Simulated Annealing for Improving Software Quality Prediction

Salah Bouktif
Département de Génie Informatique, École Polytechnique de Montréal
C.P. 6079, succ. Centre-ville
Montréal (Québec) H3C 3A7
Canada
salah.bouktif@polymtl.ca

Houari Sahraoui
Department of Computer Science
University of Montreal
C.P. 6128 succ. Centre-ville
Montréal (Québec) H3C 3J7
Canada
sahraouh@iro.umontreal.ca

Giuliano Antoniol
Département de Génie Informatique, École Polytechnique de Montréal
C.P. 6079, succ. Centre-ville
Montréal (Québec) H3C 3A7
Canada
antoniol@ieee.org

ABSTRACT

In this paper, we propose an approach for the combination and adaptation of software quality predictive models. Quality models are decomposed into sets of expertise. The approach can be seen as a search for a valuable set of expertise that when combined form a model with an optimal predictive accuracy. Since, in general, there will be several *experts* available and each *expert* will provide his *expertise*, the problem can be reformulated as an optimization and search problem in a large space of solutions.

We present how the general problem of combining quality experts, modeled as Bayesian classifiers, can be tackled via a simulated annealing algorithm customization. The general approach was applied to build an expert predicting object-oriented software stability, a facet of software quality. Our findings demonstrate that, on available data, composed expert predictive accuracy outperforms the best available expert and it compares favorably with the expert build via a customized genetic algorithm.

Categories and Subject Descriptors

D [**Software**]: Miscellaneous; D.2.7 [**Software Engineering**]: Metrics—*Product Metrics, Management*

General Terms

Measurement, Algorithms, Management

Keywords

Simulated annealing, Software quality, predictive models, Bayesian Classifiers, Expertise reuse

1. INTRODUCTION

Assessing and improving software quality are becoming permanent concerns during all the phases of the software life cycle. Software quality is a multidimensional notion often defined and evaluated according to a set of characteristics. Commonly used characteristics are, among others, complexity, maintainability, stability, understandability, portability, etc.

Unfortunately, the majority of these characteristics cannot be quantified a priori to help decision-making in the early phases of the development process. For instance, several years of operation are typically needed to determine if a given software product is maintainable. As a consequence, empirical investigations of measurable internal attributes (size, coupling, cohesion, etc.) and their relationship to quality characteristics are very promising solutions for the assessment of a software product quality [16]. When verified, the relationships between software internal attributes and its quality characteristics are often used as quality predictive models.

To build predictive models or to validate the relationships that they implement, we need representative data samples for which the values of the quality characteristic to predict are already known. For many reasons, in the field of object-oriented (OO) software, such sets of data are rare [12]. As a result, for an organization, it is difficult to select a predictive model that is relevant to its context. Moreover, the models are developed according to specific contexts with many hidden hypotheses. As a quality specialist, one of the frequent and complex request you can face is "I want to evaluate/estimate the characteristic X, what predictive/estimation model is appropriate for this task?"

Our approach borrows its inspiration from the fact that the existing models are the results of a considerable effort. The majority of them embody valuable knowledge and expertise that should be reused. However, the reuse is not straightforward. The embodied expertise must be adapted to the particular domain and context of the target organization. Moreover, we don't have any guarantee that a single model can cover any specific context. In general, a combination of several existing models is likely to be needed to obtain a satisfactory accuracy.

In this paper, we propose an approach to reuse and adapt quality predictive models of software. We view each model as a set of expertise parts. Each part covers a subset of the potential input set for a decision making problem. When we put all the expertise parts of all the models together, the reuse and adaptation can be seen a search for the *best* subset of expertise parts that combined, forms a model with an optimal predictive accuracy. Therefore, the problem of determining the optimal subset can be modeled as an optimization problem requiring a search in a large space of solutions.

Metaheuristics are known to be efficient in finding a near-optimal solutions in such cases. In our project, we experimented two techniques: genetic algorithms (GA), a population-based technique and

GECCO'06, July 8–12, 2006, Seattle, Washington, USA.

simulated annealing (SA), a neighborhood-based technique. In this paper we present an application of our approach using a simulated annealing algorithm. For lack of space, the genetic algorithm is not presented. However, in the evaluation section we will compare the results of the two algorithms.

Although the approach can be applied to different kinds of predictive models, for this project both algorithms were defined for Bayesian classifiers (BC), i.e., probabilistic predictive models. Indeed, BCs, as stated by Fenton in [13], are becoming more and more popular as they provide a good support to decision making. The quality characteristic we consider in this work is the stability of object-oriented applications.

The main contribution and novelty of this paper are as follows:

- we propose a general approach to reuse quality prediction expertise;
- we report our SA customization for BCs combination;
- we present an application of our approach on large OO software stability prediction.

This paper is organized as follows. First, we present our problem statement and we describe our approach in section 2. Since in our approach, a model is seen as a set of expertise parts/chunks, the mapping of a BC model to a set of expertise chunks is described in section 3. Deriving a predictive model for a particular organization is done by selecting progressively the valuable combinations of chunks subsets of the considered models that better fits the context of this organization. Section 4 presents this model derivation approach using simulated annealing (SA) algorithm. An evaluation of our approach using software stability BC models is presented in section 5. Finally, a conclusion is given in section 6.

2. PROBLEM STATEMENT AND APPROACH

In this section we introduce definitions and formalism that will be used throughout the paper. As explained in the introduction, a predictive model of software quality is a relationship between internal and external attributes of a software. It is built/validated empirically using a data sample $D_c = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$ containing n examples or data points, where $\mathbf{x}_i \in \mathbb{R}^d$, is an observation vector of d attributes (software internal metrics) and $y_i \in \mathcal{C}$ is a label (quality characteristic) in a particular case of classification models. $\mathbf{x}_i = (x_{i,1}, \ldots, x_{i,d})$ is the result of measuring d software internal attributes; whenever no confusion arises, to simplify notation, we will interchange $x_{i,j}$ with a_j being a_j the generic value assumed by the j^{th} attribute, realization of $\mathbf{x}_i$ in any observation process. The data set D_c should be a representative sample of a software context and properly "represent" a set of software components, an application domain, a programming style, programmer skills, good and bad practices, etc. We call D_c a *particular context* of an organization and D the set of contexts of all possible organizations. D can be seen as a virtual set of data that representatively describes all the possible types of software.

To build a predictive model for a particular organization using its context D_c, three alternatives can be considered: (1) apply a statistic-based or machine learning algorithm on D_c (2) select the best existing model[1] using D_c (3) reuse and eventually adapt as many as possible existing models using D_c to guide the search process. We believe that the third alternative is more valuable. Indeed, an ideal predictive model is a mixture of two types of knowledge: domain common knowledge and context specific knowledge. By reusing existing models, we reuse the common domain knowledge represented by versatile contexts and by guiding the adaptation via the context specific data, we consider the specific knowledge represented by D_c. By adapting and reusing multiple expertise the goal is to compose an expert outperforming the best existing model (*best expert*); this in turn will play the role of a benchmark for evaluating our proposed solution.

In this paper we consider the problem of reusing N predefined models $f_1, \ldots, f_N$ called *experts* to product a new optimal model on the available context D_c. The first and simplest way is to combine N experts by using a voting based methods. The constructed model f is a normalized weighted sum of the outputs of the existing models. Formally, a composed expert can be written as:

$$f(\mathbf{x}) = \sum_{j=1}^{N} w_j f_j(\mathbf{x}),$$

where $w_j, j = 1, \ldots, N$ are weights that can be positive constants. In more general and sophisticated voting based methods, the weights are functions of the input $w_j(\mathbf{x})$ e.g., in the technique referred to as *mixture of expert* [14]. The weights can be naturally interpreted as degrees of our confidence in the jth expert on our data set D_c. However, the disadvantage of this solution comes from the "black-box" property of the derived models in the sense that there is more than one expert responsible for the output value. To avoid the drawback of this solution, we propose an approach reusing the existing experts to derive new experts having a higher predictive accuracy, without worsening the interpretability of the original experts. Considering this objective our approach consists of three steps summarized in Figure 1:

Step 1 decomposes each expert into chunks of expertise. Each chunk represents the behavior of the expert on a "partition" of the entire input space (i.e., the whole virtual dataset D). In general, a chunk of expertise can be defined as a component of the expert knowledge, which can be represented using a certain technique such as linear regressions, decision trees, Bayesian classifiers, etc. The "partitioning" of the input space depends on the structure of the expert representation. For example, the decomposition of a decision tree leads to expertise chunks in form of rules, thus a "partition" is a decision region in the input space. For linear regression based expert, an expertise chunk is not other than the restriction of the regression function on a predetermined sub-space of inputs, a partition of the entire input space. However, in the case of Bayesian classifier, an expertise chunk is a set of prior probabilities attached to each range (interval) of attribute values (See detail in Section 4).

The first step gives more flexibility when manipulating the experts in the sense that an expert could have chunks of expertise more or less accurate than others. Moreover, the derived model which is a combinations of chunks of expertise will be more interpretable since we know the chunks responsible for the decision.

Step 2 reuses the chunks of expertise coming from different experts in a way to build progressively more accurate combinations of these using D_c to guide the search.

Step 3 modifies some chunks of expertise to obtain more adapted expertise combinations to the context D_C.

This three step process of building an optimal expert can be thought of as a searching problem where the goal is to determine the best set of expertise given D_c. In general, several experts will be available; each expert decomposes into a set of expertise and thus we face a combinatorial problem searching the best combination of expertise in a large search space.

Moreover, expert opinions can be inconsistent i.e., chunks of expertise may conflict over an output, and thus there is a need to

[1]Model predicting the same quality factor

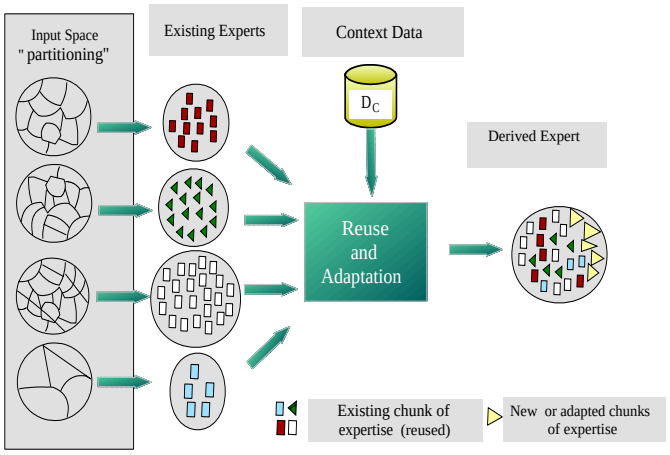

Figure 1: Model derivation process.

further subdivide chunks of expertise increasing the search space complexity. In other words, we are facing a NP-complete decision problem, that naturally gives rise to an associated NP-hard optimization problem seeking for an optimal combinations of expertise in term of predictive accuracy i.e., there is no algorithm running in polynomial time.

3. BAYESIAN CLASSIFIER DESCRIPTION

A Bayesian classifier is a simple classification method, that classifies a d-dimensional observation $\mathbf{x}_i$ by determining its most probable class c computed as:

$$c = \arg\max_{c_k} p(c_k|a_1, \ldots, a_d),$$

where c_k ranges of the set of possible classes $\mathcal{C} = \{c_1, \ldots, c_q\}$ and the observation $\mathbf{x}_i$ is written as generic attribute vector. By using *the rule of Bayes*, the probability $p(c_k|a_1, \ldots, a_d)$ called probability *a posteriori*, is rewritten as:

$$\frac{p(a_1, \ldots, a_d|c_k)}{\sum_{h=1}^{q} p(a_1, \ldots, a_d|c_h)p(c_h)} p(c_k).$$

The expert structure is drastically simplified under assumption that, given a class c_k, all attributes are conditionally independent; under this assumption the following common form of *a posteriori* probability is obtained:

$$p(c_k|a_1, \ldots, a_d) = \frac{\prod_{j=1}^{d} p(a_j|c_k)}{\sum_{h=1}^{q} \prod_{j=1}^{d} p(a_j|c_h)p(c_h)} p(c_k). \quad (1)$$

When the independence assumption is made the classifier is called Naive Bayes. $p(c_k)$ called marginal probability [12] or base probability [10], is the probability that a member of a class c_k will be observed. $p(a_j|c_k)$ called prior conditional probability, is the probability that the j^{th} attribute assumes a particular value a_j given the class c_k. These two prior probabilities determine the structure of a naive BC. They are learned, i.e., estimated, on a training set when building the classifier. As discussed in [9], a naive BC is then a simple structure that has (1) the classification node as the root node, to which is associated a distribution of marginal probabilities and (2) the attribute nodes as leaves, to each of them are associated q distribution of prior conditional probabilities, where q is the number of possible classes. Because of the strong independence assumption, no connections are allowed between nodes in the naive BC. A naive BC treats discrete and continuous attributes in different ways [15]. For each discrete attribute, $p(a_j|c_k)$ is a single real that represents the probability that the j^{th} attribute will assume a particular value a_j when the class is c_k. Continuous attributes are modeled by some continuous distribution over the range of that attribute's value. A common assumption is to consider that within each class, the values of continuous attributes are distributed as a normal (i.e., Gaussian) distribution. This distribution can be represented in term of its mean an its standard deviation. Then we interpret an attribute value a_j as laying within some interval. The attribute domain is divided into intervals I_{jt_j} and $p(I_{jt_j}|c_k)$ will be the prior conditional probability of a value of the j^{th} attribute to be in the interval I_{jt_j} when the class is c_k; $t_j \in \mathbb{N}$ is the rank of the interval in the attribute domain. To classify a new observation $\mathbf{x}_i$ (i.e., $a_1, \ldots, a_d$), a naive BC with continuous attributes apply Bayes theorem to determine the *a posteriori* probability as:

$$p(c_k|I_{1t_1}, \ldots, I_{dt_d}) = \frac{\prod_{j=1}^{d} p(I_{jt_j}|c_k)}{\sum_{h=1}^{q} \prod_{j=1}^{d} p(I_{jt_j}|c_h)p(c_h)} p(c_k). \quad (2)$$

with $a_j \in I_{jt_j}$.

4. ADAPTING SIMULATED ANNEALING ALGORITHM

In order to tailor the SA algorithm for our approach, some elements must be defined. First and foremost, we need to define an adequate representation to encode possible solutions and assure the BC decomposition into chunks of expertise (Section 4.1). Secondly, a generator of random changes in solutions is necessary to perform moves in the neighborhood of the current solution and create new intermediate solutions. This latter element is called neighborhood function and assures the combination and the adaptation (Section 4.2). Finally, to evaluate solutions and to guide search, an objective function have to be defined (Section 4.3).

4.1 Representation of Bayesian Classifier solution

The encoding of BC into chucks of expertise is central to our approach. This operation facilitates the exploration of the search space defined by all the combinations of original and modified chunks of expertise. Consequently it makes easier and efficacious the steps of reusing and adapting the existing BCs.

According to the description of Naive BCs using the continuous attributes given in Section 3, two prior parameters of a BC are candidate to represent a chunk of expertise. The first consists in the marginal probabilities of different classes $p(c_k)$, where $k = 1, \ldots, q$. The second consists in the prior conditional probabilities of the attributes $p(I_{jt_j}|c_k)$. Since prior conditional probabilities are more relevant to express a different structure for a BC, they were chosen to define a chunk of expertise.

To each attribute, say the j^{th} attribute, are associated m_j chunks of expertise. Each chunk can be represented by the $(q+1)-tuple$ made up by an interval I_{jt_j} and an array of q conditional probabilities $(p_{jt_jc_1}, \ldots, p_{jt_jc_q})$.

This form of expertise $(I_{jt_j}, p_{jt_jc_1}, \ldots, p_{jt_jc_q})$, where $p_{jt_jc_k}$ denotes the prior conditional probability $p(I_{jt_j}|c_k)$, makes it possible to preserve the structure of a BC while the search is progressing. Figure 2 shows a structure of a BC using three continuous attributes (metrics COH, LCOMB and STRESS, see Table 3) to classify OO-classes into stable and unstable ones. Each attribute domain is di-

vided into set of intervals. To each interval are associated two prior conditional probabilities. For example, the COH attribute domain ($[L_{COH}, U_{COH}]$=$[0, 1]$) is divided into four intervals. The probability of a COH-value to be in the second interval ($[0.3, 0.65]$)of the COH-domain is equal to 0.37 given the class $stable$ and equal to 0.33 given the class $unstable$. Table 1 shows all the prior probabilities associated to different intervals of the attribute COH. Note

	COH Int.	[0,0.3]	[0.3,0.65]	[0.65,0.9]	[0.9,1]
COH	*stable*	0.08	0.37	0.13	0.42
	unstable	0.4	0.33	0.22	0.05

Table 1: Prior priorities Table of COH.

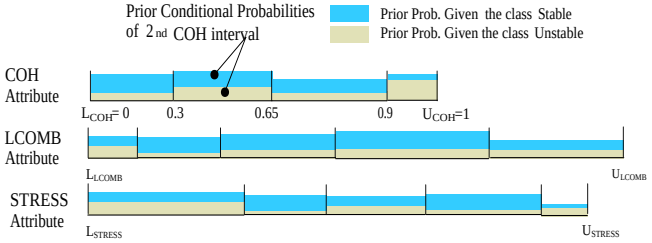

Figure 2: A graphical representation of Naive Bayesian Classifier chunk of expertise.

that for each attribute the lower and upper boundaries can be different from one classifier to an other, however in order to simplify and shorten our SA description, in this paper we assume that each attribute remains defined in same domain for all the BCs.

4.2 Neighborhood function

The neighborhood function also called transition function, is the element of SA algorithm which is responsible for the two principal mechanisms of the our approach, namely, the reuse and the adaptation of the expertise chunks. For any given BC with q classes, each interval in any attribute domain has associated q prior probabilities. Intuitively, a transition is carried out by making changes in the structure of the BC representing the current solution to obtain a nearby *improved* BC. Starting from a BC representation, as a set of expertises (intervals plus probability distributions), a transition consists in making a *reasonable* modification on some intervals and thus changing either probabilities or interval boundaries. Clearly there is a combinatorial number of possible ways to perform such a modification. In addition, the mechanism of combination of our approach is founded on the reuse of the expertise coming from the different existing BC.

Indeed, to implement this mechanism in a technically sound and easy way, we propose two new forms of transitions based on adding or deleting expertise chunks from a BC.

Three properties of a BC must be ensured while adding and deleting expertise chunks: consistency, completeness and distribution. In this paper, for consistency property, we mean that the structure of the BC must be consistent and thus for a given class, i.e., only one probability must be associated to any given interval. By completeness we mean that the structure of the BC must be complete and therefore given a class, any interval must have a prior probability. For the distribution property, we mean that given a BC, one of its attributes and a class the sum of prior probabilities over the attribute domain have to sum up to one. In the following, to summarize our approach to preserve essential BC property, we will make no distinction between the interval and the expertise and in general the term interval will be used to represent both the interval and the associated prior probabilities.

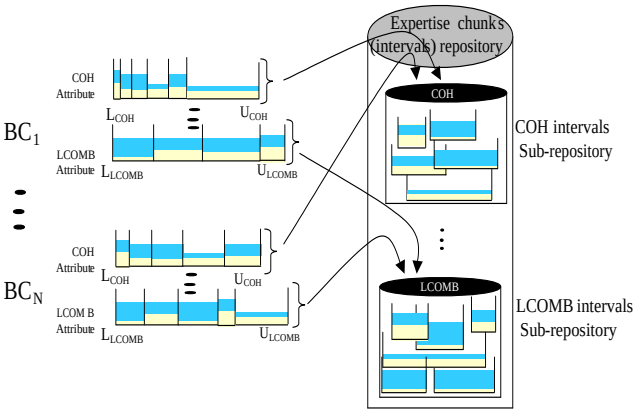

Figure 3: Expertise repository constitution. The interval repository is constituted of a set of sub-repositories. Each sub-repository contains all the interval of a particular attribute.

4.2.1 *Adding an expertise chunk*

To guarantee the completeness of a BC being modified, we proceed to a graft, a *transplantation*, of a new interval (new expertise) in the composition of the corresponding attribute domain. For example, an interval of the attribute COH in the repository is grafted in the COH-attribute domain of the BC, as shown in Figure 4. The grafted interval is randomly selected in a repository of intervals. This action assumes the existence of a repository of expertise chunks obtained by the dissociation of all the intervals composing the attributes of the all existing classifiers. The operation of the expertise repository constitution is summarized in Figure 3. Let I'_{jg} be an interval randomly select in the interval repository where g is its rank in the domain of the attribute j in its original BC. After the grafting action of I'_{jg} in the domain of the attribute j in the current BC, we obtain a new interval composition of the domain of the current[2] attribute j denoted $(I_{j1}, \ldots, I'_{jg'}, \ldots, I_{jm})$, where m is the new number of intervals in the current attribute j and g' is the new rank of the grafted interval in the current attribute j. To ensure consistency, the probabilities *a priori* attached to the grafted interval $p(I'_{jg'}|c_k)$, $k = 1, \ldots, q$ are preserved in order to dominate the original ones in the domain part covered by $I'_{jg'}$. An adjustment of prior probabilities of the remaining original intervals $(I_{j1}, \ldots, I_{j(g'-1)}, I_{j(g'+1)}, \ldots, I_{jm})$ is necessary to ensure the distribution property. These intervals are those non completely covered but possibly overlapped by $I'_{jg'}$. Several choices are possible to adjust these prior probabilities. A simple way, is to distribute the remainder of the probability $(1 - p(I'_{jg'}|c_k))$ on the remaining intervals $(I_{j1}, \ldots, I_{j(g'-1)}, I_{j(g'+1)}, \ldots, I_{jm})$.

A different way is: given a class, the prior probability associated to any remaining interval could be proportional to its length. This choice supposes an uniform distribution of the attribute values and omits their original distribution expressed by the prior probabilities. Another alternative consists in associating to each remaining interval a new prior probability proportional to its original priori probability. This alternative is more appealing because it does not

[2]Attribute j of the current BC being modified

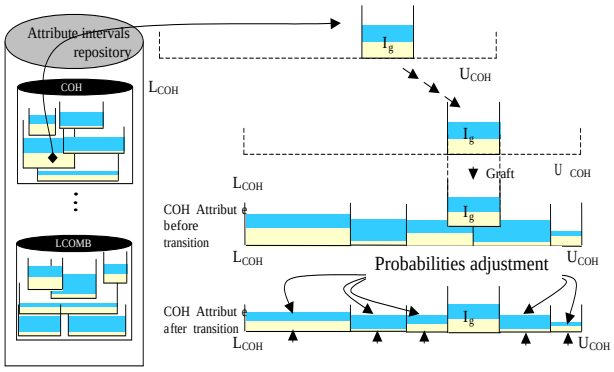

Figure 4: Transition function : expertise grafting.

suppose any arbitrary distribution of the attribute values and takes into account the original expertises attached to the remaining intervals $(I_{j1}, \ldots, I_{j(g'-1)}, I_{j(g'+1)}, \ldots, I_{jm})$. New prior probabilities $p'(I_{jt_j}|c_k)$ with $t_j = 1, \ldots, g'-1, g'+1, \ldots, m$ $(t_j \neq g')$, are updated according to the equation :

$$p'(I_{jt_j}|c_k) = \frac{p(I_{jt_j}|c_k)}{\sum_{t=1, t\neq g'}^{m} p(I_{jt}|c_k)}(1 - p(I'_{jg'}|c_k)), \quad (3)$$

where $k = 1, \ldots, q$. In this equation, given a class c_k we weight the remainder $1 - p(I'_{jg'}|c_k)$ by the proportion of the original prior probability $p(I_{jt_j}|c_k)$. The operation of grafting an expertise chunk in an attribute domain of a BC is illustrated in Figure 4

4.2.2 *Deleting an expertise chunk*

This way of performing a transition consists in removing an interval I_{js} chosen randomly in the domain of the attribute j in the BC structure $(I_{j1}, \ldots, I_{js}, \ldots, I_{jm})$. In order to ensure the completeness, one or two of the neighbor intervals ($I_{j(s-1)}$ and $I_{j(s+1)}$) of I_{js} are widened to fill the gap caused by the suppression of I_{js}. The distribution property is then ensured in the same way as when grating an interval. Given a class c_k, the prior probability of each remaining interval $p(I_{jt_j}|c_k)$ is weighted by the inverse of the distribution remainder $(1 - p(I_{js}|c_k))$. The new prior probabilities take the form of :

$$p'(I_{jt_j}|c_k) = p(I_{jt_j}|c_k)\frac{1}{1 - p(I_{js}|c_k)}, \quad (4)$$

where, $t_j = 1, \ldots, m-1$ is the rank of an interval in the new attribute composition and s is the rank of the deleted interval in the original attribute composition.

In the equation 4, the probabilities are increased proportionally to their original values. In other words the existing chunks of expertise are reused to create new ones which justifies our way of probability adjustment.

4.3 Objective function

The main goal of our approach is to derive an expert that has the higher predictive accuracy. In other words the mission of our SA algorithm is to maximize the predictive accuracy of the derived BC f. Our approach can be seen as a learning process where the dataset D_c representing the context is divided the training set of an organization. The same set can be seen as evaluation data set used for computing the predictive accuracy of the classifier proposed by the SA process.

This predictive accuracy (objective function) can be measured in different ways (see [2]). An intuitive measure of it is the *correctness function*

$$C(f) = \frac{\sum_{i=1}^{q} n_{ii}}{\sum_{i=1}^{q}\sum_{j=1}^{q} n_{ij}},$$

where n_{ij} is the number of cases in the evaluation data set with real label c_i classified as c_j (Table 2). Note that for a BC, the class label c_i of a given case is the label that has the highest posterior probability (see equation 2).

		Predicted label			
		c_1	c_2	...	c_q
real label	c_1	n_{11}	n_{12}	...	n_{1q}
	c_2	n_{21}	n_{22}	...	n_{2q}
	⋮	⋮	⋮	⋱	⋮
	c_q	n_{q1}	n_{q2}	...	n_{qq}

Table 2: The confusion matrix of a decision function f. n_{ij} is the number of cases in the evaluation data set with real label c_i classified as c_j.

Software quality prediction data is often *unbalanced*, that is, software components tend to have one label with a much higher probability than other labels. For example, in our experiments we had many more stable than unstable classes. On an unbalanced data set, low training error can be achieved by the constant classifier function f_{const} that assigns the majority label to every input vector. By using the training error for measuring the objective function, we found that the SA process tended to "neglect" unstable classes. To give more weight to data points with minority labels, we decided to use Youden's *J-index* [19] defined as

$$J(f) = \frac{1}{q}\sum_{i=1}^{q}\frac{n_{ii}}{\sum_{j=1}^{q} n_{ij}}.$$

Intuitively, $J(f)$ is the average correctness per label. If we have the same number of points for each label, then $J(f) = C(f)$. However, if the data set is unbalanced, $J(f)$ gives more relative weight to data points with rare labels. In statistical terms, $J(f)$ measures the correctness assuming that the a priori probability of each label is the same. Both a constant classifier f_{const} and a guessing classifier f_{guess} (that assigns random, uniformly distributed labels to input vectors) would have a J-index close to 0.5, while a perfect classifier would have $J(f) = 1$. For an unbalanced training set, $C(f_{\text{guess}}) \simeq 0.5$ but $C(f_{\text{const}})$ can be close to 1.

5. EVALUATION

In order to evaluate our approach, two elements are provided as inputs to the SA algorithm. The first element is a set of existing experts predicting the same quality factor. The second is a representative sample of data collected to describe a number of software components and to reflect the specificities of the software development activity in a particular organization (context data).

To this aim, we created a "semi-real" environment in which the "assumed existing" experts were trained on independent data sets collected from different software systems. Accuracy was evaluated via the prediction of the stability of Sun Microsystems JAVA developer APIs.

BCs were constructed to classify software component into *stable* and *unstable* (see Section 5.1 and the context data set was extracted from a the standard JAVA APIs (see Section 5.2).

5.1 OO Software stability predictive models

During its operation time, software undergoes various changes triggered by error detection, new requirements or environment changes. As an aftereffect, the behavior of the software gradually deteriorates as modifications increase. Consequently, there is a consensus that the software that is intended to last must remain stable. In other words, it should not need major and frequent changes in spite the emergence of new requirements. To reach this goal, a predictive model is required in two cases. To apply the model expertise during the design phase, when building the internal properties into the product. Later on, in the field, to decide after a certain number of versions, whether a major refactoring is necessary to reduce the implementation cost of future requirements.

In our study, we are interested in the stability of OO software at the class level. The key assumption is that a class is stable whenever its interface remains valid (usable) between versions. Let cl be a class. $I(cl_i)$ is the interface of cl in version i (public and protected, local and inherited methods). The level of stability of cl can be measured by comparing $I(cl_i)$ to $I(cl_{i+1})$ (following version). It represents the percentage of $I(cl_i)$ that is included in $I(cl_{i+1})$[3]. Formally

$$SL(cl_i \longrightarrow cl_{i+1}) = \frac{\#(I(cl_{i+1}) \cap I(cl_i))}{\#I(cl_i)}. \quad (5)$$

If $SL(cl_i \longrightarrow cl_{i+1}) = 1$ the class cl_i is stable otherwise is unstable.

With respect to the prediction, our hypothesis is that the stability of a class interface depends on the design (structure) of the class and the stress induced by changes like the implementation of new requirements between the two versions.

The expert will take the form of a function f that takes as input a set of structural metrics $(m_1(cl_i), \ldots, m_d(cl_i))$ and an estimation of the stress $St(cl_i \longrightarrow cl_{i+1})$ and produces as output a binary estimation of the stability $SL(cl) = SL(cl_i \longrightarrow cl_{i+1})$. We assign 1 for stable and -1 for unstable. Formally

$$SL(cl_i \longrightarrow cl_{i+1}) = f(m_1(cl_i), \ldots, m_d(cl_i), St(c_i \longrightarrow cl_{i+1}))$$

The stress $St(cl_i \longrightarrow cl_{i+1})$ represents the estimated percentage of added methods in cl between the two versions. Formally,

$$St(cl_i \longrightarrow cl_{i+1}) = \frac{\#(I(cl_{i+1}) - I(cl_i))}{\#I(cl_{i+1})}.$$

To build the experts (that simulate the existing models), we use the stress plus 18 structural metrics that belong to one of the four categories of coupling, cohesion, inheritance, and complexity, summarized in Table 3. The detailed definitions of these metrics and the predictive models where they were used can be found in [7, 1, 6, 5, 17, 11, 20].

The metrics were extracted from 11 OO systems listed in Table 4. These systems were used to "create" experts to simulate the existing probabilistic models in the following way: First, by using all the combination of the metric categories (of one, two, three and four), we formed 15 subsets of input metrics, and created $15 \times 11 = 165$ data sets. One classifier is trained on each data set by using the machine learning algorithm RoC (the Robust Bayesian Classifier, Version 1.0 of the Bayesian Knowledge Discovery project)[18]. Then, we obtained 165 BCs, among which we retained 40 BCs by eliminating classifiers with training error more than 15%.

[3]We consider a deprecated method as if it is removed.

Name	Description
Cohesion metrics	
LCOM	lack of cohesion methods
COH	cohesion
COM	cohesion metric
COMI	cohesion metric inverse
Coupling metrics	
OCMAIC	other class method attribute import coupling
OCMAEC	other class method attribute export coupling
CUB	number of classes used by a class
Inheritance metrics	
NOC	number of children
NOP	number of parents
DIT	depth of inheritance
MDS	message domain size
CHM	class hierarchy metric
Size complexity metrics	
NOM	number of methods
WMC	weighted methods per class
WMCLOC	LOC weighted methods per class
MCC	McCabe's complexity weighted meth. per cl.
NPPM	number of public and protected meth. in a cl.
NPA	number of public attributes
STRESS	Stress estimation

Table 3: The 19 software metrics used as attributes in the experiments.

System	Number of (major) versions	Number of classes
Bean browser	6(4)	388–392
Ejbvoyager	8(3)	71–78
Free	9(6)	46–93
Javamapper	2(2)	18–19
Jchempaint	2(2)	84
Jedit	2(2)	464–468
Jetty	6(3)	229–285
Jigsaw	4(3)	846–958
Jlex	4(2)	20–23
Lmjs	2(2)	106
Voji	4(4)	16–39

Table 4: The software systems used to construct the "assumed existing" experts(BCs).

5.2 Context data set

We evaluated our approach by predicting the stability of standard-JAVA APIs classes[4]. We recall that this data set is needed to guide the progression of the search process toward a near optimal solution. It describes some 2920 classes JAVA reflecting the "past experience" of Sun Microsystems through four versions of JDK ; JDK1.0(187 classes), JDK1.1(587 classes), JDK1.2(2163 classes) and JDK1.3(2737 classes). For each class of the first three versions, 18 metric-values, an estimate of the stress and an evaluation of stability were calculated. The major evolutions of JAVA, were thus pointed out by the three version-transitions JDK1.0 to JDK1.1, JDK1.1 to JDK1.2 and JDK1.2 to JDK1.3. They were particularly useful for calculating the stress estimation and stability of each class as shown in the previous Section. In Table 5 a brief description of the three JAVA transitions in given, indicating the number of stable and unstable classes according to the equation 5.

[4]See http://java.sun.com.

JDK Transition	Number of Stable classes)	Number of Unstable classes
jdk1.0.2 to 1.1.6	147	42
jdk1.1.6 to 1.2.004	366	217
jdk1.2.004 to 1.3.0	1982	180

Table 5: Summary of the JAVA context of Sun Microsystems

.

5.3 Simulated annealing setting

The major difficulty of implementing the SA algorithm is related to the definition of the search space, and the way in which the solution will be modified. In other words some influencing parameters should be specified namely, Ts_0; the initial temperature, the annealing strategy, Nsa; the number of iterations before decreasing the temperature (shortly, the number of loops) and the criterion of move (transition) acceptance. Several tests are usually carried out to tune parameter values of a metaheuristic algorithm. In our case, these tests show that the behavior of the tailored SA as expected has a dependence on the annealing strategy, on the criterion of move acceptation, on the number of loops. However the SA algorithm achieve higher accuracies of the derived BC above certain thresholds of some parameters. In particular, the quality of the derived BC does not seem to be affected by the initial temperature when this later is sufficiently high. For this reason it was fixed at $10\Delta J$, where ΔJ is the maximum variation of the predictive accuracy of the existing classifiers. Cooling, i.e., *annealing strategy*, was done via the geometric method $Tsa(i) = \alpha Tsa(i-1)$, $0 < \alpha < 1$ (i is a counter of iterations) with $\alpha = 0.97$. The number of loops in the experiment was fixed at $Nsa = 600$. The probability of accepting worsening moves is computed via the Glauber's acceptance criterion [8] given by the following expression: acceptance is

$$p = \frac{exp\frac{\Delta J(f)}{Tsa}}{1 + \frac{\Delta J(f)}{Tsa}}$$

where Tsa is the current pseudo temperature. This Probability is close to 50% at the beginning of the algorithm and becomes close to 0% at end of the algorithm.

5.4 Hypotheses

With the above detailed setting and the tuned parameter the resulting tailored SA algorithm was run to verify four hypotheses.

1. To test if the derived BC has a higher predictive accuracy than the best existing BC.
2. To verify whether the reuse and adaptation of BCs via SA produce BCs with accuracy comparable to those given by GAs.
3. To test if the predictive accuracy of the resulting expert (BC) is proportional to the number of reused existing experts.
4. To verify if an expert with low predictive accuracy, in isolation, can contain *good* (accurate and reusable) chunks of expertise.

5.5 Results

The predictive accuracy was evaluated with the J-index of Youden (See Section 4.3. The J-index of the resulting BCs was estimated using 10-fold cross validation. In this technique, the data set is randomly split into 10 subsets of equal size (292 points in our case). The progression of the SA algorithm, to derive a new BC, is guided by the union of 9 subsets. In other words, a new BC is trained on the union of 9 subsets, and tested on the remaining subset. The process is repeated for all 10 possible combinations. Mean and standard deviation values are computed for J-index on both the training and the test samples. Table 6 summarizes results of BC derivation. We used two benchmarks to evaluate our SA result. We compared our derived classifier, to the best expert f_{Best} simply selected among the existing BCs, to the classifier f_{GA} resulting from the implementation of our approach using Genetic Algorithm (GA). The table also reports the average time required to produce a final BC.

		J-index $J(f)$	CPU Time
Training	f_{best}	63.15(0.54)	-
	f_{GA}	78.56(1.32)	12
	f_{SA}	77.62(1.14)	10
Test	f_{best}	63.10(4.67)	-
	f_{GA}	77.63(2.95)	-
	f_{SA}	77.15(3.37)	-

Table 6: Experimental results. The mean(standard deviation) percentage values of the J-index and CPU time.

The obtained results show a considerable improvement in the predictive accuracy. Compared to the best expert, the resulting BC f_{SA} has gained (11.43%) of a predictive accuracy, which is significantly bigger than two standard deviations respectively of f_{Best} (4.67) and of f_{SA} (3.37). The null hypothesis H_0, assuming that no differences exist between f_{Best} and f_{SA}, is rejected by a statistical test with very strong evidence (1%).

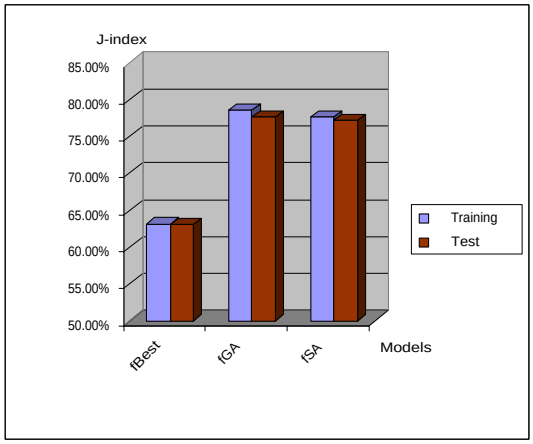

Figure 5: Experimental results: Predictive accuracy improvement with SA.

The analysis of the standard deviation (see Table 6) also shows that the dispersion of the predictive accuracies (J-index) around the average is relatively narrow for the SA algorithm. This result proves that the models produced by SA are stable in spite of its indeterministic-character and that the estimate of the value of the predictive accuracy is more precise. In addition, the small difference between the training and test results indicates that there is not visible overfitting of the deriver BC. SA results are very close to those of GA (f_{GA}, see Table 6). They are so similar that we have no visible reason to prefer one on the other. The results of the predictive accuracy of $J(f_{SA})$, $J(f_{Best})$and $J(f_{GA})$ are summarized and easily seen in Figure 5.

Finally, we note that the average computational time for the execution of our SA algorithm is indeed reasonable for such complex problem (10 minutes for 19 metrics and 2920 data points as context dataset size).

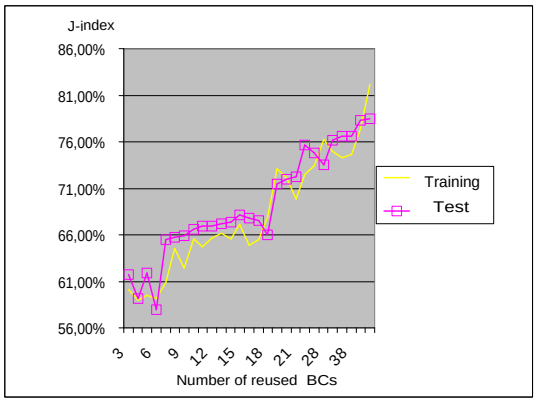

Figure 6: Predictive accuracy evolution and number of existing experts(BCs).

In order to verify the third and the fourth hypotheses, we execute our SA algorithm with the reuse of different number of existing models. At each time, we increase the number of expert by involving a new existing one. As shown in the Figure 5, the resulting expert is increasingly accurate as the number of reused models increases. Moreover, the Spearman's correlation coefficient confirms the direct proportionality between the number of existing BCs taken as inputs and the predictive accuracy of the resulting BC (Correlation. 0.86). In the same Figure 5, the classifier number 19 having 50.72% as J-index, which is a low predictive accuracy had participated in several runs in significant increase (by 6%) of the resulting BC accuracy. This phenomenon implies that a good chunks of expertise can be hidden in inaccurate models.

6. CONCLUSION

The results show that the final resulting model is clearly more accurate than the initial best model. Using SA derivation strategies, good results are obtained for Bayesian classifier case. Both SA and GA are suitable for our combinatorial problem. More precisely, the resulting classifiers are accurate and stable. With regard to the CPU time, the derivation process takes few minutes, announcing that SA is slightly less time consuming than GA for our particular problem. Our approach of reusing and adapting existing model performs well with probabilistic models. Adding theses results to those of our previous work on GA and tabu search using decision trees [4, 3, 2], we are about to say that our approach is at the same time combining-method independent and adaptable for different types of predictive models. This conclusion requires more investigation and experiments. Although our approach yield good improvement over the initial model on a particular context, we strongly believe that if we use more numerous and real probabilistic models on cleaner and less ambiguous data, the improvement will be more significant.

7. ACKNOWLEDGMENTS

This research was partially supported by the Natural Sciences and Engineering Research Council of Canada (Research Chair in Software Evolution #950-202658).

8. REFERENCES

[1] F. Abreu. Metrics for object-oriented environment. In *Proceedings of the Third International Conference on Software Quality Lake Tahoe Nevada*, pages 55–65, 1993.

[2] S. Bouktif. *Improving software Quality prediction by combining and adapting predictive models*. PhD thesis, Montreal University, 2005.

[3] S. Bouktif, D. Azar, S. Sahraoui, B. Kgl, and D. Precup. Improving rule set based software quality prediction: A genetic algorithm-based approach. *Journal of Object Technology*, 3(4):227–241, 2004.

[4] S. Bouktif, B. Kégl, and S. Sahraoui. Combining software quality predictive models: An evolutionary approach. In *Proceeding of the International Conference on Software Maintenance*, pages 385–392, 2002.

[5] L. Briand and J. Wüst. Empirical studies of quality models in object-oriented systems. In M. Zelkowitz, editor, *Advances in Computers*. Academic Press, 2002.

[6] L. Briand, J. Wüst, J. Daly, and V. Porter. Exploring the relationships between design measures and software quality in object oriented systems. *Journal of Systems and Software*, 51:245–273, 2000.

[7] S. Chidamber and C. Kemerer. A metrics suite for object oriented design. *IEEE Transactions of Software Engineering*, 20(6):476–493, 1994.

[8] W. Dolan, P. Cummings, and M. Le-Van. Algorithmic efficiency of simulated annealing for heat exchanger network design. In *Proceedings of the computers in Chemical Engineering conference*, pages 1039–1050, 1990.

[9] R. Duda and P. Hart. *Pattern classification and scene analysis*. John Wiley and Sons, 1973.

[10] C. Elkan. Naive bayesian learning. Technical report, Department of Computer Science Harvard University, 1997.

[11] M. C. Feathers. Stability through change. In *Proceedings of the Accomplishing Software Stability Workshop, OOPSLA 99 Denver, CO*, 1999.

[12] N. Fenton and M. Neil. A critique of software defect prediction models. *IEEE Transactions on Software Engineering*, 25(5):675–689, 1999.

[13] N. Fenton and N. Ohlsson. Quantitative analysis of faults and failures in a complex sofware system. *IEEE Transactions on Software Engineering*, 26(8):797–814, 2000.

[14] R. Jacobs, M. Jordan, S. Nowlan, and G. Hinton. Adaptive mixtures of local experts. *Neural Computation*, 3(1):79–87, 1991.

[15] G. H. John and P. Langley. Estimating continuous distributions in bayesian classifiers. In *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, pages 338–345, 1995.

[16] T. Khoshgoftaar and J. Munson. The lines of code metric as a predictor of program faults: A critical analysis. In *Proceedings of Computer Software and Applications Conference*, pages 408–413, 1990.

[17] R. Martin. Stability. *C++ Report*, 9(2), 1997.

[18] R. Ramoni and P. Sebastiani. Robust bayesian classification. Technical report, Knowledge Media Institute, the Open University, 1999.

[19] W. J. Youden. How to evaluate accuracy. *Materials Research and Standards, ASTM*, 1961.

[20] H. Zuse. *A Framework of Software Measurement*. Walter de Gruyter, 1998.