A Model-Driven Approach to Generate Relevant and Realistic Datasets

Adel Ferdjoukh, Eric Bourreau, Annie Chateau, Clémentine Nebut
Lirmm, CNRS and University of Montpellier
{firstname.lastname}@lirmm.fr

Keywords: Relevant datasets generation, Meta-model instantiation, Probabilistic simulation

Abstract: Disposing of relevant and realistic datasets is a difficult challenge in many areas, for benchmarking or testing purpose. Datasets may contain complexly structured data such as graphs or models, and obtaining such kind of data is sometimes expensive and available benchmarks are not as relevant as they should be. In this paper we propose a model-driven approach based on a probabilistic simulation using domain specific metrics for automated generation of relevant and realistic datasets.

1 Introduction & Motivations

Developing software handling complex structured data requires to dispose of datasets to validate the built software. For example, when developing algorithms to assist the reconstruction of genomic sequences, sets of DNA sequences are required to experiment the algorithms. Similarly, to validate a program handling models (called a model transformation), a poll of relevant models is required. Disposing of such datasets of structured data is usually complex. Data is itself complex, and the dataset must fulfil properties such as relevancy and realism: it must contain data that look like real data. Some approaches propose an *ad hoc* generation of test cases, for instance in [10], the author presents an approach for generation of uniform lambda terms for testing a compiler of Haskell language. However, characterizing data is not fully intuitive in the proposed way.

In this paper, we propose an alternative approach for automated generation of relevant datasets. Our approach uses domain specific metrics and simulates usual probability distributions in order to generate relevant and realistic datasets. The approach is model-based: the wished data is modelled in a metamodel, and the data generation is achieved through the metamodel instantiation. Though metamodels are a strong tool to represent the abstraction underlying data, existing metamodel instantiation approaches encounter difficulties to adapt to various concrete situations, and suffer from a lack of realism in produced models. Our contribution is to inject relevance and realism into the model generation process approaching the characteristics of the desired models by the mean of *a priori* probability distributions concerning metrics about elements in models.

The rest of the paper is organised as follows. Related work is presented in Section 2. Section 3 relates the simulation of probability distribution and its integration to our original approach $\mathcal{G}$RIMM. Section 4.1 presents our first case study, the generation of real-close skeletons of Java projects. The second case study, generation of relevant scaffold graphs, is described in Section 4.2.

2 Related work

We give here an overview of contributions from the literature that are close to our proposal, i.e. in the fieds of model generation through automated meta-model instantiation. Many underlying techniques have been used. *Cabot et al.* [2] translate a meta-model and its OCL constraints into constraint programming. In [9], *Mougenot et al.* use random tree generation to generate the tree structure of a model. *Wu et al.* [12] translate a meta-model into SMT (Satisfiability Modulo Theory) in order to automatically generate conform models. In [3], *Ehrig et al.* transform a meta-model into a graph grammar which is used to produce instances. The advantages and drawbacks of our original approach relatively to the other generating methods have been discussed in [4].

Nevertheless, only two approaches have treated the problem of relevance and realism of generated models. In [9], authors use a uniform distribution during the generation process and add weights in order to influence the frequency of appearance of different elements. In [12], authors describe two techniques to obtain relevant instances. The first one is the use

DOI reference number: 10.18293/SEKE2016-029

of partition-based criteria which must be provided by the users. The second one is the encoding of common graph-based properties. For example, they want to generate acyclic graphs, *i.e.* models.

Our goal being the generation of relevant and realistic data, we infer characteristics from real models, in a way that allows us to generate models which are close to reality.

3 Generation of relevant models

$\mathcal{G}$RIMM ($\mathcal{G}$ene**R**ating **I**nstances of **M**eta-**M**odels) method [4, 5] relies on translating meta-models into constraint satisfaction problems (CSP) in order to produce conform instances (models). In a nutshell, we encode the classes, references and OCL constraints of a meta-model into a set of variables connected by constraint relationships. A CSP solver performs a smart exhaustive search for values which satisfy the given constraints. Generation is fast and provides models which are guaranteed to be conform to the meta-model.

The main remaining issue is related to the usefulness of generated models. Indeed, we have to produce models that are as realistic as possible, regarding to the data it is supposed to simulate. We propose here a method that intends to achieve this goal.

The declarative approach is intrinsically deterministic, since the solver follows a deterministic algorithm to produce a unique solution. The CSP solver can easily produce thousands of solutions, but they are often far from the reality. Here we take into account the flexibility given by the CSP to encode various parameters before the solving process, and the fact that some elements of the real models follow usual probability distributions. These distributions are simulated and, *a priori*, injected to the CSP, in order to produce generated models closer to real ones.

3.1 Sampling probability distributions

Generating samples of well-known probability distributions is a way to add randomness to the deterministic CSP solving process. The idea is to get models that have more diversity in their elements' degrees and their attributes' values in order to cover a lot of possible values. For example, when generating UML models, we want to generate a package which has 5 classes, another one with 7 classes and so on.

Figure 1 shows the basic operation with which we can sample all usual probability distributions whatever they are continuous or discrete. Thus, to generate a sample of a random variable X we need its cumulative function $F(X)$ and a sample of uniform values u. Result values x are obtained by an inversion of F: $u = F(x) \Rightarrow x = F^{-1}(u)$.

Previous method is then adapted to each probability distribution we want to sample.

Discrete distribution on a finite set For all discrete distribution, are given the probabilities of a finite set of values. The cumulative function is then deduced from the accumulation of probabilities and a sample can be easily generated.

Inverse cumulative function method This method is used for continuous distribution if their inverse cumulative function is easily computable. This method is used to simulate the exponential distribution ($\varepsilon(\lambda)$).

Normal distribution: Box Muller transform Sometimes, inverting a cumulative function is difficult. In these cases, special algorithms are used. For example, a normal distribution ($\mathcal{N}(\mu,\sigma)$) does not have a known inverse function, so previous method is useless. However, many other methods exist to simulate a normally distributed sample. Our implementation uses Box Muller algorithm.

For a more complete overview about probability and simulation, please refer to [7]

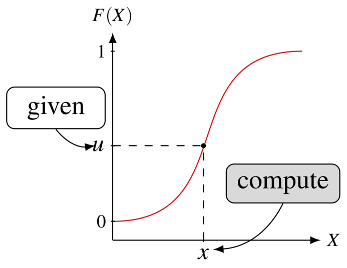

Figure 1: Simulation of random values x given a cumulative function $F(X)$ of a random variable X and uniform u

3.2 Integration into $\mathcal{G}$RIMM

Many methods coming from the area of random graphs (see for example, [1]) use specific degree distributions to generate random graphs. Our idea is to apply such a method to randomly generate relevant models. Indeed, models are also graphs, in which vertices and edges are typed (classes and relations). There exist many domain-based metrics on the elements of models. We propose to use those metrics (viewed as probability distributions) in order to improve the relevance of randomly generated models.

The degree distribution of a link As well as in graphs, choosing the number of elements to link with a vertex (its degree) is a key to generate realistic models. The observation of real models shows us that the degrees are diverse. So the greater is the diversity of degrees, the more realistic will be the models. In our method, the users provide probability distributions on the degree of some associations or references. Then, these distributions are simulated and integrated to the $\mathcal{G}$RIMM process.

Distribution on the value of an attribute The values of an attribute are also very important for the relevance of models. Indeed, generated models should have attributes with values that are close to real models. Distributions for attributes are given by the user. A probability is defined for each possible value, and simulation will choose adequate values and assign them to class instances.

Improving the connectivity *Molloy and Reed* show in [8] that the parametrisation of a random graph generator can influence the connectivity of the generated graphs. Indeed, choosing the adequate number of vertices and edges, and the right degree distribution gives graphs that are more connected. Our method takes into account this important aspect during the simulation process in order to get the most connected models.

Figure 2 shows the different inputs (white boxes) and the different steps (grey boxes) of our new tool $\mathcal{G}$ЯRIMM ($\mathcal{G}$enerating **Я**andomized and **R**elevant **I**nstances of **M**eta-**M**odels).

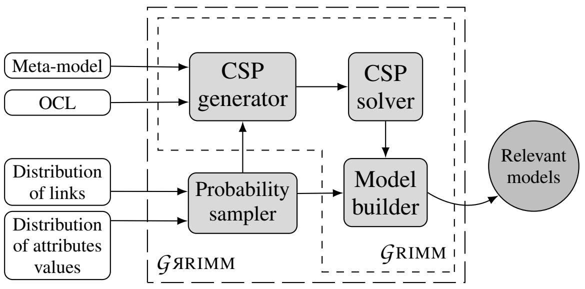

Figure 2: Methodology process.

4 Case studies

We experimentally show in this section how taking probability distributions improves the quality of generated datasets. We consider two case studies, one from Software Engineering area and the other from Bioinformatics. All data concerning the two case studies and the evaluation can be found at: `http://www.lirmm.fr/~ferdjoukh/english/experiments4Seke.html`.

4.1 Java code generation

One of the main objectives of our approach is the generation of benchmarks of test programs for different applications, such as compilers or virtual machines. In this experiment, we generate realistic and

Metric	$\rightsquigarrow$	**Theoretical distrib.**
Class/Package	$\rightsquigarrow$	$\varepsilon(\frac{1}{8.387})$
Methods/Type	$\rightsquigarrow$	$\varepsilon(\frac{1}{7.06})$
Attributes/Type	$\rightsquigarrow$	$\mathcal{N}(3.46, 2.09)$
Constructor/Type	$\rightsquigarrow$	$\mathcal{N}(0.73, 0.26)$
Sub-Classes/Classe	$\rightsquigarrow$	$\varepsilon(\frac{1}{0.22})$
% Interface/Package	$\rightsquigarrow$	$\varepsilon(\frac{1}{8.001})$
Parameters/Methods	$\rightsquigarrow$	$\mathcal{N}(0.87, 0.25)$

Table 1: Chosen code metrics with their theoretical probability distribution. ε: Exponential distribution, $\mathcal{N}$: Normal distribution.

relevant skeletons of Java programs using real code measurements. We choose Java for facility to find real programs to collect desired measurements. However, our method can be applied to any programming language. We collected 200 real Java projects coming from two corpus (Github and Qualitas corpus[1]). For more heterogeneity, we chose projects having different sizes (big project for qualitas corpus and smaller ones from github) and different origins (well-known software such as Eclipse, Apache or ArgoUML and also small software written by only one developer). We measured metrics related to their structure, such as the percentage of concrete classes/ abstract classes, the average number of constructors for a class, the visibility of fields and methods, etc [6]. To measure these metrics we used an open source tool called `Metrics`[2]. After that, we use R software[3] to compute histogram of each metric in order to deduce its theoretical probability distribution. Table 1 gives the different metrics and their theoretical probability distributions.

According to these metrics, we automatically generate Java programs having the same characteristics as the real ones. To achieve this goal, we design a meta-model representing skeletons of Java projects and we adjoin some OCL constraints. 300 Java projects are generated using three versions of our approach. Four corpus are then compared: (1) projects generated by $\mathcal{G}$RIMM but without OCL (2) projects generated by $\mathcal{G}$RIMM (3) projects generated by $\mathcal{G}$ЯRIMM (4) real Java projects. Figure 3 gives the distributions of constructors per class for each corpus. We observe that the two first versions without probability distributions give results that are very far from the characteristics of real models. On the other hand, introducing simulated probability distributions leads to substantial improvement. We see that the distribution of the number

[1]Qualitas corpus: `http://qualitascorpus.com/docs/catalogue/20130901/index.html`

[2]Metrics tool: `http://metrics.sourceforge.net`

[3]R softaware: `https://www.r-project.org/`

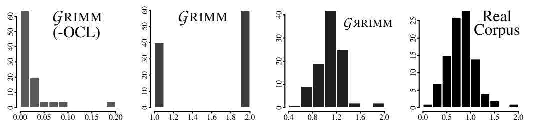

Figure 3: Comparing the number of constructors per class in Java projects. x: constructors per class, y: frequency.

of constructors of generated models are close to real ones. Moreover, these results are always better when adding probabilities for all other measurements presented in Table 1.

4.2 Scaffold graphs generation

Scaffold graphs are used in Bioinformatics to assist the reconstruction of genomic sequences. They are introduced late in the process, when some DNA sequences of various lengths, called *contigs*, have already been produced by the assembly step. Scaffolding consists in ordering and orienting the contigs, thanks to oriented relationships inferred from the initial sequencing data. A scaffold graph is built as follows: vertices represent extremities of the contigs, and there are two kind of edges. Contig edges link both extremities of a given contig (strong edges in Figure 5), whereas scaffolding edges represent the relationship between the extremities of distinct contigs. Contig edges constitute a perfect matching in the graph, and scaffolding edges are weighted by a confidence measure. Those graphs are described and used in the scaffolding process in [11] for instance. The scaffold problem can be viewed as an optimisation problem in those graphs, and consists in exhibiting a linear sub-graph from the original graph. Therefore it can be considered as well as a model transformation, when models conform to the Scaffold graph meta-model that we designed. Producing datasets to test the algorithms is a long process, somehow biased by the choices of the methods (DNA sequences generation, assembly, mapping), and there does not exist a benchmark of scaffold graphs of various sizes and densities. Moreover, real graphs are difficult and expensive to obtain. Thus, it is interesting to automatically produce scaffold graphs of arbitrary sizes, with characteristics close to the usual ones. In [11], the authors present some of these characteristics, that are used here to compare the GRIMM instances vs. the "hand-made" graphs.

The probability distribution chosen to produce the graph emerges from the observation that the degree distribution in those graphs is not uniform, but follows an exponential distribution. We compare several datasets, distributed in several classes according to their sizes: (1) graphs generated by GRIMM, (2)

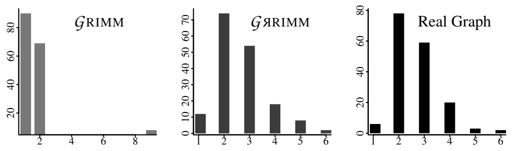

Figure 4: Comparing the degree distribution between a real scaffold graph and its equivalent generated ones (168 nodes and 223 edges).x: degree, y: frequency.

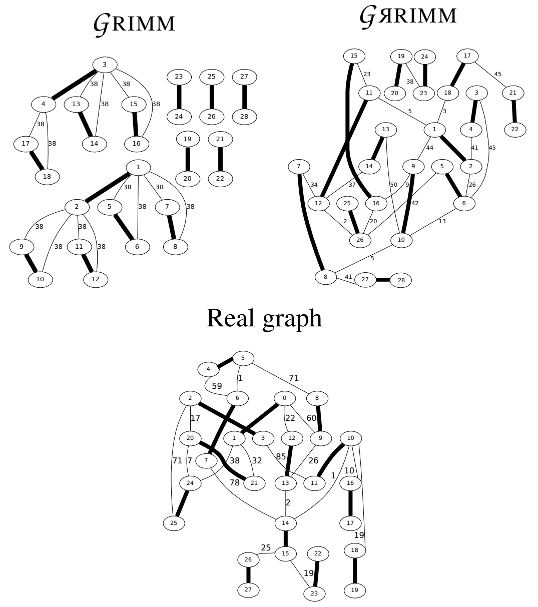

Figure 5: Three Scaffold graphs corresponding to the same species (monarch butterfly). Strong edges represent contig edges, other edges are scaffolding edges.

graphs produced by GPRIMM and (3) real graphs of different species, described in [11].

For each real graph, 60 graphs of the same size are automatically generated. 30 graphs are naively generated using the original GRIMM method [4, 5], and 30 others are generated after the simulating of probability distribution. These models are then compared in term of visual appearance (Figure 5), degree distribution (Figure 4) and according to some graph measurements (Table 2).

We see in Figure 5 three models (scaffolds graphs) corresponding to the same species (monarch butterfly[4]). The naive method generates a graph that does not look like the real one. This graph is too weakly connected, and the connected parts have a recurring pattern. This is not suitable for a useful scaffold graph. Whereas, introducing probabilities provides graphs having shapes close to reality. Thus, both real graphs and generated graphs (with probability distri-

[4]It refers to mitochondrial DNA of monarch butterfly.

Graph size			Measurements					
			$\mathcal{G}$RIMM generation		**$\mathcal{G}$ЯRIMM generation**		**Real graphs**	
Species	nodes	edges	min/max degree	h-index	min/max degree	h-index	min/max degree	h-index
monarch	28	33	1/9	3	1/ 4.6	4.06	1/ 4	4
ebola	34	43	1/9	3	1/ 4.83	4.60	1/ 5	4
rice	168	223	1/9	8	1/ 6.03	5.93	1/ 6	5
sacchr3	592	823	1/9	10	1/ 7	6.76	1/ 7	6
sacchr12	1778	2411	–	–	1/ 7.53	7	1/10	7
lactobacillus	3796	5233	–	–	1/ 8.06	7.8	1/12	8
pandora	4092	6722	–	–	1/ 8.23	7.96	1/ 7	7
anthrax	8110	11013	–	–	1/ 8.3	8.03	1/ 7	7
gloeobacter	9034	12402	–	–	1/ 8.46	8	1/12	8
pseudomonas	10496	14334	–	–	1/ 8.43	8	1/ 9	8
anopheles	84090	113497	–	–	1/ 8.96	9	1/ 51	12

Table 2: Comparing graph metrics on real scaffold graphs and average for 60 generated ones for each species.

bution) are strongly connected, and more randomness can be observed in the connections and the weights of edges.

Figure 4 compares the degree distributions for three scaffold graphs of the same species. We see that generating with probabilities gives a distribution very similar to the distribution in the real graph.

Table 2 compares the three benchmarks of scaffold graphs (naive generation, generation with probabilities and real graphs) according to some graph measurements. We can observe again, that the graphs generated with probabilities are closer to real graphs than the naively generated graphs in all cases. The measurements on the naive graph suffer from a lack of diversity and randomness. Indeed, the minimal and the maximal degree are the same for all generated models. This, of course, does not reflects the reality. Notice also that it was not possible with the naive generation method to generate largest graphs corresponding to largest genomes.

5 Conclusion & Discussion

In this work, we presented a model driven approach to generate realistic and useful datasets. Our approach exploits domain-based metrics and the simulation of probability distributions to tackle the issue of relevance for generated instances. We evaluated our work by applying the method to two different fields: generation of source code skeletons in Software Engineering and generation of scaffold graphs in Bioinformatics. The method follows four main steps: (1) Design a meta-model to represent the domain, (2) Collect metrics on real models (The metrics can also be related to a specific use, so given by an expert), (3) Sample probability distributions with respecting previous metrics and (4) Generate realistic and relevant instances using $\mathcal{G}$ЯRIMM tool. We observed a substantial improvement of relevance when simulated probabilities samples are added to the model generator. New generated instances have characteristics very close to real models, improving their usefulness for testing programs. This is especially interesting when data is rare, difficult or expensive to obtain, like in the case of scaffold graphs.

REFERENCES

[1] B. Bollobás. *Random Graphs*. Cambridge University Press, 2nd edition, 2001.

[2] J. Cabot, R. Clarisó, and D. Riera. Verification of UML/OCL Class Diagrams using Constraint Programming. In *IEEE ICSTW*, pages 73–80, 2008.

[3] K. Ehrig, J. M. Kister, G. Taentzer, and J. Winkelmann. Generating Instance Models from Meta Models. In *FMOODS*, pages 156–170, 2006.

[4] A. Ferdjoukh, A.-E. Baert, E. Bourreau, A. Chateau, R. Coletta, and C. Nebut. Instantiation of Meta-models Constrained with OCL: a CSP Approach. In *MODELSWARD*, pages 213–222, 2015.

[5] A. Ferdjoukh, A.-E. Baert, A. Chateau, R. Coletta, and C. Nebut. A CSP Approach for Metamodel Instantiation. In *IEEE ICTAI*, pages 1044–1051, 2013.

[6] B. Henderson-Sellers. *Object-Oriented Metrics: Measures of Complexity*. Prentice Hall, 1996.

[7] M. Mitzenmacher and E. Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, 2005.

[8] M. Molloy and B. Reed. A Critical Point for Random Graphs with a Given Degree Sequence. *Random Structures & Algorithms*, 6(2-3):161–180, 1995.

[9] A. Mougenot, A. Darrasse, X. Blanc, and M. Soria. Uniform Random Generation of Huge Metamodel Instances. In *ECMDA*, pages 130–145, 2009.

[10] M. Pałka. *Random Structured Test Data Generation for Black-Box Testing*. PhD thesis, University of Gøteborg, 2014.

[11] M. Weller, A. Chateau, and R. Giroudeau. Exact approaches for scaffolding. *BMC Bioinformatics*, 16(14):1471–2105, 2015.

[12] H. Wu, R. Monahan, and J. F. Power. Exploiting Attributed Type Graphs to Generate Metamodel Instances Using an SMT Solver. In *TASE*, pages 175–182, 2013.