of the parent variables.

*The general notion of conditional independence in BBNs leads to the graphical criterium of d-separation. [35]

Copyright: the authors

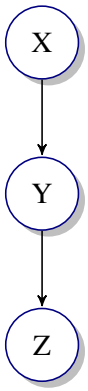


Fig. 1. Example for illustrating BBN causes and conditional independence.

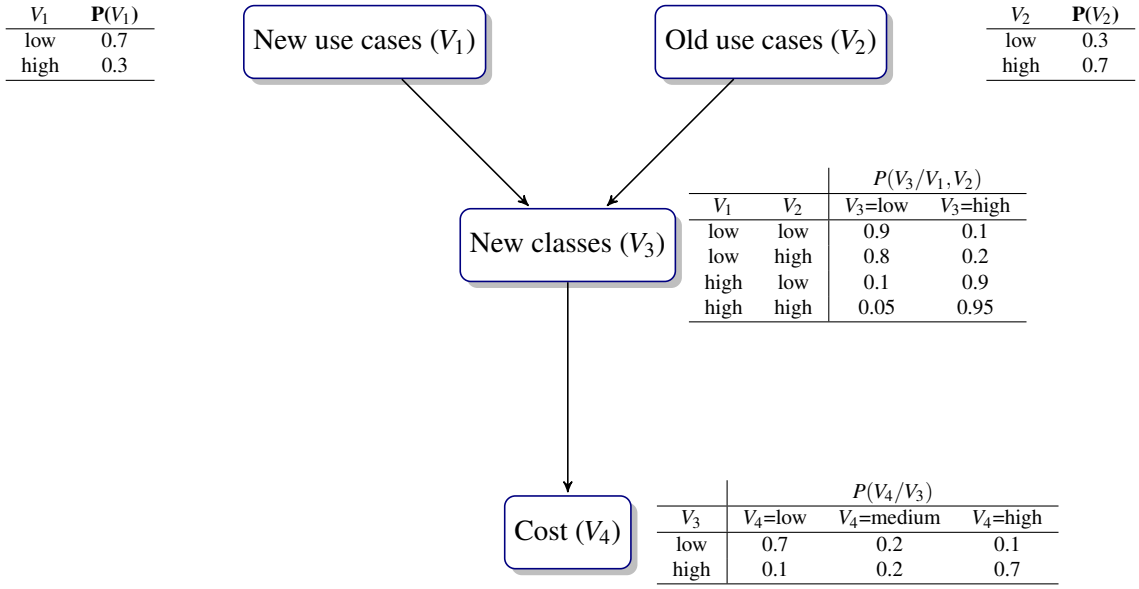
Designing a BBN (which is expressed as a causal probabilistic Directed Acyclic Graph, DAG) for a particular application domain requires considering a number of issues:

- Identification of the relevant variables, which will conform the nodes of the graph. In the case of software estimations, the variables will include all elements that affect the estimation of costs according to the selected metrics. This step also incorporates the definition of the variables values. Variables can have discrete or continuous values. If it is a discrete variable, the specific values have to be defined. For continuous variables the Gaussian distribution is the most common one. However, managing continuous variables is more difficult in general and the most often used alternative is to discretize them.
- Identification of the causal dependencies among variables. If a variable X might affect the value of another variable Y, an edge is defined in the graph between the corresponding nodes, taking special care of not creating a cycle, and considering the notion of conditionally independence (conditionally independent variables do not have a direct connection).
- Parameterization of the probabilistic information of the graph. This step requires defining:
  - The prior probabilities for each root node in the graph, and
  - The conditional probability tables (CPTs) associated with each non-root node, that quantify the relationships between nodes.

BBNs present a number of advantages, with a significant impact in the context of this paper:

- If an event is known to happen (the value of a node is known), the BBN can be fed with probability 1.0 for that value. However, any probability distribution can be used. This is, BBNs fairly generalize the behaviour of many other decision systems, which are often deterministic. For example, in the likely case of not knowing the probability of some input variables (also referred as decision variables), it is usually assumed that they are all equally likely though other scenarios can be defined as well.
- Although the most typical reasoning approach is a straight application of the definition of conditional probabilities, which are updated according to the Bayes Theorem, there are different ways of applying inference. Some of them are, but not necessarily limited to: variable elimination, mini-bucket elimination or clique propagation. In general, it is possible to run different inference algorithms over the same model.
- Explanations can be easily generated. They result from the causal links that affected (up to a given probability which does not exceed a given threshold) the node under consideration. The usage of probabilities allows designers to carefully review the behaviour of the BBN.
- Since BBNs are fully probabilistic methods, other

Copyright: the authors



V_1	**P**(V_1)
low	0.7
high	0.3

V_2	**P**(V_2)
low	0.3
high	0.7

		$P(V_3/V_1,V_2)$	
V_1	V_2	V_3=low	V_3=high
low	low	0.9	0.1
low	high	0.8	0.2
high	low	0.1	0.9
high	high	0.05	0.95

	$P(V_4/V_3)$		
V_3	V_4=low	V_4=medium	V_4=high
low	0.7	0.2	0.1
high	0.1	0.2	0.7

Fig. 2. Small fragment of a BBN that estimates the cost of a software project based on number of new and old classes.

methods for estimating the *a priori* probabilities (such as max-likelihood estimation) or learning the structure of the BBN are possible. In this regard, top-down inference (also known as predictive inference) can be seen as a generalization of Markov stochastic models.

4. Proposed Approach to Generate BBNs for Making Project Estimations

A first method to generate BBNs for making project estimations is to design the topology of each BBN manually and then fill the corresponding CPTs also manually. This approach has important drawbacks when there are many variables to consider, as in the present case:

- It is not trivial to generate the net structure. Some apparent causal relationships do not work, or do not yield the expected results, while others can be hidden.
- Filling CPTs by hand can be a very time consuming and error-prone task.

These reasons motivate applying a semi-automatic approach, where an initial model is generated automatically and then it is supervised/modified by human experts. To carry out the automatic part we have selected one off-the-shelf data mining tool, Weka. [20] Weka provides a collection of machine learning algorithms for tasks as data preprocessing, classification, regression, clustering, association rules, and visualization. Particularly, Weka can be used to automatically design BBNs from data. In this sense, Weka offers a rich variety of algorithms and it can be very useful for the phase of model building and evaluation. Once the BBN models have been generated, revised and evaluated, they can be introduced in the APES tool to be used. The first approach we followed for learning BBNs using Weka was related to previous approaches that used machine learning for generating project estimations. The approach can be defined as:

- Given a set of examples (projects data) in terms of the values of some attributes (project metrics) and a class value (cost, quality, or risk),
- obtain a classifier (program) that takes as input a new project and returns the class value for that project (its cost, quality o risk, depending on the selected class)

So, we have to learn three classifiers. Thus, for each project estimator (cost, quality and risk) we

Copyright: the authors

used Weka to generate them. Weka incorporates not only techniques for learning BBNs, but also many classification and regression techniques, as well as many techniques for preprocessing the data. Therefore, we experimented with many different combinations of machine learning and filtering techniques. The following is a summary of some of the results we obtained with the data provided by GMV and Skysoft.

4.1. A First Approach: BBN for the whole Project (one-step BBN)

One-step BBNs generation is the simplest way of producing BBNs using Weka. With this approach, the final BBN for predicting the value of the corresponding estimation variable is generated directly, using variables of all project stages. The process we follow for building BBNs consists of the following steps:

1. Generation of data of previous projects. We describe in the Experimental set-up section some of the features we used.
2. Preprocessing the data, which can be divided in:
   (a) Remove attributes considered irrelevant for predicting the class.
   (b) Replace missing values and remove useless values.
   (c) Discretize numeric variables.
3. Select the more relevant attributes for the class.
4. Generation of the model, which implies selecting an algorithm provided by Weka and giving appropriate values to its parameters.
5. Evaluation of the model, computing some score useful to determine the utility of the resulting BBN.

From these steps, 2.b), 2.c), 3 and 4 can be performed automatically using Weka. However, some of them require selecting an algorithm from a pool and then to select the parameters values of that algorithm. Thus, the main parameters for step 2.c) are, in our case:

- Number of bins: for building BBNs, the recommended number of bins is a small value (between 3 and 5). On one hand, because it is desirable to be able of semantically interpreting the meaning of each bin, and on the other hand for efficiency reasons. The size of the CPTs learned for the BBNs grows exponentially with the number of values of the BBN nodes; that is, the number of intervals of the corresponding attributes.
- A parameter that indicates whether bins should be of equal length or of equal frequency. We tend to prefer equal frequency since equal length can lead to unbalanced intervals. However, equal frequency usually means preventing the occurrence of outliers. Besides, with equal frequency, many occurrences of a continuous value can cause the occurrences to be assigned to different bins. [27,26]

Discretization can also be done by hand. When possible, we believe this is the best option given that it solves the described problems about the semantic meaning of bins, and it does not assume neither equal frequency nor equal length bins. For the evaluation of the model, Weka offers some measures related with the model performance when it learns classifiers. The most frequently used measure is the percentage of correctly classified instances (accuracy) and the confusion matrix, which specifies the number of instances of each class that are correctly and incorrectly classified. The standard way of estimating such measures is by performing a cross-validation process. In such case, Weka provides an estimate of both measures for new data, i.e., data not used during the generation of the BBN.

As explained before, the general process should be instantiated for each particular case, defining clearly each step, including the algorithms and the parameters to be used for the automatic tasks. Thus, for generating the first cost model we performed the following specific steps at each phase of the mentioned process:

1. Generation of data: provided by GMV and Skysoft

Copyright: the authors

2. In the preprocessing phase:

   - Attributes considered not relevant were removed, as for example the project name and the project dates. We maintain all data about delays on the schedule so that relative time can be considered by the model. Then, we also removed attributes that are highly related with the class, as all the cost estimations and other attributes known at the end of the project that can be used also to measure the cost.
   - Then, two Weka filters were applied: *RemoveUseless*, and *ReplaceMissingValues*. The former removes useless attributes i.e. those that do not vary at all or that vary too much. The latter generates probable values given the rest of the training data for the missing values of attributes of some instances.
   - Discretization was performed in three bins of equal frequency.
   - The *AttributeSelection* method was the default one (*BestFirst* with *CfsSubsetEval*) that resulted in a selection of 15 attributes. The *BestFirst* method searches the space of attribute subsets using a Hill-Climbing search algorithm augmented with backtracking. The evaluation function, *CfsSubsetEval*, considers the individual predictive ability of each attribute and the degree of redundancy between them.

3. In the BBN generation phase we applied two of the algorithms defined in Weka for learning BBNs: *K2* and *TabuSearch*. They build a classifier performing a search process (adding or deleting arcs) in order to generate the net structure that maximizes some score (as, for example, the log likelihood of the data). Specifically, the *K2* algorithm performs Hill-Climbing search restricted by a particular order of the variables. [15,9] The *TabuSearch* algorithm uses tabu search, [8] which is similar to Hill Climbing with additional constraints.

The resulting BBN for estimating cost, measured as total man power hours, is shown in Fig. 3. We generated also BBNs for estimating quality and risk following the explained process, though we do not include the resulting models in the paper. After analyzing the resulting BBNs and the results on accuracy (shown on the Experiments section), we detected important problems with the approach, summarized below. This led us to define an alternative approach which is described in the next section.

The main problems of the approach are:

- The learned BBNs were in the opposite direction with respect to the ones that users are expecting, where the estimation variable is on top, the explanatory variables are in the bottom and the estimation variable points to the rest. This effect occurs because we are generating automatically a BBN classifier for predicting the class. However, this BBN is not necessarily a correct causal model. In correct causal models arcs follow the direction of a causal process, observing causality relations occurring in real world. The obtained BBNs were generated automatically with no more information other than the data set, and the learning algorithm is not aware of the real causal relations among nodes. In other words, statistical correlation does not always imply causality. For generating causal BBNs it is usually necessary some kind of human supervision. Also, it is important to observe the temporal order of events given that causes always occur before their effects. In most projects, the life cycle can be defined linearly from a temporal point of view. For example with the following phases: proposal, requirements specification, design, construction, installation and end of project. In the next section we take advantage of this observation to come up with better models.
- Most of the generated BBNs included variables whose value is only known at the middle/end of the project. This effect occurs because the classifier takes as class the main variable to predict, and therefore Weka does not generate classifiers for predicting others. When using the BBN, this means assigning *a posteriori* uniform distributions for unknown variables at a given project

Copyright: the authors

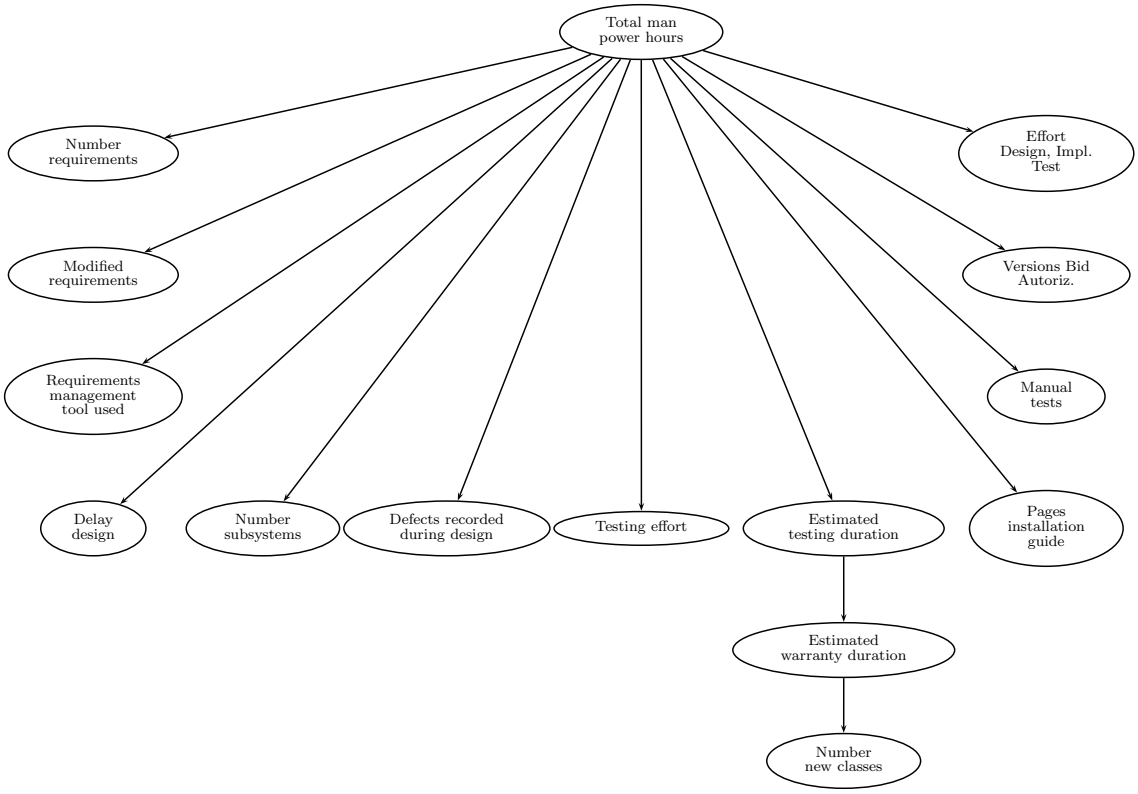


Fig. 3. Resulting BBN from the one-step learning process.

phase, which is a source of inaccuracy for predictions. However, we wanted to generate estimations during all phases. The alternative approach described in the following section allows the user to estimate other variables than the main class.

4.2. Second Approach: BBN Based on Project Phases (multi-step BBN)

The second approach we followed tried to solve the problems generated by the one-step BBN approach. This approach considers the projects phases and has a step of human intervention for revising and repairing BBNs learned automatically, specifically the causal relations they contain. For each core variable to be predicted (as for example the project cost) the approach builds several BBNs (partial BBNs) that are then merged into a global BBN for the core variable.

The new process has three main phases:

- Phase I: **Preparation of the data files for each project phase**. This phase (illustrated in Fig. 4) is composed of two steps:
  - (I.1) *Preprocessing the data*, which can be divided in:
    1. Remove attributes considered irrelevant for predicting the class.
    2. Remove missing values and remove useless values.
    3. Discretize numeric variables.
  - (I.2) *Generation of the data files for each project phase*. At the end of (I.1) we have a preprocessed data file with all projects attributes. This steps takes this file and generates a data file for each project phase. First of all, the set of project phases has to be defined. Then, each input variable (metric) is assigned to one of the phases in the set. The data file for each phase contains the variables that are known at the end of the phase, so they can be

Copyright: the authors

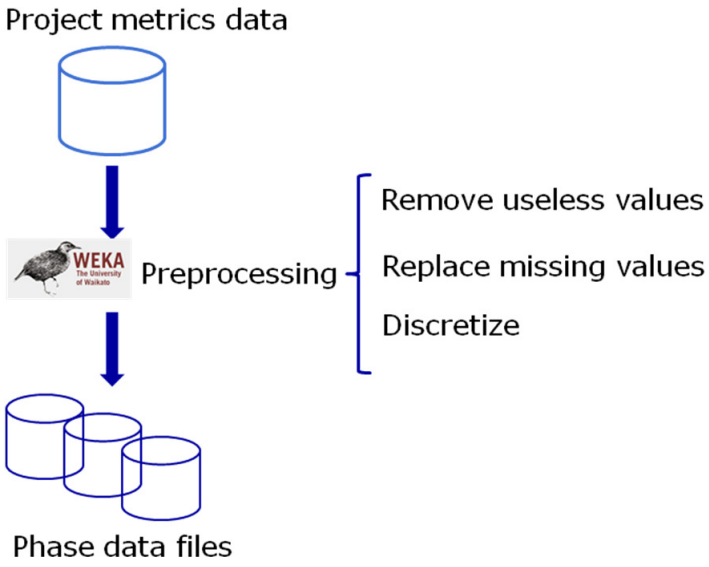


Fig. 4. Multi-step BBN building process. Phase I.

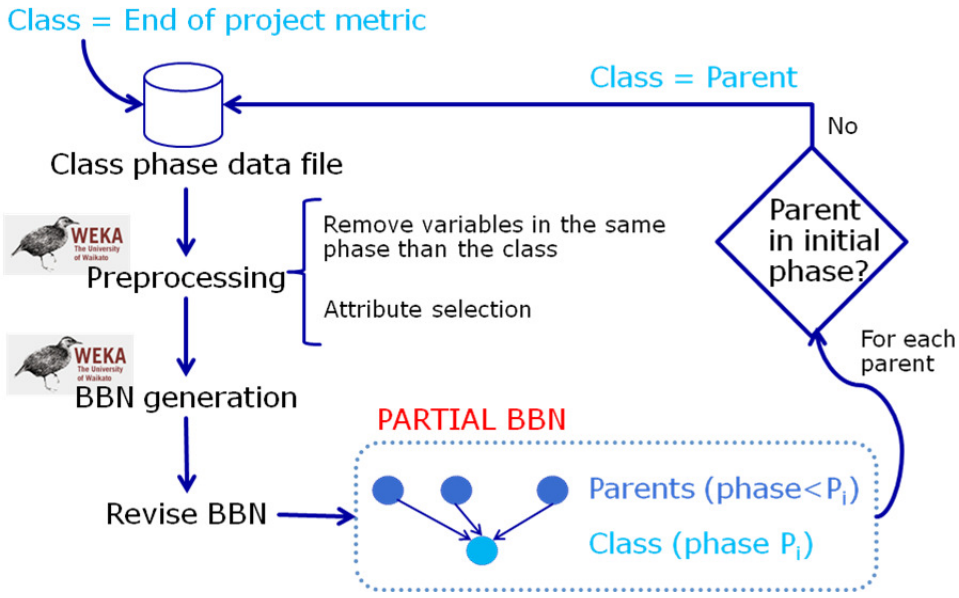


Fig. 5. Multi-step BBN building process. Phase II.

Copyright: the authors

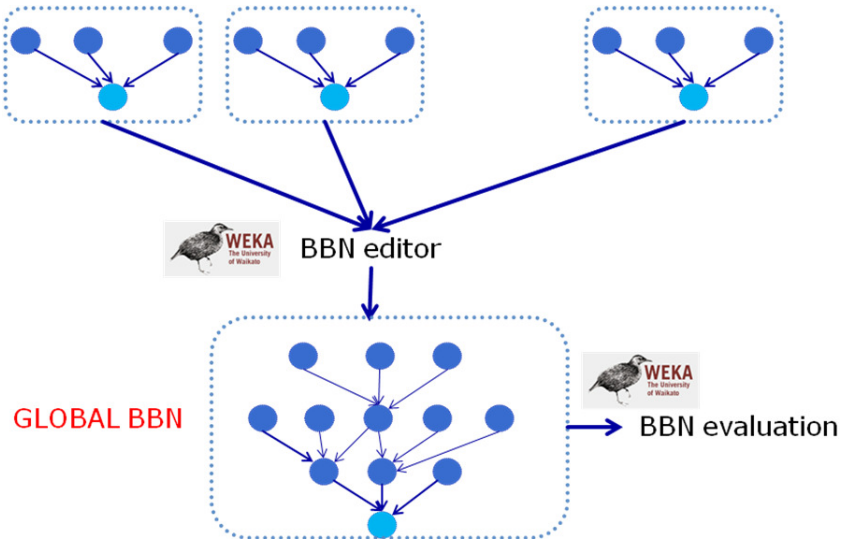


Fig. 6. Multi-step BBN building process. Phase III.

used to predict the values of variables on the next phase. Thus, we limit the number of causal relations for the learning algorithm, given that all causes occur often before their effect. The data file for a project phase contains all the variables in the phase and all variables in previous phases. Thus, if the overall project has N temporal phases $P_1, \ldots, P_N$, we obtain $N-1$ data files, such that the data file for phase P_i contains all attributes of all phases P_j with $j \leqslant i$.

- Phase II: **Generation of partial BBNs**. This step takes as input a class variable, and the phase data file corresponding to the project phase of this variable. Initially, this variable is the corresponding main estimation variable (cost quality or risk). It outputs a set of partial BBNs for predicting this core variable. The process, illustrated in Fig. 5, is recursive and works backwards. It comprises the following steps:
  - (II.1) *Preprocessing the data*. This preprocess consists of:
    1. Remove the variables from the data file which are in the same project phase than the class. These variables can not be used for estimating the class given that they are known at the same time.
    2. Select the more relevant attributes for the class.
  - (II.2) *BBN generation*: generation of the model, selecting first one of the algorithms provided by Weka.
  - (II.3) *BBN revision*: model refinement by a human expert, which needs two tasks performed manually:
    - To remove all relations not involving the class, as we want to identify direct causes of the class and these relations involve variables that are conditionally independent of it.
    - To analyze arcs (if any) issued from the class in order to determine whether to delete or reverse the arc.
  - (II.4) *Recusive step*. Now we have a partial BBN in which the class variable has only parents of previous phases, and these parents have no causes. Then, for each of these parents if the parent does not belong to the first project phase, a new BBN is generated in a recursive call. In this call, the class variable is the corresponding parent and the data file is the data file for its project phase. Each call to the recursive step generates a new partial BBN for predicting the class variable. The recursive step is performed recursively backwards until all root variables in the main BBN belong to the first phase.

Copyright: the authors

- Phase III: **Generation of the global BBN**. In this phase, illustrated in Fig. 6, all partial BBNs are integrated into a global one. The integration is simple: join the shared nodes (variables) of the partial nets. When joining a shared node, the set of links starting at it in the global net is the union of the links starting at the variable in all partial nets. The procedure for building the partial BBNs guarantees the same variable has always the same parents in all partial nets. Therefore, the merging procees does not affect d-separation. After generating the global BBN, it is manually revised.

Currently, we maintain three separate BBN models for the three relevant variables to be predicted: cost, quality and risk. However, these three models can also be integrated into a larger BBN using the method described above for integrating the partial nets, given that the parents of every variable remain the same in the different models.

The described process allows experts to use all available information for building the net. The BBN estimates the corresponding class variable. The attribute selection step selects the relevant variables for predicting the class. Thus, only these variables can appear in the resulting BBN. However, the relevant variables for predicting each of its causes can be different than those selected for the class. For this reason, we need the recursive step, in which the attribute selection step is applied again and recursively for each cause, using all variables susceptible of being used for estimating it.

The resulting BBNs can be used from the start of the project, by propagating the knowledge about variables whose values are known since the very beginning of the project. Then, after new intermediate variable values are known, their values are fixed and propagated forward throughout the BBN. Also, even if they are sometimes large, they can be easily understood by humans, as was done within the project by people from GMV and Skysoft.

The defined process is general in the sense it could be applied in any domain with temporal phases and using any machine learning tool providing the adequate algorithms. However, we have an specific domain (SE prediction) and an specific machine learning tool (Weka). Now, we explain how the steps involving the use of Weka have been instantiated for our domain. These steps are those marked with the Weka symbol in Figs. 4, 5 and 6.

- Discretization (step I.1.3): some variables have been discretized manually (by GMV and Skysoft). The maximum number of bins allowed for the manual discretization was five, though finally most of them have four bins. Seventy-four of 126 variables were discretized manually. The remaining variables are automatically discretized using Weka. We chose an automatic discretization in three bins, in order to generate BBNs of a reasonable complexity and with bins of equal frequency, in order to obtain balanced intervals.
- For the attribute selection method in step II.1.2 we used a evaluator based on the information gain (*infoGainAttributeEval*) with a Ranker search. With these parameters we can limit the number of selected attributes to six. This value limits the number of parents for each node in the BBNs.
- For the generation of the model in step II.2 we chose an algorithm that performs conditional independence tests (ICS). [41] This algorithm identifies variables that are conditionally independent and builds a DAG that is consistent with these relations. Usually, this algorithm selects the class as final effect, instead of as initial cause, as the algorithm described in the previous section.

5. Experiments and Results

In this section we first describe the experimental setup, and then the results we have obtained using the two processes described above: the one-step BBN generation and the multi-step BBN generation methods. Then, we compare these results with the application of other machine learning techniques.

5.1. Experimental Setup

In order to train and test our approaches, we selected data from projects developed by GMV and Skysoft over the last years. In relation to GMV projects:

- The projects were developed during the last two years. That means that some of the projects were

Copyright: the authors

still in final phases of the development, and ultimate data (like data from the warranty period) could not be accessed.
- We expected to find some kind of clustering between those projects related to a particular software asset, but we did not prejudge that.
- The amount of projects initially selected, almost thirty, were considered enough, because the BBN model does not need very large volumes of data in order to converge.
- All the projects use, at least, one of these GMV Software Assets. Moreover, some of them use even two GMV Software Assets. A pre-selected set of 28 projects was considered, each of these using one of the software assets. This general rule was finally discarded, in order to add new projects to the list. That is, new projects were added that were not built using a previous software asset.
- All the software products were used as operational software systems.
- The projects belonged to Aerospace Sector, exclusively.
- Different Divisions of GMV Aerospace Directorate of Operations developed the projects.
- The products were deployed in Europe, Asia and USA, covering a wide range of customers.
- Data was gathered from various sources: economic internal tool, knowledge manager of the company, projects documentation, and interviews with project managers.
- For a preselected set of 34 projects, finally only 15 projects from GMV Spain were considered, due to lack of data or lack of representativity. Some of the projects were just studies, and did not have enough relevance.

In the case of Skysoft, projects were too recent (less than five years old), whose output included a software product in the category of tools, prototypes or operational software. Fourteen projects were selected to be used in the project out of an initial set of 20.

A very important attribute of a metric is the moment of the project life cycle at which it is known. For instance, the data about the team in a project. This is a fact usually known at the beginning of the project, and remains stable unless there are changes during the project. This was a very important characteristic, since we wanted to generate BBNs that were based initially on metrics are available as soon as possible. Phases are possibly the most critical attributes for a metric, since they define the moments at which every parameter can be known. We have defined six different phases:

- Phase I: Proposal's time.
- Phase II: Specification of requirements.
- Phase III: Design.
- Phase IV: Coding & Construction.
- Phase V: Installation.
- Phase VI: End of project: considering the Warranty period. (The end of project is at end of Warranty, not when the software is delivered for its use).

Fig. 7 shows the temporal constraints between phases for building the BBNs using the multi-step approach. Thus, the model for variables at a phase are constrained to use only variables in previous phases. This is denoted by the arrows in the figure.

Some projects did not have all these phases; in particular, some projects did not have the Specifications phase. For these projects, the Specifications phase was assumed to have no duration. Apart from that, all the projects fit very well into this categorization of phases.

We were given 126 different features of projects, which were categorized in different types:

- Economic data, with a difference between features whose value was known at the start and the ones that were known at the end. Examples are: Gross invoicing (start and end figures), Man power cost, Planned man power hours, Contract duration, Costs per hour of the project, Percentage of extra costs w. r. t. Gross income, etc.
- Planning and scheduling data. Examples are: Number of project milestones, Duration of each phase, or Delay of each phase.
- Phase II features: Number of pages in documents, Number of modified requirements, etc.

Copyright: the authors

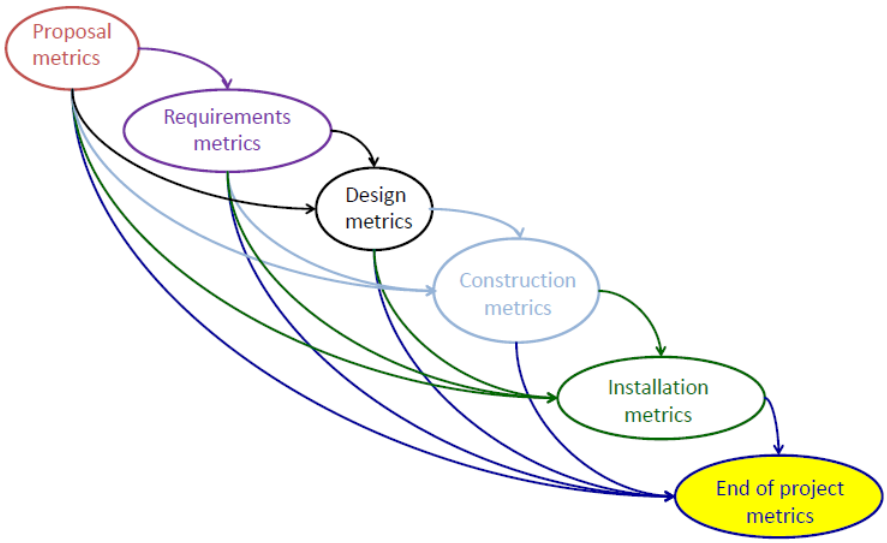


Fig. 7. Temporal causal constraints over the project phases.

- Phase III features: Number of classes (new and old), subsystems (new and old), interfaces (new and old), etc.
- Phase IV features: LOC, Percentage of the project devoted to code production or unit testing, etc.
- Team: Size, Studies, Experience (team and leader), Dedication, etc.
- Testing: Number of defects detected per phase, Testing effort, etc.
- Phase V: Number of pages of the User's Manual, Number of pages of Design Justification's file, Number of releases delivered, etc.
- Phase VI: Hours spent during the warranty period, Number of releases that provided new functionality, Total number of risks being contemplated during the project's life-cycle, Percentage of effort devoted by the Project Management, etc.

Regarding the classes of the estimation models, we selected:

- Cost: the number of total man power hours, or total effort. The reason for using it instead of the real cost was that different rates per hour were applied between both companies.
- Quality: the number of defects (Software Problem Reports, SPR) raised during the warranty phase.
- Risk: we used two metrics; the total number of risks in the project and the sum of all delays with respect the initial schedule, as a way to predict the deviation in time.

Fig. 8 shows the BBN for estimating costs obtained by applying the phase-dependent process. Root nodes correspond to metrics known at the beginning of the project (in the proposal or specification phase). Then, metrics belonging to a specific phase (as for example the number of subsystems in the design phase) have as causes only variables belonging to previous phases, and effects belonging to later phases.

The BBN for estimating quality is shown in Fig. 9. The variable to be predicted is *Number of SPRs Ph4*, that represents the number of Software Problem Reports raised during the warranty period, at the end of the project.

Finally, Fig. 10 shows the BBN for estimating risk using as metric the total number of risks contemplated during the project's life-cycle.

To measure the performance of the obtained models we use the accuracy (ACC) defined by Eq. (1), and the Matthews correlation coefficient (MCC) for multiple classes, [29,23] defined in Eq. (2). This coefficient overcomes the weaknesses of the accuracy when the classes are not well balanced. These

Copyright: the authors

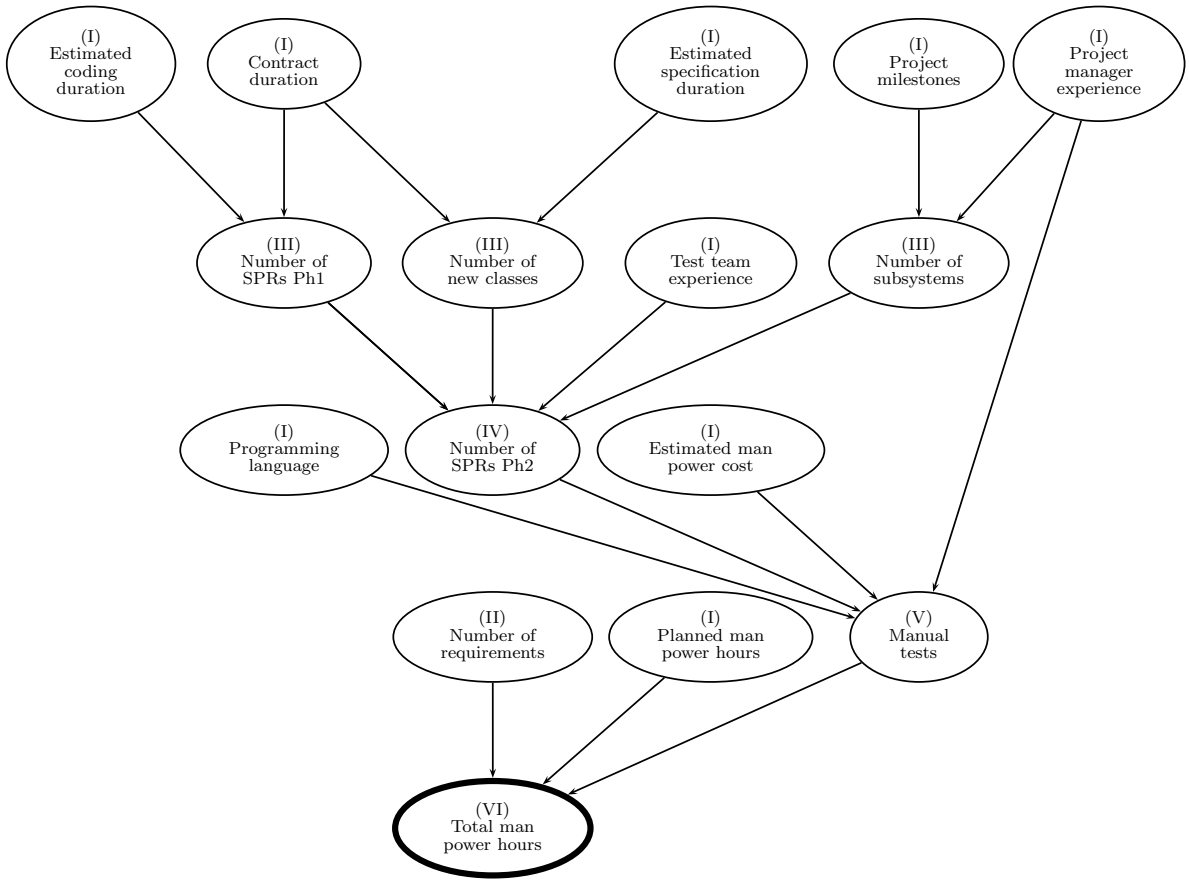


Fig. 8. Multi-step BBN for estimating cost. Project phases: (I) Proposal, (II) Requirements, (III) Design, (IV) Coding, (V): Instalation, (VI): End.

measures are defined in terms of the sample size, S, the number of classes, N, and the entries of the confusion matrix, denoted as C. Each C_{ij} is the number of instances with true class i that have been assigned to class j by the classifier.

$$ACC = \frac{\sum_{k=1}^{N} C_{kk}}{S}. \quad (1)$$

$$MCC = \frac{\sum_{k,l,m=1}^{N} C_{kk}C_{ml} - C_{lk}C_{km}}{\sqrt{\sum_{k=1}^{N}\left(\sum_{l=1}^{N} C_{lk}\right)\left(\sum_{\substack{f,g=1 \\ f\neq k}}^{N} C_{gf}\right)}\sqrt{\sum_{k=1}^{N}\left(\sum_{l=1}^{N} C_{kl}\right)\left(\sum_{\substack{f,g=1 \\ f\neq k}}^{N} C_{fg}\right)}}. \quad (2)$$

MCC provides a value in the range [-1,1], where the perfect classification has value 1.

5.2. *Results of the One-Step BBN Training*

In the evaluation phase, we analyzed the measures provided by Weka, selecting an evaluation by means of cross-validation. The total number of projects in the data set is 30 and we selected 3 folds: 20 projects for training and 10 for testing. The accuracy of the obtained cost model, measured as the number of correctly classified instances, was 86,6%, which is quite high. The MCC is 0.80. The accuracy (number of correctly classified instances) provided by Weka with cross-validation (3 folds) for the quality model was accuracy 100% and MCC 1; and for the risk model the accuracy is 79.3% and the MCC is 0.69.

Copyright: the authors

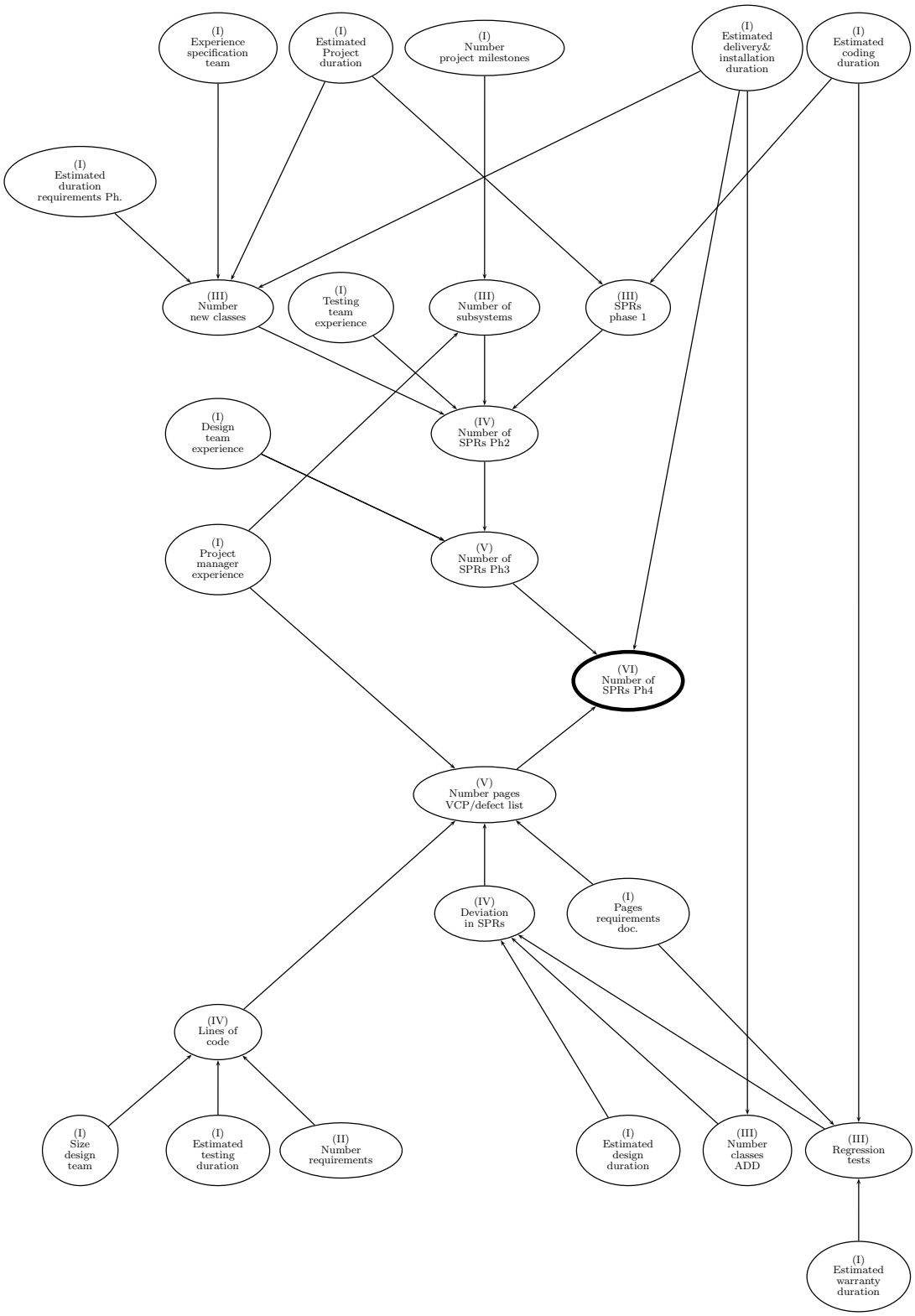


Fig. 9. Multi-step BBN for estimating quality. Project phases: (I) Proposal, (II) Requirements, (III) Design, (IV) Coding, (V): Instalation, (VI): End.

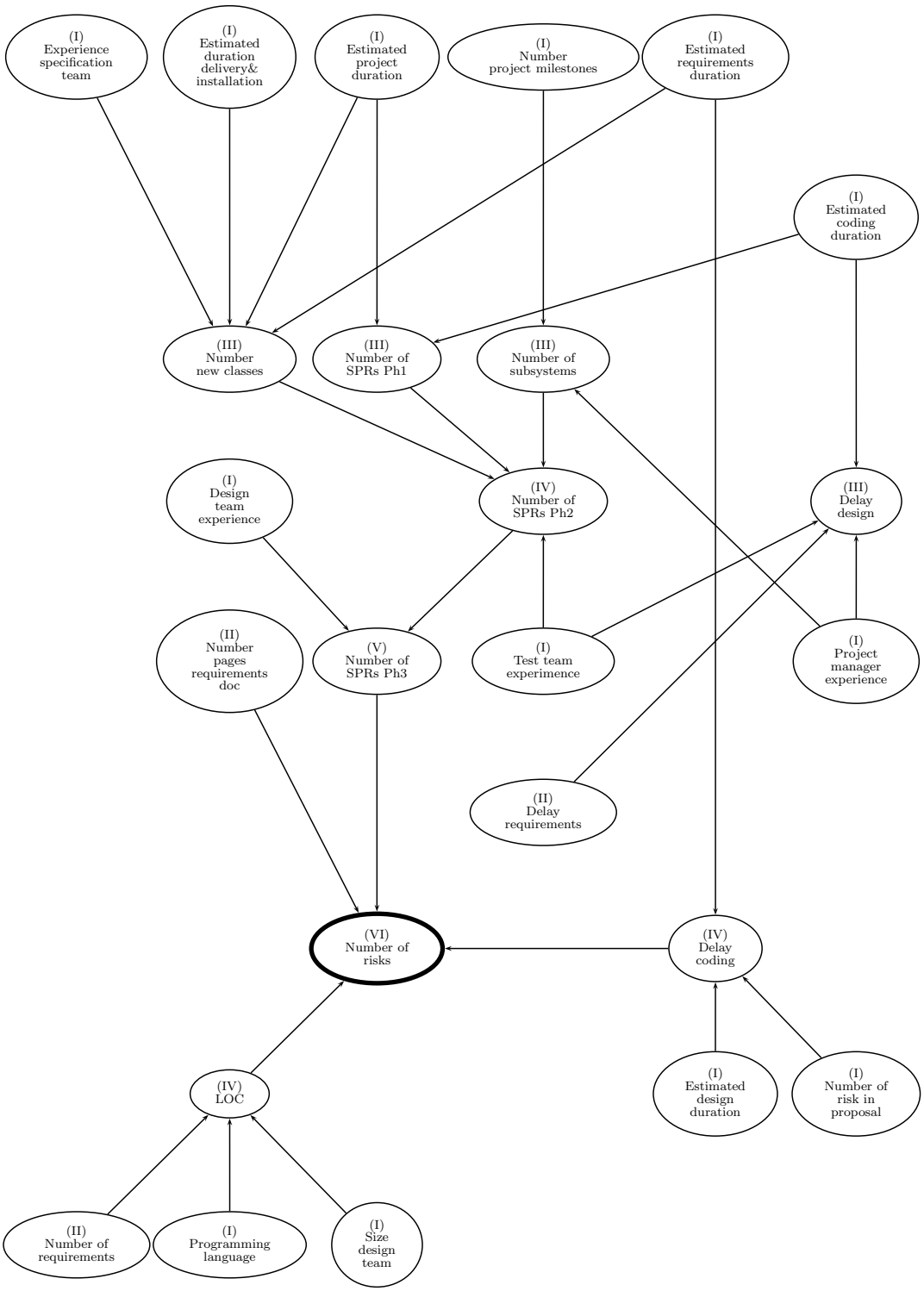


Fig. 10. Multi-step BBN for estimating risk. Project phases: (I) Proposal, (II) Requirements, (III) Design, (IV) Coding, (V): Instalation, (VI): End.

5.3. Results of Multi-Step BBN Training

To evaluate the multi-step BBNs we perform also 3-fold cross validation. Now, the structure of the net is fixed over the cross validation process. Since we are using all data to build the initial structure of this net, it could be considered that we are testing with training data. However, the initial structure is modified by humans to obtain the final one, and this makes the bias smaller. The solution to this issue would be to ask the humans to build 10 different BBNs, providing them 10 initial models built with the training data in the corresponding cross-validation fold. However, this overload for humans is not viable in most real-world projects.

For the phase-dependent cost BBN, the accuracy is 63.3% and the MCC is 0.45. This accuracy is lower than the accuracy obtained by the one-step process. Also there is a decrease in the MCC. However, the BBN we have now can be seen as a correct causal model, given that it was revised and modified by human experts. Also, the causal relations are consistent with the temporal development of projects. Thus, initial causes (nodes without parents) are variables that are known in the first or second phase (proposal and requirements definition phases), and every cause of an effect is known before the effect, since it belongs to a previous phase.

While the phase-dependent model is semantically correct, the one-step model is adjusted to the data only syntactically and it surely suffers from over-fitting. This is because the specific ML algorithm used tries to adjust the BBN model to the data. Though many ML algorithms do not guarantee the best adjustment, they, if possible, achieve a good one. So, other models (including the human corrected one) can have less accuracy. We assume we have to pay this prize to obtain a semantically correct model that can provide good explanations and thus is understandable and usable by humans.

Another cause of the magnitude of the decrement in accuracy is that CPTs in the phase-dependent BBN are bigger than CPTs in the one-step BBN. In the former, we have nodes with up to four parents, while, in the latter, nodes only have one parent. CPTs with more nodes also imply the dataset to train the CPTs should be larger. At least, it should contain all combinations of possible values of causes-effect for all causal relations. Of course, this is also dependent on the number and type of bins in which variables have been discretized. In the multi-step approach many variables have been discretized manually. Thus, each bin has an adequate semantic defined by the expert. However, with manual discretizations many variables (including the class) are not well balanced. In spite of these problems, we consider the BBN structure we obtained to be correct, and it can be easily trained again to obtain larger accuracy with more projects data when available.

The accuracy for the phase-dependent quality BBN is 63.3%, and the MCC is 0. The MCC has a value of zero when the confusion matrix is all zeros but for one single column. This means that the performance of the classifier is similar to always returning the majority class. This bad perfomance is due to the same problems we descibed for the cost model. However, the same BBN structure can be trained with future data. We have made an experiment training the quality net with synthetic data generated using SMOTE (Synthetic Minority Oversampling TEchnique). [11] We fixed the parameter defining the nearest neighbours to three. The other parameters take the default values. Then, we applied the procedure twenty times. Thus, we obtained a training set of 720 instances. With this set, the 3-fold cross-validation provides an accuracy of 98.6% and a MCC of 0.98.

For the phase-dependent risk BBN the accuracy is 44.8% and the MCC is 0.2. As before, this values have been obtained from a cross-validation process with 3 folds. The reasons for this decrease in accuracy (when compared to the one-step BBNs) are similar to the ones explained for the cost model. Particularly, note that in the risk phase-dependent BBN the class variable has four parents (more than the class variables for the other two models).

5.4. Comparison with other Machine Learning Techniques

Weka offers many other techniques for performing machine learning tasks. In this section we include a summary of some experiments we have performed

Copyright: the authors

for predicting cost using some of them. This experiment is merely informative, given that the models we compare with do not have the features of the BBN generated by the multi-step approach. Specifically, they do not allow us to incorporate new information as the project advances, and many of them do not provide explanations.

In a first experiment (E1) we apply the same discretization we used for generating the multi-step BBN to estimate cost. Then, we select only those attributes appearing in this BBN. Table 1 shows the accuracy, second column, the RRSE (Root Relative Squared Error), third column, and the MCC (Matthews Correlation Coefficient), fourth column, for the estimations obtained with the technique in the first column. The parameters of the corresponding technique have the default values provided by Weka except for the parameter k for IBk, for which we have chosen a value of 3. The accuracy and the RRSE are computed by using a 3-fold cross-validation process. The one-step BBN approach here is different than the one used previously, given that now the attributes are selected in a different way and the discretization has been done manually for some attributes (as it was described for the multi-step method).

Also, we have performed the same experiment, but not using only the attributes in the multi-step BBN (E2). For this case, the atribute selection was done using the algorithm that was chosen for the multi-step approach, but without restricting the number of selected attributes. The results are shown in Table 1, third and four columns.

As can be observed in this table, there are other machine learning techniques apart from BBNs that can be useful for the kind of project estimations we are dealing with. These techniques obtain similar, or in some cases better, accuracy than multi-step BBNs. However, none of these models has the power of expressing causal relations as BBNs have. In fact, some of these techniques, as Logistic, Multilayer Perceptron, SMO, IBk and Bagging, generate models that are very difficult to be interpreted/understood by humans. Decision trees could be more adequate in relation to understandability. Also, we encountered the same problem we had with one-step BBNs (i.e. not all variables are known at the same time and we wanted to perform estimations at different instants of the project, starting from the beginning of the project). To solve this problem, a process similar to the one we defined for phase-dependent BBNs could be defined.

Another experiment consisted on using directly the numerical data (without discretization) and applying the attribute selection default algorithm. Results are shown in Table 2. Here, the accuracy is computed as the Pearson correlation coefficient. As before, we used 3-fold cross-validation. For the cost model, apart from the Multilayer perceptron and Support Vector Machines, the accuracy of most of the techniques is very low. Also, the use of Multilayer perceptrons or Support Vector Machines has the same problems we identified previously: the lack of explanatory power and that they can not manage variables that are not known at the same time. However, for the quality model, M5P obtains a good performance and the lowest RRSE. As our multi-step BBN, this technique provides a good explanatory level. Finally, for the risk model, all techniques present a bad performance.

6. Related Work

Some other works have proposed the use of BBNs for making software project estimations. A recent survey was carried out by Radlinski. [37] In this paper, BBNs are classified following their topology in four groups:

- Converging star: the topology resembles a star with links from each predictor variable to a single dependent variable.
- Naive bayes: diverging star with links from a single dependent variable to several predictors.
- Causal BBN: general BBN.
- Dynamic BBN: set of sequentially linked BBNs, where each instance reflects the state of the system at a specific point in time.

While the single-step approach we present here is similar to a Naive Bayes structure, the multi-step approach we propose results in a more general BBN.

Copyright: the authors

Table 1. Accuracy of different learning techniques using discrete data.

		E1			E2		
Model	**Learning Technique**	**ACC**	**RRSE**	**MCC**	**ACC**	**RRSE**	**MCC**
Cost	Multinomial Logistic Regression	63.3%	120.2%	0.28	60%	119.4%	0.35
	Multilayer Perceptron	70%	103.4%	0.34	76.6%	84.1%	0.47
	Support Vector Machine (SMO)	70%	103.4%	0.34	76.6%	105.6%	0.45
	Naive Bayes	76.6%	92.4%	0.50	80%	88.8%	0.55
	k-Nearest Neighbours (IBk, k=3)	60%	101.2%	0.09	56.7%	102.3%	0.09
	Bagging	70%	95.1%	0.17	66.6%	92.7%	-0.06
	Decision trees (J48)	70%	102.4%	0.15	66.6%	106.2%	0.12
	One-step BBN	70%	97.9%	0.39	80%	91.8%	0.56
	Multi-Step BBN	63.3%	106.3%	0.45	-	-	-
Quality	Multinomial Logistic Regression	63.3%	100.4%	0	40%	133.7%	0.04
	Multilayer Perceptron	56.6%	107.2%	0.24	63.3%	104%	0.16
	Support Vector Machine (SMO)	56.6%	104.7%	0.16	63.3%	109.4%	0
	Naive Bayes	50%	114.1%	0.15	50%	116.3%	0.09
	k-Nearest Neighbours (IBk, k=3)	63.3%	99.6%	0.20	60%	101.1%	-0.08
	Bagging	56.6%	101.4%	-0.11	63.3%	101%	0
	Decision trees (J48)	66.6%	94.9%	0.28	56.6%	114.4%	0.07
	One-step BBN	46.6%	119.4%	0.12	53.3%	104%	0.15
	Multi-Step BBN	63.3%	100.4%	0	-	-	-
Risk	Multinomial Logistic Regression	51.7%	119%	0.27	55.1%	109.7%	0.33
	Multilayer Perceptron	51.7%	110.4%	0.27	62%	89.4%	0.44
	Support Vector Machine (SMO)	44.8%	113.6%	0.17	65.5%	93.8%	0.48
	Naive Bayes	48.2%	113.7%	0.22	58.6%	107.5%	0.40
	k-Nearest Neighbours (IBk, k=3)	37.9%	108.9%	0.06	34.5%	104.9%	0.02
	Bagging	55.17%	966.7%	0.32	48.2%	97.6%	0.22
	Decision trees (J48)	37.9%	116.5%	0.07	41.4%	125.5%	0.12
	One-step BBN	48.2%	115.9%	0.22	55.1%	108.8%	0.34
	Multi-Step BBN	44.8%	97.5%	0.20	-	-	-

The main difference between our solution and Dynamic Bayesian Networks (DBNs) is that in DBNs, the structure of the network is fixed over all time intervals (phases in our case), while we allow different network structures to represent different phases. Training DBNs for this task would require knowing the values of the same variables over different time points. In our case, we have a different set of variables to consider at each time point. So, we believe we are a bit more general in the sense of allowing different variables sets for each phase.

General BBNs have been used also in other works. Fenton *et al.* automatically define the following factors for predicting the size of the resulting software, [17] its quality and the necessary effort for developing it: factors influencing prior rates, prior error and productivity rates, constants describing process and project attributes which adjust prior error and productivity error, process and people factors which also adjust error and productivity rates, adjusted error and productivity rates, trade-off component between the quality, functionality and effort. Another example of the use of general BBNs for software engineering is AREL (Architecture Rationale and Element Linkage) that exploits the idea of BBNs to propagate probabilities of tracing change

Copyright: the authors

Table 2. Accuracy of different learning techniques using numeric data.

Model	Learning Technique	Correlation	RRSE
Cost	Linear Regression	0.09	101%
	Multilayer Perceptron	0.74	80%
	Support Vector Machine (SMOreg)	0.62	92%
	k-nearest neighbours (IBk, k=3)	0.40	95%
	Bagging	-0.08	97%
	Regression Trees (M5P)	-0.02	104%
Quality	Linear Regression	0.96	131%
	Multilayer Perceptron	0.56	82%
	Support Vector Machine (SMOreg)	0.98	39%
	k-nearest neighbours (IBk, k=3)	0.53	89%
	Bagging	0.40	92%
	Regression Trees (M5P)	0.92	64%
Risk	Linear Regression	0.45	97%
	Multilayer Perceptron	0.57	94%
	Support Vector Machine (SMOreg)	0.57	94%
	k-nearest neighbours (IBk, k=3)	0.57	75%
	Bagging	0.16	91%
	Regression Trees (M5P)	0.43	88%

impact decisions back from the architectural design of software. [40] The authors claim that it is highly desirable to automatically derive the design of BBNs to be used in the project estimation reasoning.

These approaches, though, do not automatically derive the conditional probability tables (CPTs) that are used in the non-root nodes of the BBN or the *a priori* probability distributions that are fed at the root nodes. Thus, CPTs and *a priori* probability distributions are specified manually. Also, decision variables are all discrete with a rather narrow domain (usually bi-valued such as "yes/no" or "stable/volatile", yet more restricted than the aforementioned cases). While the models shown here are rather small, the idea can be further generalized to nets arbitrarily large, though it shall be stressed that this model consists of building blocks of single nets.

Another interesting example in making predictions with a single BBN is the work of Amasaki *et al.* [3] The authors estimate risk defined as generating software products with poor final quality using BBNs in the development of computer control systems with embedded software. They estimate project quality as the amount of residual faults in the software product. Another way of using BBNs in model estimation consists of relating different BBNs making different predictions among them. Wang *et al.* start by observing that most works use BBNs to make separate predictions and only a few consider the integration of different factors in the form of BBNs. [42] This work makes a novel proposal by modeling the development process as a Directed Acyclic Graph (DAG) of BBNs, which represents the different input and outputs to take place at different stages of the whole life cycle. This approach has two significant advantages: it provides the project managers with a trade-off analysis of all the parameters involved in the software development process; and BBNs can be fed again at the end of every stage so that, first, beliefs are updated and predictions are subsequently refined; secondly, it is possible to automatically compute deviations from the initial plan. This approach is similar to the one we propose here. The main differences are that they focus on quality, effort, schedule and scope, while we focus on cost, quality and risk. Also, they have defined the BBN

Copyright: the authors

models manually, while we use machine learning to derive them. Wang *et al.* generate BBNs using a mixed-initiave process and learn the BBN parameters from data, [43] though their BBNs do not reflect time. Other recent examples use DBNs to predict the effort derived of software defects and also to predict effort, [39,31] but in the context of Web-based projects.

So, BBNs provide a reasonable solution for most of the problems that classical techniques have for project estimations. They solve part of those problems, being able to: naturally cope with uncertainty in the values of variables or the relationships among them; easily integrate previous data from projects estimations, and experts knowledge on relevant attributes, or important causal relationships; gradually adapting to changes in the values of the variables; providing three kinds of belief propagation models (predictive, diagnostic, or mixed) that can be used for performing advanced what-if analysis, simulate changes and evaluate their impact, or perform complex mixed project estimation analysis by mixing top-down and bottom-up approaches; generating better explanations of the results in terms of higher level variables, so this allows users for argumentation-based design and providing design rationale; or reasoning about a mixture of discrete and numeric variables, though reasoning with numeric variables can be more complex. These advantages can be easily inferred from BBN characteristics as explained in Section 3.

However, using BBNs for software estimation still presents some challenges, such as: defining the right estimation metrics and variables; defining the right causal relationships among the variables in the model; and defining the *a priori* probabilities for the input nodes and the CPTs for the non-root nodes. We propose here the use of machine learning to overcome part of these problems. Regarding the net structure, we propose a semi-automatic approach to generate it. The justification is that is is difficult to generate automatically structures that are semantically correct with respect to causal relations. There are recent works that propose algorithms that consider structural constraints provided by the user for expressing where arcs may or may not be included. [10] Compared with our approach, the main advantage of these techniques is efficiency, since constraints are used to prune the space of structures. However, in our context it is not viable for experts to *a priori* define such constraints as the number of variables is high. Instead, we believe our multi-step method is more expert-friendly, since it provides initial structures that are then modified.

Another drawback of current methods based on BBNs is that they are not accompanied by a software tool that helps users to design and use those models. We have developed also such tool, which is integrated in Eclipse. The tool already has some base models that have been created using post-mortem data from GMV and Skysoft companies, but they can be easily adapted to different companies.

7. Conclusions

In this paper we have described an approach to generate BBN-based models semi-automatically, from data about past projects but supervised by humans. The obtained BBN models can be used to estimate project metrics for new projects of the same type as those used for generating and training the BBNs.

For all steps of the proposed process, except for the human supervision part, a machine learning tool providing the adequate algorithms can be used. Specifically we used Weka, an off-the-shelf tool. The main positive conclusions of the generation of BBNs using the multi-step method are:

- The generated BBNs using the multi-step approach include a reasonable set of variables and their causal relationships. The resulting BBNs can be used to estimate the values of variables throughout the project. Whenever new metrics values are known, they replace their predicted values from previous metrics, and are propagated throughout the BBN. That is, the approach avoids the user trying to find causal relationships manually.
- The multi-step approach allowed building the BBNs in an ordered fashion, so that variables that are known in the later stages of the project are located at the deepest depths of the DAG, whereas variables that are known at the beginning of the project become the roots of the DAG tree.

Copyright: the authors

- Weka can be used to generate BBNs. Currently, the approach we have followed includes some manual refinements, but most of these steps could be easily automated. Thus, given a file with data from various projects, the corresponding BBNs can be automatically generated by embedding Weka in a recursive algorithm. Obviously, the steps related to semantically revising the BBNs by humans cannot be automated.
- Generated BBNs can be automatically translated to the input format of the APES tool, so that they can be loaded into APES for further use.

The main drawbacks of the approach for building BBNs using the multi-step method can be summarized as:

- Finding the right parameters to generate the BBNs can be a tedious process. The main parameters that we have tuned are: number of bins for discretizing the variables, B; and maximum number of parents for any given BBN node, P. The combination of these two parameters defines the size of the CPTs in the BBN nodes. In general, their size will be B^{P+1}. So, the size of CPTs grows exponentially with the number of the parents, P.
- Different discretizations give rise to different BBNs both in the BBN structure, given that the attribute selection process depends on the discretization, and in the CPTs. A potentially good solution consists of manually discretizing as many variables as possible. In this case the meaning of the bins would be closer to the correct one, since the user assigns those bins, independently of their length and frequency.
- The quality of the learned knowledge by any machine learning technique (either BBN or any other) strongly depends on the data. In the case of projects estimations, usually there is a lack of data. This is especially true in relation to number of projects and values of variables for some projects (unknown values for some variables). Also, when there is little data, most entries in the CPTs matrices will not correspond to real data, since in general only N cells in those matrices will have a corresponding entry, where N is the number of projects. In those cases the learning techniques will fill the CPTs with uniform distributions of values. This implies that the resulting BBNs will only be good for describing causal relations among variables, rather than being good estimators. Thus, BBNs are good solutions in relation to pure machine learning (or regression) approaches, or pure manual definition: they allow users to provide knowledge at the beginning (by defining a structure of the DBNs) or during the generation process (by redefining the structure, or controlling the machine learning task); and as a complement, they use machine learning for the more tedious and difficult tasks of assigning the probabilities.

We have performed several experiments using data of past projects from GMV and Skysoft. Specifically, following the proposed approach we have built models for estimating cost, quality and risk of new projects. The resulting BBNs allow also users to estimate all other variables implied in the models. Regarding the quality of the obtained BBNs, the main positive conclusions are:

- The generated BBNs using the last scheme include a reasonable set of variables and their causal relations. The resulting BBNs can be used to estimate the values of variables throughout the project. Whenever new metrics values are known, they replace their predicted values from previous metrics, and are propagated throughout the BBN.
- Although the accuracy that we have reached is not very high in some cases, the method proposed has shown its validity to obtain estimation models. Higher accuracy can be found by gathering more data from existing and future projects. These data will contribute to refine the existing BBNs, adapting its CPTs, and finding new causal relationships.

Acknowledgements

This work was developed under a collaborative research project with the European Space Agency (ESA). We would like to thank the great help provided by Yuri Yushtein during the whole development of the project.

Copyright: the authors

References

1. Moataz A. Ahmed, Moshood Omolade Saliu, Jarallah AlGhamdi, Ken Currie, and Austin Tate. Adaptive fuzzy logic-based framework for software development effort prediction. *Information and Software Technology*, 47(1):31–48, 2005.
2. Allan Albrecht. Measuring application development productivity. In *Proceedings of the IBM Applications Development Symposium*, pages 83–92, 1979.
3. Sousuke Amasaki, Yasunari Takagi, Osamu Mizuno, and Tohru Kikuno. Constructing a bayesian belief network for predicting final quality in embedded system development. *IEICE - Transactions on Information and Systems*, E88-D(6):1134–1141, 2005.
4. Stanislav Berlin, Tzvi Raz, Chanan Glezer, and Moshe Zviran. Comparison of estimation methods of cost and duration in IT projects. *Information and Software Technology*, 51(4):738–748, 2009.
5. Barry Boehm. *Software Risk Management*. IEEE Computer Society Press, CA, 1989.
6. Barry Boehm, Chris Abts, and Sunita Chulani. Software development cost estimation approaches: A survey. *Annals of Software Engineering*, 10:177–205, 2000.
7. Barry W. Boehm. *Software Engineering Economics*. Prentice-Hall, 1981.
8. Remco R. Bouckaert. *Bayesian Belief Networks: from Construction to Inference*. PhD thesis, Utrecht, Netherlands, 1995.
9. Remco R. Bouckaert. Bayesian network classifiers in Weka for version 3-5-7. Technical report, Waikato University, 2008.
10. Qiang Ji Cassio P. de Campos. Efficient structure learning of bayesian networks using constraints. *Journal of Machine Learning Research*, 12:663–689, 2011.
11. Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. Smote: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, 16:321–357, 2002.
12. Shyam R. Chidamber and Chris F. Kemerer. A metrics suite for object oriented design. *IEEE Transactions in Software Engineering*, 20:476–493, 1994.
13. Eun Sook Cho, Min Sun Kim, and Soo Dong Kim. Component metrics to measure component quality. In *Proceedings of the Software Engineering Conference (APSEC 2001)*, pages 419–426, 2001.
14. Sunita Chulani. Bayesian analysis of software cost and quality models and software maintenance. In *Proceedings of the IEEE International Conference on Software Maintenance (ICSM'01)*, pages 565–568, 2001.
15. Gregory F. Cooper and Tom Dietterich. A bayesian method for the induction of probabilistic networks from data. pages 309–347, 1992.
16. Taz Daughtrey. *Fundamental Concepts for the Software Quality Engineer*. American Society for Quality, 2001.
17. Norman Fenton, Lukasz Radlinski, and Martin Neil. Improved bayesian networks for software project risk assessment using dynamic discretisation. In *Proceedings of the IFIP Conference Software Engineering Techniques*, pages 139–148, 2006.
18. Tom Gilb. *Software Metrics*. Chartwell-Bratt, 1976.
19. Andrew R. Gray and Stephen G. MacDonell. A comparison of techniques for developing predictive models of software metrics. *Information and Software Technology*, 39(6):425–437, 1997.
20. Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The Weka data mining software: an update. *SIGKDD Explorations*, 11(1), 2009.
21. Khaled Hamdan, Hazem El Khatib, John Moses, and Peter Smith. A software cost ontology system for assisting estimation of software project effort for use with case-based reasoning. In *Proceedings of Innovations in Information Technology*, pages 1–5, 2006.
22. Sallie Henry and Dennins Kafura. Software structure measurements based on information flow. *IEEE Transactions in Software Engineering*, SE-7:510–518, 1976.
23. Giuseppe Jurman and Cesare Furlanello. A comparison of MCC and CEN error measures in multi-class prediction. *PLoS ONE 7(8): e41882*, 2012.
24. Chris F. Kemerer. An empirical validation of software cost estimation models. *Communications of the ACM*, 30(5):416–429, 1987.
25. Ekrem Kocaguneli, Tim Menzies, and Jacky M. Keung. On the value of ensemble effort estimation. *IEEE Transactions on Software Engineering*, (6):1403–1416, 2011.
26. Sotiris Kotsiantis and Dimitris Kanellopoulos. Discretization techniques: A recent survey. *GESTS International Transactions on Computer Science and Engineering*, 32(1):47–58, 2006.
27. Huan Liu, Farhad Hussain, Chew Lim Tan, and Manoranjan Dash. Discretization: An enabling technique. *Data Mining and Knowledge Discovery*, 6(4):393–423, October 2002.
28. Carolyn Mair, Gada Kadoda, Martin Lefley, Keith Phalp, Chris Schofield, Martin Shepperd, and Steve Webster. An investigation of machine learning based prediction systems. *The Journal of Systems and Software*, 53:23–29, 2000.
29. B Matthews. Comparison of the predicted and observed secondary structure of t4 phage lysozyme. *Biochimica et Biophysica Acta*, 400:442–451, 1975.
30. Thomas J. McCabe. A complexity measure. *IEEE Transactions in Software Engineering*, 2(4):308–320,

Copyright: the authors

1976.
31. Emilia Mendes and Nile Mosley. Bayesian network models for web effort prediction: A comparative study. *IEEE Transactions on Software Engineering*, (34):723–737, 2008.
32. Kjetil Molokken and Magne Jorgensen. A review of software surveys on software effort estimation. In *Proceedings of the International Symposium on Empirical Software Engineering*, pages 223–230, 2003.
33. Richard E. Neapolitan. *Learning Bayesian Networks*. Prentice Hall, 2004.
34. Martin Neil Norman E Fenton. Software metrics: Roadmap. In *Proceedings of the Conference on The Future of Software Engineering*, pages 357–370. ACM Press, 2002.
35. Judea Pearl. *Probabilistic Reasoning in Intelligence Systems*. Morgan Kaufmann, 1988.
36. Lawrence H. Putnam. A general empirical solution to the macro software sizing and estimating problem. *IEEE Transactions on Software Engineering*, 4(4):345–361, 1978.
37. Lukasz Radlinski. A survey of bayesian net models for software development effort prediction. *International Journal of Software Engineering and Computing*, 2(2):95–109, 2010.
38. Roy Schmidt, Kalle Lyytinen, Mark Keil, and Paul Cule. Identifying software project risks: An international delphi study. *Journal of Management Information Systems*, 17:5–36, March 2001.
39. Thomas Schulz, Lucas Radlinski, Thomas Gorges, and Wolfgang Rosenstiel. Defect cost flow model - a bayesian network for predicting defect correction effort. In *Proceedings of the International Conference on Predictive Models in Software Engineering*, page Article No. 16, 2010.
40. Antony Tang, Ann Nicholson, Yan Jin, and Jun Han. Using bayesian belief networks for change impact analysis in architecture design. *The Journal of Systems and Software*, 80:127–148, 2007.
41. Thomas Verma and Judea Pearl. An algorithm for deciding if a set of observed independencies has a causal explanation. In *Proc. of the Eighth Conference on Uncertainty in Artificial Intelligence*, pages 323–330, 1992.
42. Hao Wang, Fei Peng, Chao Zhang, and Andrej Pietschker. Software project level estimation model framework based on bayesian belief networks. In *Proceedings of the Sixth International Conference on Quality Software (QSIC'06)*, pages 26–30, 2006.
43. Xiaoxu Wang, Chaoying Wu, and Lin Ma. Software project schedule variance using bayesian network. In *Proceedings of the IEEE International Conference on Advanced Management Science (ICAMS)*, pages 209–218, 2010.

Copyright: the authors