ATLANTIS PRESS
Journal of Robotics, Networking and Artificial Life, Vol. 5, No. 2 (September 2018) 122-127

# Genetic Algorithm-Based Technique and Tool for Generating Mutants of Extended Place/Transition Nets

**Tomohiko Takagi**
*Faculty of Engineering and Design, Kagawa University*
*2217-20 Hayashi-cho, Takamatsu-shi, Kagawa 761-0396, Japan*

**Shogo Morimoto**
*Graduate School of Engineering, Kagawa University*
*2217-20 Hayashi-cho, Takamatsu-shi, Kagawa 761-0396, Japan*
*E-mail: takagi@eng.kagawa-u.ac.jp, s17g483@stu.kagawa-u.ac.jp*

**Abstract**

An EPN (Extended Place/transition Net) is used as a formal model that represents the behavior of software. When mutation testing is performed based on the EPN, failures are intentionally inserted into an original EPN (EPN that represents the expected behavior of software) in order to create mutant EPNs. A large number of higher-quality mutant EPNs are needed to expect the higher degree of accuracy for a mutation score, but the techniques to generate them have not been established. To address this problem, we construct a technique to generate mutant EPNs, and develop a tool to support the technique. In this technique based on a genetic algorithm, a set of mutant EPNs corresponds to a chromosome, and the fitness of each chromosome is evaluated based on an original EPN weighted by metrics. This paper shows the procedure of this technique, the functions of the tool, and the discussion about its effectiveness.

*Keywords*: Mutation Testing, Model-Based Testing, Place/Transition Net, Genetic Algorithm

## 1. Introduction

An EPN (Extended Place/transition Net) is a place/transition net that includes actions and guards written in a specification description language of VDM (Vienna Development Method),[1] and it is used as a formal model to represent the relatively detailed behavior of software. When model-based mutation testing[2] is performed based on the EPN, failures are intentionally inserted into an original EPN (EPN that represents the expected behavior of software) in order to create mutant EPNs, and then a test suite (a set of test cases) to be evaluate is executed on the mutant EPNs.[3-5] The ratio of killed mutant EPNs (mutant EPNs whose failures were detected by the test suite) to all of the created mutant EPNs is called a mutation score, and it gives test engineers a measurement of the quality of the test suite. A large number of higher-quality mutant EPNs are needed to expect the higher degree of accuracy for a mutation score.

In our previous study,[5] we developed a tool to construct an original EPN, construct mutant EPNs, execute a test suite, calculate a mutation score, and so on. However, the systematic techniques to generate higher-quality mutant EPNs had not been established. The tool provided only two simple ways to construct mutant EPNs, that is, the way to generate mutant EPNs at random by use of model-based mutation operators, and the way to construct mutant EPNs by manual application of model-based or code-based mutation operators.

To address this problem, we construct a technique to generate better mutant EPNs, and develop a tool to

support the technique. In this technique based on a GA (Genetic Algorithm), a set of mutant EPNs corresponds to a chromosome (an individual), and the fitness of each chromosome is evaluated based on an original EPN weighted by metrics. Test engineers can select suitable metrics for the evaluation, that is, can define the meaning of "quality of mutant EPNs".

The rest of this paper is organized as follows. In section 2, we propose a technique to generate better mutant EPNs. Section 3 shows the functions of the tool to support the proposed technique. Section 4 gives the discussion about the effectiveness of the tool, and finally section 5 shows a conclusion and future work.

## 2. Mutant EPN Generation Technique

In this section, we propose a GA-based technique to generate better mutant EPNs.

### 2.1. *Overview of the way to address problems*

A mutant EPN is an EPN that includes an intentional failure, and it is created by applying one or more mutation operators to an original EPN that test engineers had constructed based on specifications of SUT (Software Under Test). The mutation operators can be broadly classified into model-based ones[3] and code-based ones. The combination of the mutation operators and the parts of the original EPN to be mutated brings astronomical numbers of possible mutant EPNs, and therefore they should be selected based on quality. However, it is not obvious (a) how to evaluate the quality of mutant EPNs, and (b) how to apply mutation operators in order to generate high-quality mutant EPNs.

In this study, we introduce weights[4] into an original EPN in order to address (a). The weights are values between 0.0 and 1.0 that represent fault-proneness and the impact on software reliability. They are calculated based on metrics relating to usage distribution, complexity, and test execution history, and then are given to each transition of an original EPN. It is assumed that a good mutant EPN includes an intentional failure relating to a highly-weighted transition, and the quality of mutant EPNs can be evaluated based on the weights.

The weights are calculated by the following steps.

(i) Test engineers select metrics suitable for SUT, and then gather materials for evaluation of the metrics.
(ii) Test engineers define the relationships between the original EPN and the materials.

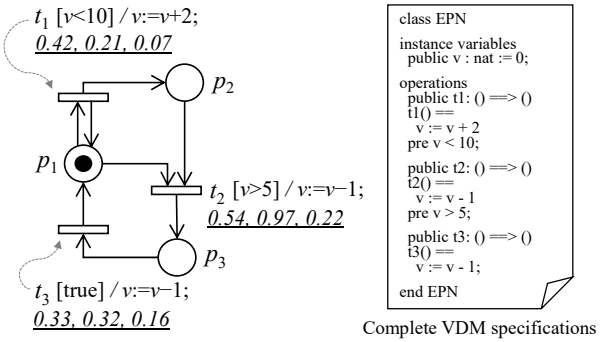


Fig. 1. Example of an original EPN with weights.

(iii) The metrics are evaluated based on the materials.
(iv) According to the results of (ii), the results of (iii) are given to each transition of the original EPN, and then they are converted to values between 0.0 and 1.0.

A simple example of an original EPN with weights is shown in Fig. 1. It consists of three places ($p_1$, $p_2$, and $p_3$), three transitions ($t_1$, $t_2$, and $t_3$), and eight arcs. There is a token on $p_1$. Each transition has a guard, an action, and three weights, since three kinds of metrics have been selected. For example, $t_1$ has "$v$<10" as a guard, "$v$:=$v$+2;" as an action, and "0.42, 0.21, 0.07" as weights.

Also, we introduce a GA into our mutant EPN generation technique in order to address (b). The GA is a well-known metaheuristic approach that imitates natural selection, and it is suitable to solve a problem in which it is difficult to formulate equations and get a best solution within reasonable time and cost. In our technique, a set of mutant EPNs corresponds to a chromosome, and the fitness of each chromosome is evaluated based on an original EPN weighted by metrics. Chromosomes with higher fitness tend to remain in next generation (and thus tend to take part in producing offspring). Finally a chromosome with highest fitness is selected as a solution. The detailed procedure of this GA-based technique is shown in the next section.

### 2.2. *Procedure of the GA-based technique*

The procedure of our GA-based technique to generate mutant EPNs consists of the following six steps.

(i) An initial population is generated for the first generation. The population consists of $N_c$ ($N_c \geq 2$) chromosomes, and each chromosome consists of $N_g$ ($N_g \geq 2$) genes. A chromosome represents a set of

mutant EPNs (that is, a candidate solution). $N_g$ is the number of genes, that is, the number of mutant EPNs that test engineers need for the evaluation of a mutation score. Each gene is generated by applying a mutation operator to an original EPN. A mutation operator to be used is randomly selected based on a uniform distribution, and a part of the original EPN to be mutated is randomly selected according to a distribution of weights.

(ii) In order to generate new chromosomes (that is, offspring), mutation and crossover as genetic operators are applied to the existing chromosomes (that is, parents) in the population of current generation.

In mutation, chromosomes are randomly selected with a mutation probability $p_m$ ($0.0 \leq p_m \leq 1.0$), and then genes are randomly selected with $p_m$ in the selected chromosomes. The selected genes are replaced with new genes that are generated by the same way as (i). In crossover, chromosomes are randomly selected with a crossover probability $p_c$ ($0.0 \leq p_c \leq 1.0$) in order to make pairs, and then genes between randomly selected cut positions are exchanged in a copy of each pair. Fig. 2 shows a simple example of crossover in $N_g$=5. $c_x$ ($x$=1, 2, ⋯, 4) and $g_y$ ($y$=1, 2, ⋯, 10) represent chromosomes and genes, respectively. $c_1$ and $c_2$ are selected as parents. Crossover is applied to the pair of $c_1$ and $c_2$, and new chromosomes $c_3$ and $c_4$ are generated by exchanging the genes between cut positions (that is, $g_3$, $g_4$, $g_8$, and $g_9$).

Finally, new chromosomes are added to the population of current generation.

(iii) Fitness of each chromosome in the population of current generation is evaluated by the following equation.

$$fitness(c) = \sum_{i=1}^{M} imp_i \cdot eval_i(c) \quad (1)$$

In Eq. (1), $c$ is a chromosome to be evaluated, $M$ ($M \geq 1$) is the number of kinds of metrics that test engineers introduced to calculate weights, $imp_i$ expresses the degree of importance of $i$th metrics, and satisfies the following.

$$\begin{cases} \sum_{i=1}^{M} imp_i = 1.0 \\ 0.0 \leq imp_i \leq 1.0 , \quad (i=1, 2, \cdots, M) \end{cases} \quad (2)$$

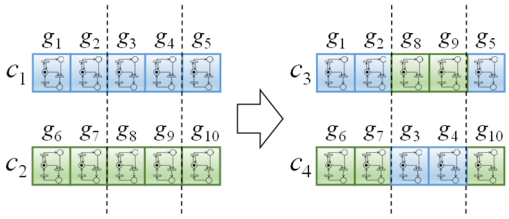


Fig. 2. Example of crossover.

$eval_i(c)$ in Eq. (1) expresses the result of evaluating $i$th metrics in $c$, and it is defined as follows.

$$eval_i(c) = \frac{\sum_{j=1}^{T} \left\{ weight_{i,j} \cdot (1 - 2^{-num(c,j)}) \right\}}{\sum_{j=1}^{T} weight_{i,j}} \quad (3)$$

In Eq. (3), $T$ is the number of transitions of the original EPN, $weight_{i,j}$ is a value of a weight given by $i$th metrics on $j$th transition, and $num(c, j)$ expresses the number of genes of $c$ in which $j$th transition and/or its ingoing/outgoing arc(s) are mutated. If equivalent mutant EPNs are detected in $c$, they are ignored on the calculation of $num(c, j)$.

(iv) If the number of alternation of generations reaches $G$ ($G \geq 1$), a chromosome with the highest fitness through all generations is selected as a final solution, and the procedure of this technique is terminated.

(v) $N_c$ chromosomes that the initial population of next generation consists of are selected from the population of current generation. To be precise, two chromosomes are randomly selected based on a uniform distribution from the population of current generation, and their fitness are compared. Then one with higher fitness is selected and is moved to the population of next generation, and another is returned to the population of current generation. These are repeatedly performed until the number of the population of next generation reaches $N_c$.

(vi) Generation is changed, and then the procedure of this technique returns to (ii).

## 3. Functions of a Tool for Mutant EPN Generation

We have developed a tool for EPN-based mutation testing. This section shows the functions of the tool to support the proposed technique.

The tool includes (a) a function to construct an original EPN, (b) a function to generate mutant EPNs, (c) a function to execute a test suite on mutant EPNs and

calculate its mutation score. (a) and (c) had already been implemented and discussed in our previous study,[5] and therefore we concentrate on the implementation of (b) in this study. (b) includes two new sub-functions, that is, a sub-function to calculate weights based on metrics, and a sub-function to generate better mutant EPNs by the GA-based technique.

### 3.1. *Weight calculation sub-function*

As already discussed in section 2.1, we introduce weights into an original EPN, and our tool provides a sub-function to calculate weights based on metrics. Fig. 3 shows a screen shot of our tool (a main window) that is executing the sub-function. Its GUI (Graphical User Interface) consists of (A) a pane to start calculation and output the results, (B) a pane to select materials for the evaluation of metrics, and (C) a pane to show an original EPN with weights.

A user of our tool, that is, a test engineer first selects materials by using (B). (B) consists of three tab pages that specialize in materials relating to usage distribution, complexity, and test execution history, respectively. For example, in Fig. 3, an access log file of Apache is selected in order to evaluate usage distribution of a Web application to be tested. Keywords of actions that appear in the selected access log file are automatically extracted based on a specified log format, and a test engineer defines the relationship between the actions and the transitions of an original EPN.

After selecting materials, a test engineer pushes a button to start calculation on (A), and then confirms its results on (C). In Fig. 3, weights based on usage distribution, complexity, and test execution history are displayed in blue, red, and green characters, respectively. The usage distribution was measured by REET (Ratio of the frequency of Executing Each Transition). The complexity was given by $1.0-MI/100$, where $MI$ means a maintainability index. The test execution history was evaluated by RFTC (Rate of the frequency of Failing Test Cases).

If there are no problems on the results, a test engineer pushes an output button on (A) in order to generate an XML (eXtensible Markup Language) file of an original EPN with weights.

### 3.2. *Mutant EPN generation sub-function*

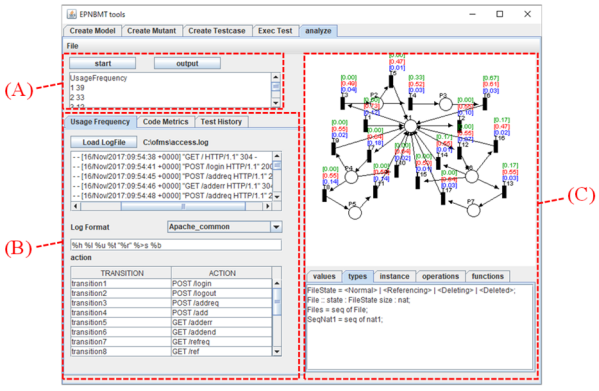


Fig. 3. Screen shot of the tool that is executing the weight calculation based on metrics.

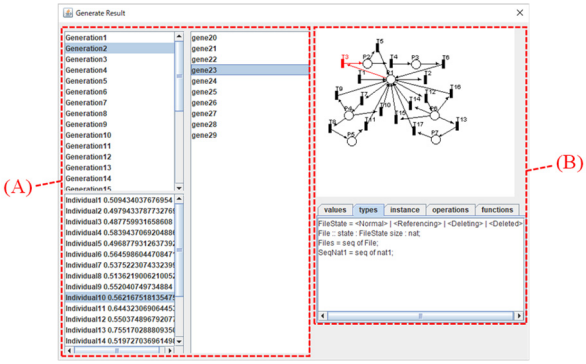


Fig. 4. Screen shot of the tool that shows the results of mutant EPN generation by the GA-based technique.

Our tool also provides a sub-function to generate better mutant EPNs by the GA-based technique that was discussed in section 2.2. After completing the calculation of weights, a test engineer specifies the parameter values for the GA-based technique on a dialog box of our tool, and our tool starts mutant EPN generation. Fig. 4 shows a screen shot of our tool (a sub-window) that gives the results of mutant EPN generation to a test engineer. Its GUI consists of (A) a pane to show the lists of all genes of all chromosomes in all generation, and (B) a pane to show a selected mutant EPN.

On (A), a test engineer selects an arbitrary generation from the upper left list, and all chromosomes with fitness in the selected generation are shown in the lower left list. Additionally, a test engineer selects an arbitrary chromosome from the list, and all genes of the selected chromosome are shown in the right list. If a test engineer

selects an arbitrary gene in the list, a mutant EPN that corresponds to the selected gene is shown in (B). A mutated part in the mutant EPN is highlighted in order that a test engineer can easily understand it.

After the sub-window that shows the results of mutant EPN generation is closed by a test engineer, our tool shows only a final solution (that is, a chromosome with the highest fitness through all generations) on its main window shown in Fig. 5. Our tool allows a test engineer to modify the final solution, if needed. The modification function was already discussed in our previous study.[5]

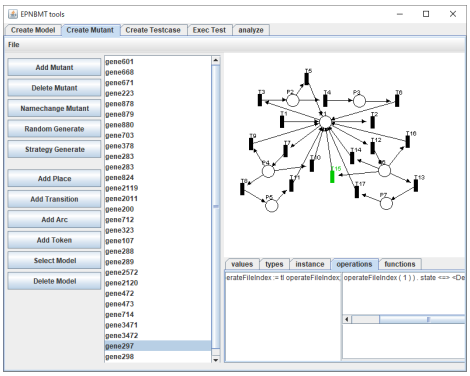


Fig. 5. Screen shot of the tool that shows a final solution.

## 4. Discussion

This section shows an application example, and gives the discussion about the effectiveness of our tool shown in the previous section.

In this study, we applied our tool to an original EPN of an OFMS (Online File Management System) that was discussed in Ref. 5. The environment to run our tool was a personal computer with i5-6500T processor (2.50 GHz, up to 3.10 GHz) and 4 GB RAM.

First, we developed a prototype of the OFMS based on the original EPN. The size of the prototype is about 830 lines of JavaScript code. We then selected three kinds of metrics (REET, MI, and RFTC) in order to calculate weights for transitions of the original EPN. As discussed in section 3.1, REET relates to usage distribution, and an access log file of Apache was gathered as its material. MI relates to complexity, and an output file from an existing source code analysis tool was gathered as its material. RFTC relates to test execution history, and log files of an existing test tool were gathered as its materials. We specified the materials on our tool, and determined the setting to execute automatic weight calculation. The effort of these manual operations on our tool was about 10 minutes, which will depend on the skill of test engineers. We then started the automatic weight calculation by our tool, and successfully got the original EPN with weights. The processing time for the calculation was about 0.9 seconds. Note that the development environment for the prototype of the OFMS is on a server, and our tool interacts with it. Therefore, the processing time will depend also on the performance of the server and network. The effort and the time to get the original EPN with weights seem to be acceptable for most of test engineers.

Subsequently, we generated mutant EPNs based on the original EPN with weights, by using our tool. The parameter setting for the GA-based technique to generate mutant EPNs was $N_c$=30, $N_g$=30, $p_m$=0.5, $p_c$=1.0, $G$=30, $imp_1$=0.3, $imp_2$=0.5, and $imp_3$=0.2 ($imp_1$, $imp_2$, and $imp_3$ are the degree of importance of REET, MI, and RFTC, respectively). Our tool executed the procedure of the GA-based technique, and successfully outputted 30 mutant EPNs as a final solution. The processing time for completing the procedure was about 3.4 seconds, and it seems to be acceptable for most of test engineers. The average quality (that is, average fitness) of candidate solutions in the first generation is about 0.59, and the quality of the final solution is about 0.94. The improvement of about 0.36 was achieved through the alternation of generations by the GA-based technique, and therefore our tool seems to be effective, at least in this case. We reviewed the outputted mutant EPNs, and found that:

- mutation operators were intensively applied to highly weighted transitions and their related elements (that is, parts that correspond to frequently used functions, complex functions, and/or frequently failed functions of the prototype),
- but some mutant EPNs included non-effective intended failures that would not be important or would not be made in actual development, and also, there were equivalent mutant EPNs.

An additional technique to select appropriate mutation operators based on the structure of an original EPN, and an additional technique to remove equivalent mutant EPNs need to be developed in order to make more improvements of the effectiveness.

## 5. Conclusion and Future Work

In order to improve EPN-based mutation testing, we introduced weights into an original EPN to evaluate the quality of mutant EPNs, and then we proposed a GA-based technique to generate high-quality mutant EPNs within reasonable time and cost. Moreover, we have developed a tool including the functions to calculate weights based on metrics and to execute the procedure of the GA-based technique. The results of applying the tool to a non-trivial example indicated its effectiveness, and the room for further improvements.

Our future work includes the development of additional techniques to select appropriate mutation operators based on the structure of an original EPN, and also to remove equivalent mutant EPNs.

## Acknowledgements

This work was supported by JSPS KAKENHI Grant Number JP26730038.

## References

1. J. Fitzgerald, P.G. Larsen, P. Mukherjee, N. Plat and M. Verhoef, *Validated Designs for Object-Oriented Systems*, (Springer-Verlag, London, 2005).
2. F. Belli, C.J. Budnik and W.E. Wong, Basic Operations for Generating Behavioral Mutants, in *Proc. of 2nd Workshop on Mutation Analysis in conjunction with ISSRE'06*, Nov. 2006, pp.9-18.
3. T. Takagi, R. Takata, Z. Furukawa, F. Belli and M. Beyazıt, Metrics for Model-Based Mutation Testing Based on Place/Transition Nets, in *Proc. of Joint Conf. of 21st Int. Workshop on Software Measurement and 6th Int. Conf. on Software Process and Product Measurement (IWSM-MENSURA)*, Nov. 2011, pp.7-10.
4. T. Takagi and T. Teramoto, Extended Mutation Score Based on Weighted Place/Transition Nets to Evaluate Test Suites, in *Proc. of 15th Int. Conf. on Computer and Information Science (ICIS)*, June 2016, pp.959-961.
5. T. Takagi, S. Morimoto and T. Katayama, Development of a Tool for Extended Place/Transition Net-Based Mutation Testing and Its Application Example, *Journal of Robotics, Networking and Artificial Life (JRNAL)*, Vol.4, No.2, Sept. 2017, pp.168-174.