Estimation of Distribution Algorithms for Testing Object Oriented Software

Ramón Sagarna, Andrea Arcuri and Xin Yao, *Fellow, IEEE*

***Abstract*— One of the main tasks software testing involves is the generation of the test cases to be used during the test. Due to its expensive cost, the automation of this task has become one of the key issues in the area. While most of the work on test data generation has concentrated on procedural software, little attention has been paid to object oriented programs, even so they are a usual practice nowadays.**

We present an approach based on Estimation of Distribution Algorithms (EDAs) for dealing with the test data generation of a particular type of objects, that is, containers. This is the first time that an EDA has been applied to testing object oriented software. In addition to automated test data generation, the EDA approach also offers the potential of modelling the fitness landscape defined by the testing problem and thus could provide some insight into the problem. Firstly, we show results from empirical evaluations and comment on some appealing properties of EDAs in this context. Next, a framework is discussed in order to deal with the generation of efficient tests for the container classes. Preliminary results are provided as well.

I. Introduction

Considering the crucial role software plays nowadays, quality assurance becomes a main issue for the industry in the field. A major way of attaining quality is testing. This element from the software's life cycle is the primary way used in practice to verify the correctness of software. In fact, 50% of the project resources are usually committed to this phase [1]. Among the problems related to testing, the generation of the input cases to be applied to the program under test is especially relevant. Exhaustive testing is generally prohibitive due to the huge size of the input domain, so tests are designed with the purpose of addressing particular aspects of the software system. This makes the generation of test inputs a non-trivial task, as they must conform to the test type and its requirements. Moreover, owing to the fact that in most organizations this generation is performed manually, the automatic generation of test cases is desirable and has turned into one of the main working lines in the area.

A common strategy for tackling this task consists of creating test inputs that fulfill an adequacy criterion based on the program structure. That is, adequacy criteria are defined by the entities revealed by the program source code. For example, entities such as the branches the flow of control can take from a conditional statement define the *branch coverage* criterion, i.e. every program branch must be exercised. In order to know the level of completion attained by a set of test cases, a coverage measure provides the number of exercised structural entities.

Additionally, tests may be applied at different levels, ranging from *unit* testing to *acceptance* testing. A unit refers to the minimal software module that can be tested, e.g. a code function. Acceptance testing, however, alludes to the user validation, e.g. by running the system under simulated real-world operating conditions. Typically, testing proceeds from unit to acceptance level, since the cost of correcting a fault grows superlinear in this direction [2]. A further reason is that a significant amount of the failures (up to 65% [1]) may be detected with unit testing.

Real-world demands and modern development tools are dramatically increasing programmers productivity and, in consequence, software complexity. Therefore, there is a need for advanced test strategies in order to be able to face such a challenge. More precisely, object oriented programming is a usual paradigm for software development nowadays. However, while most of the work on test data generation has concentrated on procedural software, relatively little attention has been paid to object oriented programs.

In the last few years, a number of approaches under the name of Search Based Software Test Data Generation (SBSTDG) have been developed, offering interesting results [3]. SBSTDG tackles the test case generation as a search for the appropriate inputs by formulating an optimization problem. This problem is then solved using metaheuristic search techniques.

The present work tackles test data generation for object oriented software by means of Estimation of Distribution Algorithms (EDAs) [4], [5]. We focus on a common type of object class named *container*. A container class allows objects to be treated as a collection, that is, it encodes a memory structure and the possible actions over it. In particular, we deal with branch coverage for unit testing of such a class type, and we choose Java as the programming language. The aim is then generating tests that cover every branch of every method in the container class. In order to achieve this, the test data generation is posed as an optimization problem whose solution is sought through an EDA. EDAs constitute a family of evolutionary algorithms which induce, at each step of the search, a probability distribution from the selected individuals. To the best of our knowledge, this is the first application of EDAs to object oriented testing. In addition, we believe the induced probability distribution could be a potential source of knowledge about the problem at hand.

Ramón Sagarna, Andrea Arcuri and Xin Yao are with The Centre of Excellence for Research in Computational Intelligence and Applications (CERCIA), School of Computer Science, University of Birmingham, Edgbaston, Birmingham B15 2TT, UK; email: {R.Sagarna,A.Arcuri,X.Yao}@cs.bham.ac.uk

An appealing additional goal of the test case generation for object oriented software is to obtain the most efficient test. A *natural* formulation for this problem implies variable length solutions, which is a fairly unexplored topic in EDAs. We propose a preliminary EDA based framework for overcoming this handicap in the second part of the work.

The remaining sections are arranged as follows. Next, the test data generation problem for object oriented software is outlined. After a brief introduction to EDAs, the approach developed for solving the problem is described and studied through experiments. We continue with the description and evaluation of the EDA based framework. Once related work on the field has been discussed, we finish with conclusions and some ideas for future work.

II. Object Oriented Unit Test Data Generation

In the context of object oriented software, an object is the basic building block of the programs. In contrast to the traditional view in which a program may be seen as a list of instructions or functions, objects act on each other. In other words, each object changes its internal state during program execution. A class is a programming language construct defining the abstract characteristics of an object. More precisely, a class defines the attributes of an object together with its related set of *methods* (functions) or actions. An object becomes then an instance of a class. The values taken by the object on its attributes specify the current state, while the methods allow for the object moving between different states. A container is a specific class type designed to manage any arbitrary data structure, e.g. arrays, lists or trees.

In the present work, the test data generation aims at obtaining a test fulfilling branch coverage for the units of a container class. This involves covering every branch of every method in the class. However, certain parts of the source code in a method may only be exercised when an object is in a particular state, which is reached through a sequence of functions calls. Hence, the problem consists of finding a sequence of function calls (FCs) such that full branch coverage is attained. The situation where a test case consists of a sequence of inputs was studied in a previous work in the context of Search Based Software Test Data Generation [6]; the authors observed a significant increase in the difficulty of the process under these circumstances. The order of the FCs in a sequence is important, since a FC may change the internal state of the object on which it is called, and this might influence the behaviour of the FCs that will be called later on the same object.

A FC can be seen as a triple:

$$< object\_reference, function\_name, input\_list >$$

It is straightforward to map a FC in a command statement. For example:

```
ref.f1(input[0],...,input[n]);
```

That is, given an $object\_reference$ named `ref`, the function `f1` is called with $input\_list$ taking values `input[0]`, ..., `input[n]`. For testing purposes, given a sequence S, only one instance of the class under test is created. All the FCs in S are invoked on the same instance. Therefore, in practice, we could simplify the notation above and represent a FC as a pair:

$$< function\_name, input\_list >$$

We define the length of S as the number of its FCs.

Each FC has its own input parameters ($input\_list$), with the number of parameters depending on the function. We only consider two common types of input parameters: integer values that represent *indices* (locations in the container) and object references for the *elements* that a container can store or use as *keys* (used in hash tables, for instance). Elements can be references to any type of object, e.g. network sockets. Anyhow, they can be ordered and enumerated according to a *natural order*. Since the actual values of the elements are usually not important (only their relative order is relevant [7]), we restrict to references to objects of the class `Integer`. An exception to this is the `null` value, which we treat as well.

III. Estimation of Distribution Algorithms

The term Estimation of Distribution Algorithm (EDA) [4], [8], [9] alludes to a family of evolutionary algorithms which represents an alternative to the classical optimization methods in the area. Algorithmically, a Genetic Algorithm and an EDA only differ in the procedure to generate new individuals. Instead of using the typical breeding operators, EDAs perform this task by sampling a probability distribution previously built from the set of selected individuals. Indeed, this distribution is responsible for one of the main characteristics of these algorithms, that is, the explicit description of the relationships between the problem variables.

A. Abstract EDA

In order to simplify the discussion, only discrete domains are considered in the following. For a more extended description, including continuous domains, the reader is referred to [4].

Given an n-dimensional random variable $\boldsymbol{X} = (X_1, X_2, ..., X_n)$ and a possible instantiation $\boldsymbol{x} = (x_1, x_2, ..., x_n)$, the joint probability distribution of $\boldsymbol{X}$ will be denoted by $p(\boldsymbol{x}) = p(\boldsymbol{X} = \boldsymbol{x})$. In the case of two unidimensional random variables X_i, X_j and their respective possible values x_i, x_j, the conditional probability of X_i given $X_j = x_j$ will be represented as $p(x_i|x_j) = p(X_i = x_i|X_j = x_j)$. In the context of evolutionary algorithms, an individual of length n can be considered an instantiation $\boldsymbol{x} = (x_1, x_2, ..., x_n)$ of $\boldsymbol{X} = (X_1, X_2, ..., X_n)$. Let the population of the l-th generation be D_l. The individuals selected D_l^{Se} constitute a dataset of N cases of $\boldsymbol{X} = (X_1, X_2, ..., X_n)$. EDAs estimate $p(\boldsymbol{x})$ from D_l^{Se}, therefore, the joint probability distribution of the l-th generation will be represented by $p_l(\boldsymbol{x}) = p(\boldsymbol{x}|D_{l-1}^{Se})$. New individuals are then obtained sampling $p_l(\boldsymbol{x})$. A pseudocode for the abstract EDA is presented in Figure 1.

$D_0 \leftarrow$ Generate M initial individuals randomly
Repeat for $l = 1, 2, ...,$until stopping criterion is met
  $D_{l-1}^{Se} \leftarrow$ Select $N \leq M$ individuals from D_{l-1}
  $p_l(\boldsymbol{x}) = p(\boldsymbol{x}|D_{l-1}^{Se}) \leftarrow$ Estimate a probability distribution from D_{l-1}^{Se}
  $D_l \leftarrow$ Sample M new individuals from $p_l(\boldsymbol{x})$

Fig. 1. Abstract EDA pseudocode.

The key point of EDAs is how the probability distribution is estimated at each generation. The computation of all the parameters of $p_l(\boldsymbol{x})$ is unviable since they are, at least, $2^n - 1$ (the case of binary variables). Thus, it is factorized according to a probability model that, in some cases, limits the possible dependencies among the variables $X_1, X_2, ..., X_n$. This leads to several approximations assuming different levels of complexity in their models. The alternatives range from those where the variables are mutually independent to those with no restrictions on variable interdependencies. The most restrictive models avoid the induction of probability distributions with dependencies. However, they allow for a fast and easy estimation that may convert them into a suitable possibility to solve a problem. On the other hand, the least restrictive models are able to show all the dependencies between the variables even though their computational cost is expensive and, in some cases, can result in an impractical choice.

B. EDA instances

The problem of model induction has been tackled by separate scientific fields, such as statistical physics or probabilistic reasoning. EDAs benefit from this knowledge through interdisciplinary research. Taking probabilistic model complexity into account, they may be classified as univariate, bivariate and multivariate.

1) Univariate EDAs: These EDAs assume that the n-dimensional joint probability distribution is decomposed as a product of n univariate probability distributions, that is:

$$p_l(\boldsymbol{x}) = \prod_{i=1}^{n} p_l(x_i).$$

For example, the Univariate Marginal Distribution Algorithm [10] is an instance belonging to this category, which estimates $p_l(x_i)$ as the relative frequencies of x_i in dataset D_{l-1}^{Se}. That is:

$$p_l(x_i) = \frac{\sum_{j=1}^{N} \delta_j(X_i = x_i | D_{l-1}^{Se})}{N}$$

where

$$\delta_j(X_i = x_i | D_{l-1}^{Se}) = \begin{cases} 1 & \text{if } X_i = x_i \text{ in the } j\text{-th case of } D_{l-1}^{Se} \\ 0 & \text{otherwise.} \end{cases}$$

2) Bivariate EDAs: In this category, second order statistics are used in order to estimate the joint probability distribution. Thus, apart from the probability values, a structure that reflects the dependencies must be learnt. The factorization carried out by the models from this category can be expressed as follows:

$$p_l(\boldsymbol{x}) = \prod_{i=1}^{n} p_l(x_i | x_{j(i)})$$

where $X_{j(i)}$ is the variable, if any, on which X_i depends.

TREE [4] is a bivariate EDA which refers to an adaptation of the Combining Optimizers with Mutual Information Trees (COMIT) algorithm [11]. In COMIT, $p_l(\boldsymbol{x})$ is estimated through the Maximum Weight Spanning Tree algorithm described in [12]. In this last work, the objective of the authors was to find the first order dependence probability distribution $p^t(\boldsymbol{x}) = \prod_{i=1}^{n} p(x_i | x_{j(i)})$ which best approximates $p(\boldsymbol{x})$. As a discrepancy measure, the Kullback-Leibler cross-entropy from p^t to p ($KL(p|p^t)$) was chosen [13]. More precisely, the authors decomposed $KL(p|p^t)$ as follows:

$$KL(p|p^t) = -\sum_{i=1}^{n} I(X_i, X_{j(i)}) + \sum_{i=1}^{n} H(X_i) - H(\boldsymbol{X})$$

where $I(X_i, X_{j(i)})$ is the mutual information measure between X_i and $X_{j(i)}$, and $H(X_i)$ and $H(\boldsymbol{X})$ denote the entropy of p with regard to X_i and $\boldsymbol{X}$, respectively. Since the values of $H(\boldsymbol{X})$ and $H(X_i)$ for all i are not influenced by the dependencies in p^t, minimizing $KL(p|p^t)$ is equivalent to maximizing $\sum_{i=1}^{n} I(X_i, X_{j(i)})$. In order to obtain the best approximation, the algorithm considers the set of tree structures where each edge is weighted with the mutual information between the variables involved. Then, they propose a simple method to obtain the structure with the maximum sum of weights, which is the structure of the p^t minimizing $KL(p|p^t)$. Once an estimation of $p_l(\boldsymbol{x})$ is obtained, COMIT samples a number of individuals from $p_l(\boldsymbol{x})$ and selects the best as the initial solutions of a local search. The resulting individuals are then used to create a new population. In TREE, this local search step is eliminated and, thus, the next population is obtained directly from $p_l(\boldsymbol{x})$.

3) Multivariate EDAs: In these EDAs, the probability distribution is estimated by means of graphical models [14], which represent the detected dependencies between the variables through a graph. Thus, the factorization associated with this type of EDAs is as follows:

$$p_l(\boldsymbol{x}) = \prod_{i=1}^{n} p_l(x_i | \boldsymbol{pa_i})$$

where $\boldsymbol{pa_i}$ are the instantiations of $\boldsymbol{Pa_i}$, the set of variables on which X_i depends.

In the Estimation of Bayesian Network Algorithm (EBNA) [15], the factorization of the joint probability distribution is given by a Bayesian network learned from D_{l-1}^{Se}. A Bayesian network is a pair $(S, \boldsymbol{\theta})$ where S is a directed acyclic graph representing the (in)dependencies between the variables and $\boldsymbol{\theta}$ is the set of conditional probability values needed to

define the joint probability distribution. The method used to learn S leads to different EBNA instantiations. For example, EBNA_{BIC} employs a *score + search* approach. That is, each structure is evaluated with a score function value calculated from data and a search method is used to find the best estructure. The score of EBNA_{BIC} is the Bayesian Information Criterion [16], which penalizes the log maximum likelihood of data D given S, resulting in the following function:

$$BIC(S,D) = \sum_{i=1}^{n}\sum_{j=1}^{q_i}\sum_{k=1}^{r_i} N_{ijk}\log\frac{N_{ijk}}{N_{ij}} - \frac{1}{2}\log N(r_i-1)q_i$$

where r_i and q_i are, respectively, the number of values that X_i and its *parent* variables $\boldsymbol{Pa_i}$ can take, N_{ijk} is the number of configurations in D where $\boldsymbol{Pa_i}$ takes its j-th value and X_i takes its k-th value, $N_{ij} = \sum_{k=1}^{r_i} N_{ijk}$ and $N = |D|$.

A usual way to determine $\boldsymbol{\theta}$ is through their maximum likelihood estimates.

IV. An EDAs based Approach for Testing Container Classes

The approach for test data generation of container classes using EDAs relys on a fixed length sequence of FCs. We then look for the sequence with the highest coverage for the fixed length. Thus, given a length L, a solution can be represented as $\boldsymbol{S} = (X_1, Y_1, X_2, Y_2, \ldots, X_L, Y_L)$, where X_i and Y_i denote, respectively, the function called in position i of the sequence and its parameter values, $i = 1, ..., L$. The domain of X_i comes defined by the functions in the class under test. For the Y_i, by contrast, the domain is the set of configurations for the inputs of each function. Besides of the number of input parameters of each function, this set depends on the values each parameter can take. In the case of index type parameters, we choose the interval $[-\frac{\alpha}{10}, \alpha]$, so that approximately a 10% of the values are negative, which allows for *underflow* indices. In the case of element type paremeters, we choose the interval $[-\frac{\alpha}{10}, \alpha]$ once again. In this case, however, we consider negative values as `null`, and the rest of values as `Integer` objects taking the corresponding value.

Additionally, our approach departs from a particular hypothesis: the set of parameter values $\{Y_1, Y_2, \ldots, Y_L\}$ is conditionally dependent on the set of function names $\{X_1, X_2, \ldots, X_L\}$. This implies that, if we consider the test data generation process as the sampling of a probability distribution, the following holds:

$$p(\boldsymbol{S}) = p(y_1, y_2, \ldots, y_L | x_1, x_2, \ldots, x_L) p(x_1, x_2, \ldots, x_L) \quad (1)$$

Hence, a test sequence generation approach conforming to such an hypothesis could select the set of function names independently of the parameter values. Then, the set of parameter values should be generated given the function names, so that the performance of the test sequence could be evaluated. In our case, we use an EDA for selecting the function names, while we resort to a local search for obtaining the parameter values given the function names. So, for the evaluation of each individual in the EDA a local search must be carried out. This allows for regarding at the approach as a search for the best sequence of function names, considering values of their parameters as sub-optimal.

The adopted local search conforms to a best first strategy. A neighbour solution is obtained by adding an increment value of 2 to one input parameter of the current best solution.

The evaluation of a sequence of FCs involves two measures. On the one hand, the number of branches covered is taken into account. On the other, a *branch distance* is calculated for each uncovered branch. The use of such a distance is a common practice in Search Based Test Data Generation for procedural software [17], [18]. Typically, in these approaches, an input covering one branch is sought at a time, thus formulating one optimization problem for each branch. Given branch j and an expression $\mathcal{A}\,\mathbf{OP}\,\mathcal{B}$ of the conditional statement $\mathbf{COND}$ associated with j in the source code, with $\mathbf{OP}$ denoting a comparison operator, the branch distance value for an executed input y which reaches $\mathbf{COND}$ is determined by the following function:

$$h_j(y) = d(\mathcal{A}_y, \mathcal{B}_y) + K \quad (2)$$

where $\mathcal{A}_y$ and $\mathcal{B}_y$ are an appropriate representation of the values taken by $\mathcal{A}$ and $\mathcal{B}$ in the execution, $d(\mathcal{A}_y, \mathcal{B}_y)$ is a distance measurement between $\mathcal{A}_y$ and $\mathcal{B}_y$, and $K > 0$ is a previously defined constant. Commonly, if $\mathcal{A}$ and $\mathcal{B}$ are numerical, then $\mathcal{A}_y$ and $\mathcal{B}_y$ are their values and $d(\mathcal{A}_y, \mathcal{B}_y) = |\mathcal{A}_y - \mathcal{B}_y|$. In the case of more complex data types, a binary representation of the values for $\mathcal{A}$ and $\mathcal{B}$ can be obtained and, for instance, let $d(\mathcal{A}_y, \mathcal{B}_y)$ be the Hamming distance [19].

However, in the case of object oriented software, specific circumstances arise that may not have been considered before in the field. To be precise, instead of just one FC, we have a sequence of them, and, instead of covering just one branch in a particular method, we aim at covering every branch in every method. So, denoting with b the number of branches in the class, we extend the branch distance in Equation 2 as follows [7]:

$$B(S) = \sum_{j=1}^{b} \frac{g(S,j)}{b}$$

where

$$g(S,j) = \begin{cases} 0 & \text{if branch } j \text{ is covered ,} \\ H(S,j) & \text{if branch } j \text{ is not covered and its opposite branch is covered at least twice ,} \\ 1 & \text{otherwise} \end{cases}$$

and

$$H(S,j) = \min\{\frac{h_j(y_i)}{\sum_{k=1}^{L} h_j(y_k)},\ i = 1, \ldots, L\}\ .$$

$H(S,j)$ refers to the minimum normalized $h_j(y_i)$, $i = 1, \ldots, L$. $g(S,j)$ describes how close the sequence S is to cover the not yet exercised branch j. If its associated conditional statement is never reached, $g(S,j) = 1$ since we cannot compute h_j for its predicate. Besides, $g(S,j) = 1$ also

TABLE I

CHARACTERISTICS OF THE CONTAINER CLASSES USED IN THE EXPERIMENTS.

Container	LOC	FuT	Achievable Coverage
`Vector`	1019	34	100
`LinkedList`	708	20	84
`Hashtable`	1060	18	106

if its conditional statement is reached only once. Otherwise, if j will be covered due to the use of $g(S, j)$ during the search, necessarily the opposite of j will not be covered any more (we need to reach the conditional statement at least twice if we want to cover both of its branches).

Finally, we can define the objective function by combining the normalized branch distance of all branches ($B(S)$) and the number of covered branches, $C(S)$, in the following way:

$$f(S) = C(S) + (1 - B(S)) \ .$$

Regardless of the function employed, it is important to realize the significant amount of preprocessing required for the evaluation of a test sequence. That is, the sequence must be previously executed on an instrumented version of the software, which will output the necessary information. Obviously, this instrumented version is created at the beginning of the process.

A. Experimental Results

In order to validate the EDAs based approach for testing containers, we generated test data for three classes, `Vector`, `LinkedList` and `Hashtable`, extracted from the Java API 1.4, package `java.util`. Table I summarizes some of their characteristics. LOC refers to the number of code lines and FuT to the number of functions under test.

For each container, experiments were conducted for three lengths of the sequence of tests, i.e. 15, 45 and 75. The value chosen for α was 20, so that each input parameter took values in the interval $[-2, 20]$. The three EDAs given as examples in Section III were used in the evaluation, that is, UMDA, TREE and EBNA$_{\text{BIC}}$. A common practice in evolutionary algorithms is to fix the population size proportionally to the number of variables. For instance, in [20], several rules of thumb are suggested for a number of EDAs under specific conditions. In the present work, the population size is set at twice the length of the individual, that is, twice the length of the test sequence. The number of selected individuals was set to half the population size. All EDAs used elitism. The stopping criterion was the achievement of complete branch coverage or a maximum number of evaluations, which for the experiments was set to 300000. For the local search, the maximum number of steps was restricted to 40.

Table II shows, for each configuration, the average coverage attained in 20 runs; standard deviations are also included.

In Table II, an intuitive result can be clearly observed: coverage grows as the length of the sequence increases. An exception to this are the results of `Hashtable` for $L = 45$ and $L = 75$. In these cases, the EDAs are stuck in the same coverage value and seem to be unable to improve. A cause for this could be the interval of values chosen for the input parameters, which might prevent from reaching some desirable configurations. Another possible reason could be the strong assumption raised in Equation 1.

TABLE II

COVERAGE RESULTS FOR THE EXPERIMENTS WITH THE EDAS BASED TEST DATA GENERATOR.

Container	L	UMDA	TREE	EBNA$_{\text{BIC}}$
`Vector`	15	66.49 $\pm$ 0.17	62.35 $\pm$ 0.43	66.79 $\pm$ 0.14
	45	90.37 $\pm$ 0.16	91.74 $\pm$ 0.33	91.84 $\pm$ 0.25
	75	97.18 $\pm$ 0.15	98.94 $\pm$ 0.15	98.54 $\pm$ 0.12
`LinkedList`	15	62.5 $\pm$ 0.16	58.02 $\pm$ 0.44	62.55 $\pm$ 0.21
	45	81.4 $\pm$ 0.17	82.3 $\pm$ 0.21	82.4 $\pm$ 0.14
	75	83.75 $\pm$ 0.1	83.95 $\pm$ 0.05	83.8 $\pm$ 0.09
`Hashtable`	15	79.43 $\pm$ 0.24	77.42 $\pm$ 0.42	79.53 $\pm$ 0.2
	45	91 $\pm$ 0	91 $\pm$ 0	91 $\pm$ 0
	75	91 $\pm$ 0	91 $\pm$ 0	91 $\pm$ 0

In order to compare the EDAs' performance, the Mann-Whitney test was conducted for each container and sequence length. Statistically significant differences (95% confidence interval) were found between every pair of EDAs but UMDA and EBNA$_{\text{BIC}}$ in `LinkedList` with $L = 15$, UMDA and EBNA$_{\text{BIC}}$ in `Hashtable` with $L = 15$, and all pairs in `Hashtable` with $L = 45$ and $L = 75$. According to such dissimilarities we may conclude that EBNA$_{\text{BIC}}$ performs the best overall, as it offers the best or second best values in all cases. Bearing in mind these results are not conclusive at all, the numbers conform to a common observation in other domains: multivariate EDAs seem to perform better than simpler alternatives. This reinforces the belief that the capability of these EDAs for reflecting the relationships between variables during the search (the so called *linkage learning* in the field of Genetic Algorithms) is a desirable property. Anyhow, full coverage is attained in none of the cases, so there is a margin for improvement.

An appealing characteristic of EDAs is their intrinsic ability to express the distribution of data through the induced probability model. We believe such capability may provide valuable information about the problems characteristics under certain conditions. Bringing this idea into our problem, for each of the 20 runs of the EBNA$_{\text{BIC}}$, we verified the Bayesian network from which the best solution was obtained. Apropos the structure, all runs consisted of sparse graphs with almost no overlaping in the arcs of different runs. This suggests that, in the context of the algorithm parameters chosen, the results obtained and the initial hypothesis raised above (Equation 1), there is no clear relationship between the positions at which functions are called in the sequence. That is, they are independent of each other. As we noted before, these comments by no means can be taken as definitive statements, though they serve as an example of the utility of EDAs for explaining problem characteristics.

Initialize length L and the probability distribution
Repeat while improvement is obtained
  Seed an EDA with the current probability distribution
  Generate test sequences of length L with the EDA
  Update the resulting probability distribution with k new FCs
  $L \leftarrow L + k$

Fig. 2. Scheme of the EDA based framework for variable length sequences.

V. An EDAs based Framework for Searching most Efficient Tests

Given two sequences offering the same coverage value we prefer the shortest one, since the cost of testing the software object is lower. In fact, the test process could become significantly more expensive if large sequences are employed instead of shorter. The reason for this is a scaling effect, that is, in practice, the number of classes in a system is huge, so that a slight saving in some of them multiplys. Moreover, the fact that increasing the length of a test sequence cannot lead to a decrease in coverage might settle a test data generator in the habit of continuosly increasing the length of the sequence. Therefore, creating sequences as short as possible can be considered as a highly desirable secondary goal in object oriented testing.

Departing from this idea, we developed an EDAs based framework for dealing with distinct test sequence lengths. More precisely, the framework consists of the iterative process outlined in Figure 2.

At each iteration, the EDAs based test data generator is employed for obtaining sequences of length L. Once it is finished, the resulting probability distribution is updated by adding k new variables. If a structure needs to be induced in the EDA, the new variables are disconnected from the rest, i.e. they are considered to be independent. The marginal probability values are obtained from a uniform distribution. Then, a new run of the EDA takes place, departing from the updated probability distribution and with length $L + k$. For the stopping criterion, the efficiency measure of a sequence S, presented in [7], is adopted:

$$Ef(S) = C(S) + \frac{1}{1+L}$$

Thus, the algorithm stops when no improvement in efficiency is observed between two consecutive iterations.

A. Experimental Results

In order to validate this framework we conducted some preliminary experiments over the container classes from Section IV-A. The EDA chosen for the evaluation was EBNA_{BIC}. Each run of this algorithm was stopped when no improvement of the best individual was observed for 5 generations. The rest of the parameter values of the EDA and the local search were kept as in Section IV-A. The value for the initial length from which the framework departed was 45 for `Vector` and `LinkedList`, and 30 for `Hashtable`. Parameter k was set to 10.

Figure 3 presents the average coverage attained in 20 runs for different values of L. The results show that, coverage is improved in all containers as L grows. For `Vector` and `LinkedList`, full branch coverage is reached. For `HashTable`, the framework slightly outperforms the values in Table II, however, it also seems unable to attain full coverage. As noted in the previous section, we identify two possible reasons for this: the interval of values chosen for the parameters of a test input or the assumption raised in Equation 1.

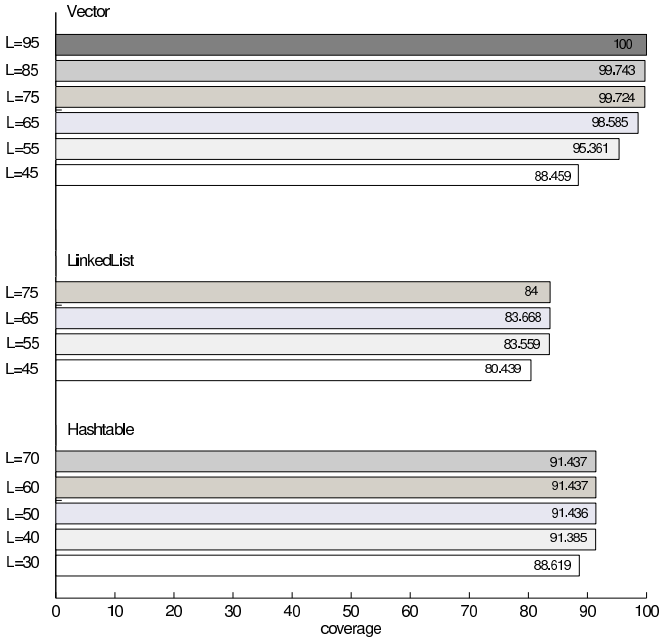

Fig. 3. Coverage results for the experiments with the EDA based framework.

Notice that, since this approach seeks at each round the best sequence of length L, it provides with a means for choosing a test of a given length, which could be beneficial in situations where resources are limited. Taking the values in the figure as an example, in an hypothetical situation where we could not afford the time required for testing container `Vector` with sequences of length 75, we could choose the test with $L < 75$ which best suits our needs.

VI. Related Work

In the present work, we use EDAs for dealing with the test data generation for container classes. Although this topic has remained relatively unexplored until now, a few recent works can be found in the literature.

One of the first studies on the employment of optimization techniques was conducted by Arcuri and Yao [7], [21]. On the one hand, they evaluate four different search algorithms for solving the same problem as in here [7]: a random search, a hill climbing local search, Simulated Annealing and Genetic Algorithms. Besides, a search space reduction that is specific to containers and a novel testability transformation are presented. On the other hand, the authors studied the use of memetic algorithms, again for the same problem [21]. In this latter work, they also developed a novel advanced rep-

resentation for the sequences of FCs, and presented specific operators for the used search techniques.

Visser et al. [22] used exhaustive techniques with symbolic execution. To avoid the generation of redundant sequences, they employed *state matching*. During the exhaustive exploration, the abstract shapes of the class under test in the heap were stored. If an abstract shape is encountered more than once, the exploration of that subspace was pruned.

A container class is object oriented software. Therefore, any system that claims to be able to generate test data for object oriented software should work on containers as well. Tonella [23] was the first to develop such a system using evolutionary algorithms. He used a Genetic Algorithm with special operators to generate unit tests for Java classes.

VII. Conclusions

We have presented an approach for test data generation of object oriented software by means of Estimation of Distribution Algorithms (EDAs). More precisely, we have focused on containers, a common type of object class. In particular, we have dealt with branch coverage for unit testing of such a class type, choosing Java as the programming language. In order to achieve this, the test data generation is posed as an optimization problem whose solution is sought through an EDA. To the best of our knowledge, this is the first application of EDAs to object oriented testing. In addition, we have commented on how the induced probability distribution could be a potential source of knowledge about the problem at hand. Considering the particular conditions of our approach, i.e. input parameters are conditionally dependent on functions and the parameter values chosen for the experiments, we concluded that the positions at which functions are called in the test are independent of each other. By contrast, experimental results suggest further work should be undertaken in order to better understand the behaviour of EDAs in this problem. Finally, an appealing secondary goal of the test case generation for object oriented software is to obtain the most efficient test. We have tackled this issue in the last part of the work, where a preliminary framework has been proposed.

Undoubtedly, much work needs to be carried out in this field. Firstly, we departed from a relatively restrictive assumption (Equation 1) regarding the relationships between functions and input parameters. We believe that the study of other assumptions could be highly beneficial for this problem. EDAs present as a suitable method for this. According to our findings, future work should be addressed towards the elicitation of ways for exploring the probability distribution obtained during the search. In addition, an interesting line of work for seeking efficient test sequences is the handling of variable length solutions within EDAs.

Acknowledgment

This work is supported by an EPSRC grant (EP/D052785/1). The authors also wish to thank Per Kristian Lehre for the useful comments provided during the preparation of this work.

References

[1] B. Beizer, *Software Testing Techniques*. New York: Van Nostrand Rheinhold, 1990.

[2] A. Endres and D. Rombach, *A Handbook of Software and Systems Engineering*. London: Pearson Education Limited, 2003.

[3] P. McMinn, "Search-based software test data generation: a survey," *Software Testing Verification and Reliability*, vol. 14, no. 2, pp. 105–156, 2004.

[4] P. Larrañaga and J. A. Lozano, *Estimation of Distribution Algorithms. A New Tool for Evolutionary Computation*. Boston: Kluwer Academic Publishers, 2002.

[5] J. A. Lozano, P. Larrañaga, I. Inza, and E. Bengoetxea, *Towards a new Evolutionary Computation: Advances in Estimation of Distribution Algorithms*. The Netherlands: Springer-Verlag, 2006.

[6] P. McMinn and M. Holcombe, "The state problem for evolutionary testing," in *Proceedings of the Genetic and Evolutionary Computation Conference*, vol. 2724. Springer-Verlag, 2003, pp. 2488–2498.

[7] A. Arcuri and X. Yao, "Search based testing of containers for object-oriented software," University of Birmingham, School of Computer Science, Tech. Rep. CSR-07-3, 2007.

[8] M. Pelikan, D. E. Goldberg, and F. Lobo, "A survey of optimization by building and using probabilistic models," *Computational Optimization and Applications*, vol. 21, no. 1, pp. 5–20, 2002.

[9] H. Mühlenbein and G. Paaß, "From recombination of genes to the estimation of distributions i. binary parameters," in *Proceedings of the Fourth International Conference on Parallel Problem Solving from Nature*, H. M. Voigt, W. Ebeling, I. Rechenberger, and H. P. Schwefel, Eds. Berlin: Springer-Verlag, 1996, pp. 178–187.

[10] H. Mühlenbein, "The equation for response to selection and its use for prediction," *Evolutionary Computation*, vol. 5, no. 3, pp. 303–346, 1998.

[11] S. Baluja and S. Davies, "Combining multiple optimization with optimal dependency trees," Carnegie Mellon University, Pittsburgh, Tech. Rep. CMU-CS-97-157, 1997.

[12] C. Chow and C. Liu, "Approximating discrete probability distributions with dependence trees," *IEEE Transactions on Information Theory*, vol. 14, no. 3, pp. 462–467, 1968.

[13] S. Kullback and R. A. Leibler, "On information and sufficiency," *Annals of Mathematical Statistics*, vol. 22, pp. 79–86, 1951.

[14] E. Castillo, J. Gutiérrez, and A. Hadi, *Expert Systems and Probabilistic Network Models*. New York: Springer-Verlag, 1997.

[15] P. Larrañaga, R. Etxeberria, J. A. Lozano, and J. M. Peña, "Combinatorial optimization by learning and simulation of bayesian networks," in *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence*, C. Boutilier and M. Goldszmidt, Eds. San Francisco, CA: Morgan Kaufmann, 2000, pp. 343–352.

[16] G. Schwarz, "Estimating the dimension of a model," *Annals of Statistics*, vol. 7, no. 2, pp. 461–464, 1978.

[17] N. Tracey, J. Clark, K. Mander, and J. McDermid, "An automated framework for structural test-data generation," in *Proceedings of the 13th IEEE Conference on Automated Software Engineering*, D. Redmiles and B. Nuseibeh, Eds. IEEE CS Press, 1998, pp. 285–288.

[18] J. Wegener, A. Baresel, and H. Sthamer, "Evolutionary test environment for automatic structural testing," *Information and Software Technology*, vol. 43, no. 14, pp. 841–854, 2001.

[19] H. Sthamer, "The automatic generation of software test data using genetic algorithms," Ph.D. dissertation, University of Glamorgan, Pontyprid, Wales, Great Britain, 1996.

[20] H. Mühlenbein and T. Mahnig, "Mathematical analysis of evolutionary algorithms for optimization," in *Proceedings of the Third International Symposium on Adaptive Systems*, La Havana, Cuba, 2001, pp. 166–185.

[21] A. Arcuri and X. Yao, "A memetic algorithm for test data generation of object-oriented software," in *Proceedings of the IEEE Congress on Evolutionary Computation*, 2007, to appear.

[22] W. Visser, C. S. Pasareanu, and R. Pelànek, "Test input generation for java containers using state matching," in *Proceedings of the International Symposium on Software Testing and Analysis (ISSTA)*, 2006, pp. 37–48.

[23] P. Tonella, "Evolutionary testing of classes," in *Proceedings of the International Symposium on Software Testing and Analysis (ISSTA)*, 2004, pp. 119–128.