# On Business Process Variants Generation

Asef Pourmasoumi[a,b], Mohsen Kahani[b], Ebrahim Bagheri[a], Mohsen Asadi[c]

[a] Department of Electrical and Computer Engineering, Ryerson University, Canada
[b] Web Technology Lab, Ferdowsi University of Mashhad, Iran
[c] SAP Canada, Vancouver Canada
a.pourmasoumi@ryerson.ca, kahani@um.ac.ir, bagheri@ryerson.ca, masadi@sfu.ca

**Abstract.** Cross-organizational mining is a new research field in the process mining domain, which focuses on the analysis and mining of processes in multiple organizations. Suitable access to collections of business process variants is necessary for researchers to evaluate their work in this research domain. To the best of our knowledge, no complete collection of process variants or any process variants/log generator tool exists for this purpose. In this paper, we propose an algorithm for generating random process variants for a given process model and a supporting toolset built on top of the PLG toolset. For this purpose, we classify different factors that can serve as variation points. Then, using the structure tree based representation of an input process, we present an algorithm for applying variation points based on a user-defined variation rate. The developed tool is publicly available for researchers to use.

**Keywords:** process model variants, process variant generator, variation point, Process structure tree

## 1 Introduction

Peer-organizations such as municipalities, hospitals and universities often employ many different variations of the same business processes. For instance, Suncorp is a famous Australian insurance company, which has over 6000 business process variants [1]. These processes have many commonalities and some degree of variability. Mining and analysis of such process variants can result in insights that can improve organization operations [2]. Based on the literature [3] [4] [5], variants are defined as process models, which follow the same goals but have slight structural differences, i.e. they have at least one feature in common and one feature in which they differ.

Cross-organizational mining encompasses different research branches: reference model extraction, process models similarity calculation, process model merging, process fragmentation, among others. In all of these research areas, there is dire need for collections of process variants and/or execution logs. Unfortunately there are not enough standard datasets of process variants or variants executions logs. The only available dataset is a small collection of process variants log in the BPI Challenge2014, but it just contains log files of five variants of a process model from the

CoSeLOG1 project. The lack of public data can be attributed to disinclination of organizations to publish their own data. So, simulation and process synthesis tools can serve as a viable alternative for researchers for the evaluation of their techniques.

In this paper, we present an approach and toolset for generating process variants. We provide the following contributions:

- First, we classify the effective factors for creating process variants. These factors can be used as a reference for proposing new algorithms and tools for process variant generation.
- Second, we propose an algorithm based on the structure tree representation of input process models for creating variation points on an input process model. Using this algorithm, a collection of process variants can be created randomly according to a probabilistic distribution and based on a user-defined variation rate parameter.

## 2 Related Works

Cross-organizational mining is a young research field in the process mining domain [6]. The prerequisite for research in this sub-domain is having appropriate process variants. Large collections of process variants enable researchers to comprehensively evaluate their work. In the past, there have been several works on random process generation. In [7], the PLG (process log generator) tool for generating random block-structured process models and their executions is presented. In the most of references, block-structured process models require that each control flow split has a corresponding join of the same types [8][9]. This tool is open source and has a plug-in for the ProM framework. PLG generates random process models using context-free grammars by employing 5 basic workflow patterns: single activity, loop, sequence, XOR split-join and AND split-join. Moreover, the user can select from three probability distribution functions: Uniform, Gaussian and Beta, which will be used for generating the number of branches for AND/XOR split-join patterns. After generating a random process model, PLG is capable of generating its execution logs by traversing the generated process graph. PLG is a great framework for generating large scale random process models and event logs. Hence, we used it as the basis for developing our process variants generator tool.

There are also other tools for generating process models [10] [11]. In [10], CPN toolset is proposed. It has a powerful GUI and users can easily edit process models. An extension of CPN is capable of generating event logs. However it uses a rare language for writing scripts which makes it difficult for development.

Shugurov et al. [12] propose an approach for generating a set of event logs with noise which is implemented as a plug-in for ProM. Their basic idea is to use token replay in Petri Net process models for log generation. They add noise using three ways: adding artificial transitions, adding existent transitions in incorrect order and by skipping events.

| Groups | Function | Description |
|---|---|---|
| Operators Change Functions | $\Phi_{O_1 \mapsto O_2}$ | This function converts a specified operator $O_1$ to operator $O_2$. The operators $O_1$ and $O_2$ can be: AND, XOR, OR, Sequence. |
| | $\Phi_{Delete}(O)$ | This function deletes specified operator O. |
| | $\Phi_{Move}(O, A, B)$ | This function moves a specified operator O to a new places between node A and B. |
| | $\Phi_{Add}(O, A, Childs)$ | This function add a specified operator O before A and connects it to the list of Childs. |
| Activities Change Functions | $\Delta_{Add}(A, R_A)$ | This function inserts a new activity A with relation $R_A$ to process. |
| | $\Delta_{Delete}(A)$ | This function removes the activity A. |
| | $\Delta_{Swap}(A, B)$ | This function swaps the activity A with the activity B. |
| | $\Delta_{Move}(A, C, D)$ | This function moves the activity A to a new place between C and D. |
| Connections Change Functions | $\Gamma_{Add}(L_{AB})$ | This function adds a new sequence link from A to B. |
| | $\Gamma_{Remove}(L_{AB})$ | This function delete a sequence A to B link. |

The only work that we have encountered for generating process variants is proposed in [11]. Stocker et al. proposed SecSy for generating a set of event logs with some deviation from the original model.
The SecSy tool generates event logs for the sake of evaluating of business process security monitoring and auditing and is useful for security-oriented information systems [12]. One of the drawbacks of SecSy is that it does not cover many possible deviation patterns. It uses three transformations for making a deviation: Converting AND to XOR, XOR to AND and swapping the order of two activities, while there can be several other forms of transformation which will be explained in this paper. Also, in the SecSy tool, the user is not able to determine the degree of variation.

## 3 Proposed Approach

For the sake of clarity, we divide this section into three sub-sections; Section 3.1 introduces the various factors that can generate a process variant. These factors are classified into groups and can be used as a reference for proposing new algorithms for process variant generation. In Section 3.2, a structure tree based representation of BPMN process models is shown. In Section 3.3, we use this tree-based representation for extracting process variants. The proposed algorithm for extracting variants is described precisely in this section. Also, the probability distribution functions that have been used for random process variant generation is described in Section 3.3.

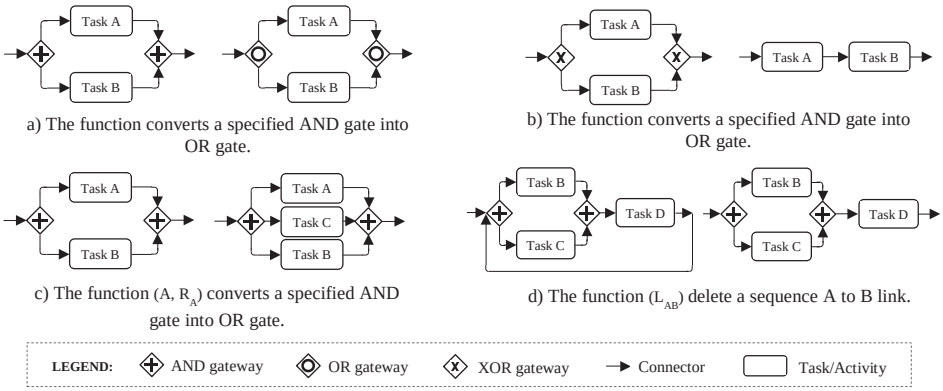



**Fig. 1.** BPMN representation of some functions described in Table 1

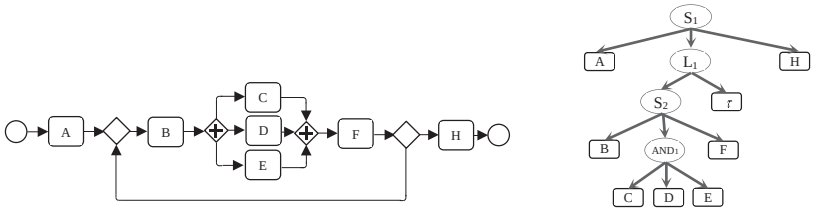
### 3.1 Factors for Variant Generation

In Table 1, we show the list of various functions that could result in process variants. This list is not intended to be comprehensive. The list is classified into three groups: *i*) Operator change functions, *ii*) Activity change functions and *iii*) Connection change functions. In the first class, the functions that lead to a variant using a change in operators of input process model are described. For example, $\Phi_{AND \mapsto OR}$ converts a given specified AND gateway to OR gateway and leads to a new process variant. As another example, the function $\Phi_{S \mapsto XOR}$ converts a given number of activities that have sequence relation and places an XOR gateway between them. The second class of functions changes the activities of the main process model for generating process variants. For example, the function $\Delta_{Insert}$ (A, $R_A$) inserts a new activity *A*, which did not exist in the main process model. The argument $R_A$ determines the relations that *A* would have with other activities. Finally, the functions in the third class change the links that are in the main process model and lead to new process variants. For example, the function $\Gamma_{Remove(L_{AB})}$ deletes a specified link from activity *A* to activity *B*.

In Figure 1, the effects of some functions on an input process model are shown as an example. For the sake of simplicity, the process model is shown in BPMN format. It is clear that these functions have different magnitude of changes on the input process model.

### 3.2 Structure Tree Representation of Block-Structured Process Models

For implementing the functions in Table 1, in the first place, a representation for process models should be selected. In this paper, we use the structure tree representation of process models [13]. In selecting suitable representation, we consider two points: *i)* that the change functions in Table 1 can be applied directly, *ii)* that the representation can support block-structured process models. The reason for focusing on




a) An example BPMN process model b) Structure tree of process model (a)

**Fig. 2.** An example of the structure tree ($S_i$ shows the sequence $i$ and $L_i$ shows Loop $i$)

block-structured processes is twofold: 1) In [13], translation from the widely used process modeling notations such as BPMN and Petri Net to structure tree and vice versa has been shown. 2) It is shown in [14] that about 95% of process models are block-structured or can be converted to an equivalent block-structured process.

In [14], a structure tree is defined as follows:

***Definition 1*** [14]: A process structure tree is a tuple T = (*N*, *C*, *E; L*) where:

- *N* is a set of leaf nodes representing activities.
- *C* is a set of connector nodes including AND, OR, XOR, Sequence, and Loop.
- $E \subseteq (C \times C) \cup (C \times N)$ is a set of edges.

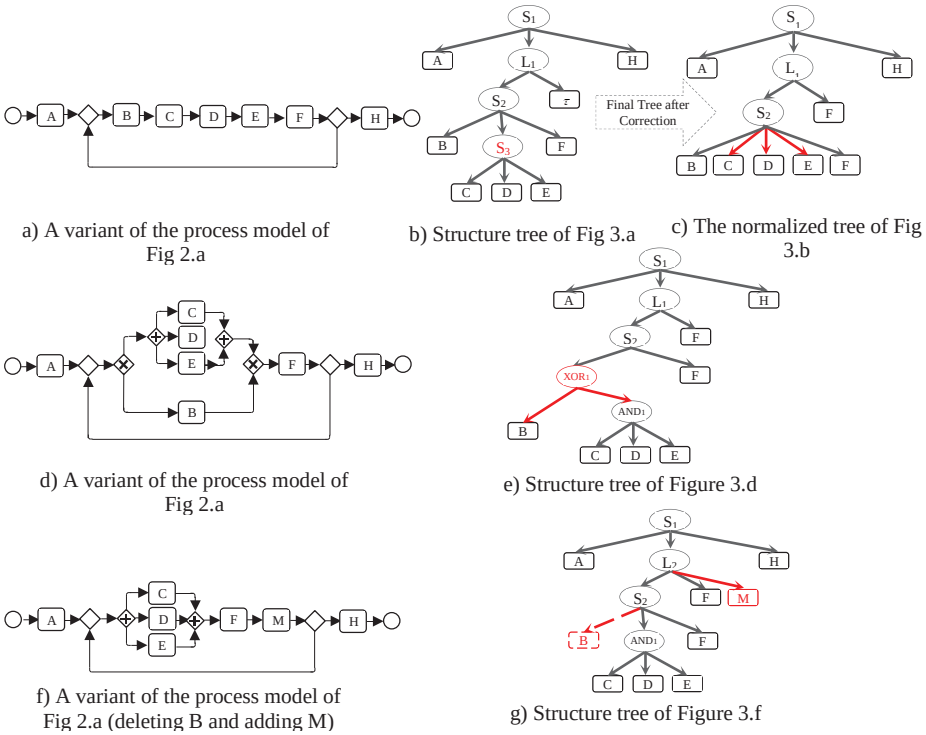
In Figure 2, an example of a block-structure process model and its corresponding structure tree is shown. In the structure tree, each intermediate node shows a process block and the tree is parsed from left to right. The intermediate nodes include AND-blocks, XOR-blocks, OR-blocks, Loops and Sequences corresponding to different patterns in process models. The leaf nodes in the tree correspond to activities in the process model.

Another reason for choosing structure tree as the representation form is that it can clearly show the blocks of a process model and their relationships. This would guarantee the soundness of the created variant after the change functions are applied. Since the changes are performed randomly and sequentially, so every permissible change applied on the process tree should keep the soundness of the process model. Moreover, every change in Table1 can be mapped to a change or a set of changes on the structure tree.

### 3.3 Generating Process Variants

The core of our idea for is to develop a mapping between the functions in Table 1 and change operations in the process structure tree. In other words, we intend to elicit the process variants by converting the structure tree of an input process model into other valid structure trees using tree conversion operations. These conversion operations will be selected based on the rate of variability, which the user specifies and through the mappings.

The operator change function ( $\Phi_{O_1 \mapsto O_2}$ ) can be performed by changing the type of the connectors. For example, Figure 3.a shows a variation of the example in Figure




a) A variant of the process model of Fig 2.a

b) Structure tree of Fig 3.a

c) The normalized tree of Fig 3.b

d) A variant of the process model of Fig 2.a

e) Structure tree of Figure 3.d

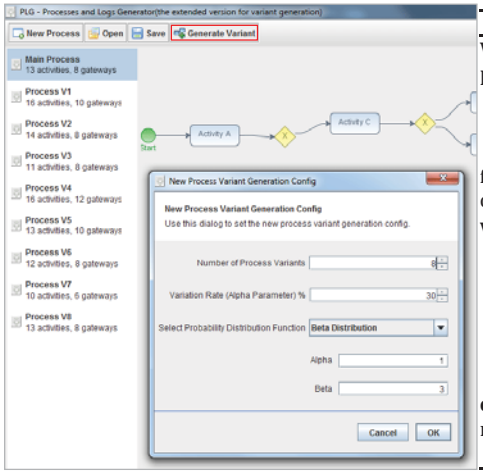
f) A variant of the process model of Fig 2.a (deleting B and adding M)

g) Structure tree of Figure 3.f

**Fig. 3.** Examples of mapping between change operations in the structure tree of Figure 2 and the functions in Table 1.

2.a. In this variation, AND gateway which sits between C, D and E has been changed to a sequence between C, D and E using the $\Phi_{AND_1 \mapsto S}$ function. The corresponding changes that need to be applied on the structure tree in order to enact this change are distinguished with red color. After every change, the modified tree would be checked and if there are any connector nodes that have at least one child of the same type (except for loop connector), the child node will be removed and its children will be connected to the parent.

The function $\Phi_{Delete}(O)$ can be implemented by removing the corresponding connector node and all of its children. The function $\Phi_{Move}(O, A, B)$ moves the block of operator O between A and B. For mapping this function on the structure tree, the parent of A would be checked. When the parent of A is OR/XOR/AND/Sequence, the subtree of O would become the right sibling of A. If the parent of A is a loop, given moving subtree of O after A requires the creation of a new connector node, we will do this by using the $\Phi_{Add}(O, A, Childs)$ change function. This function adds a new operator after A and connects it to the 'Childs' nodes. For mapping this function on the structure tree, we look at the children of A. If A has more than one child, we randomly select *n* consecutive children of A (where *n* is less than the number of children of A). Then A is connected to the newly generated operator O whereby O becomes the parent of the *n* selected children of A. It is also possible to add a new operator O with a new activity




**Algorithm1.** The pseudo code of proposed approach

**VariantGenerator**($T$, $\alpha$, $d$, $p$)
**Input**: $T$ is structure tree representation
$\alpha$ is variation rate,
$d$ is probability distribution function,
$p$ the parameter of selected distribution function
originalTree = T;
**while**(true) **do**
  f = selectVariationFunctionByProbabilityDF (d, p);
  **if**(treeBaseDistance(T,originalTree)> $\alpha$ ) **then**
    break;
  **end**
  T = performChangeOnTree(T, f);
  T = correctTree(T);
**end**
return T;

a) A screen shot of proposed tool. b) The algorithm of proposed approach

**Fig. 4.** A screen shot and algorithm of the proposed tool

child. In our tree-based mapping, all of these functions are necessary, and none of them can be implemented by other operation change functions.

The next group are activity change functions. The function $\Delta_{Add}(A, R_A)$ creates a new activity and connects it to existing connector nodes (adding a new activity to a new operator can be done using the $\Phi_{Add}(O,A,Childs)$ function). For mapping this function, there is need for creating a new activity and adding it randomly to existing connectors (Figure 3.f). The function $\Delta_{Delete}(A)$ removes activity A from the process model. It can be mapped to the tree be removing leaf node A. After removing activity A, its parent would be checked. If its parent is OR/XOR/AND/Sequence connector and has only one child, then its parent will be removed and the child is connected to its grandparents (Figure 3.g). The function $\Delta_{Move}(A, C, D)$ is used for moving an activity within a block or from one block to another block. This can be mapped in a tree by moving a leaf node between its siblings (when its parent is Sequence connector) or by changing its parent (when its parent is OR/XOR/AND). The function $\Delta_{Swap}(A, B)$ can be calculated using $\Delta_{Move}$. Since our processes are sound and block-structured, we map $\Gamma_{Add}(L_{AB})$ and $\Gamma_{Remove}(L_{AB})$ functions just for adding and removing loops.

### 3.4 Selecting Generated Variation Based On The Variation Rate

The variation rate is a parameter that specifies the degree of permitted deviation from the input process model. We consider $\alpha$ as a variation rate, which will be determined by the user. In Table 1, various functions for variation creation have been listed. Each of these functions has a different effect on the input process model, but since these change functions are selected randomly and some of them may affect the previous changes (e.g. executing $\Delta_{Swap}(A, B)$ and $\Delta_{Swap}(B, A)$ does not make any change on the

input process), so in our tool, we exploit process similarity as introduced in [15] to measure degree of change. Pawlik and Augsten have introduced a tree-edit distance whereby the similarity of two trees is calculated based on the minimum number of operations that is needed to change one tree to another. In our work, we use the tree-edit distance for generating process variants such that the functions in Table 1 are chosen in a way that the amount of variation is less than $\alpha$. So, after executing each change, the distance of the generated variation from the input process is calculated. If the distance is less than $\alpha$, another round of changes can be applied as long as the distance between the generated variant and the input process stays less than $\alpha$. The pseudo code of the proposed algorithm is showed in Figure 5.b.
The change functions are selected randomly based on a probability distribution function. In the proposed tool, the user can select different probability distribution functions such as Guassian, Uniform, Beta and Gamma functions. Based on the selected distribution function and the user-defined variation rate, the variation functions and the corresponding change operations in structure tree would be applied.

In Figure 5.a, a screenshot of the proposed tool is shown. As explained earlier, we have built our tool on top of the PLG toolset. We have added a new option to the last version of PLG2 for generating new variants (it is marked in red in Figure 5.a). The user needs to set the number of variants that is desired. The variation rate should be set between 1 to 100 percent (it is set by default to 30%). Also, the user can define the probability distribution function for generating variants randomly. The generated variants can be selected and viewed in the left pane of the tool. In the future, we intend to further develop our tool as a plug-in for the ProM framework.

## 4 Conclusion

In this paper, we introduced a toolset for generating collections of business process variants according to a user-defined variation rate. We defined and classified various factors that can lead to the generation of a variant of an input process model based on which change functions can be defined. Then, we proposed an algorithm based on the structure-tree representation of input process models for applying these change functions. These functions would be performed based on different probability distribution functions and with respect to a user defined variation rate. Our toolset is implemented in PLG and is accessible to researchers.

## References


1. Larosa, M., Dumas ,M., Uba ,R,.and Dijkman ,R. M.: Business Process Variability Modeling: A Survey .ACM Transactions on Software Engineering Methodology 22,2,11, 2013.
2. Larosa, M., Dumas, M., Uba, R.,and Dijkman, R. M., Business Process Model Merging: An Approach to Business Process Consolidation. ACM Transactions on Software Engineering Methodology 22,2,11, 2013.

3. Pourmasoumi, A.; Kahani, M.; Bagheri, E. and Asadi, M. Mining Common Morphological Fragments from Process Event Logs. In Proceedings of the 2014 Conference of the Centre for Advanced Studies on Collaborative Research, 2014.
4. Valenca, G., Alves, C., Alves, V., and Niu, N. A Systematic Mapping Study on Business Process Variability. Int. Journal of Computer Science & Information Technology 5,1, 2013.
5. Reichert, C. Li, M., and Wombacher, A., "The MINADEPT Clustering Approach for Discovering Reference Process Models out of Process Variants," International Journal of Cooperative Information Systems, vol. 19, no. 3-4, pp. 159–203, 2010.
6. Li, C. and Reichert, M. and Wombacher, A.: Mining Business Process Variants: Challenges, Scenarios, Algorithms. Data & Knowledge Engineering, 70(5), pp. 409-434, Elsevier, 2011.
7. Burattin, A., and Sperduti, A., PLG: A Framework for the Generation of Business Process Models and Their Execution Logs. In Business Process Management Workshops, pages 214–219. Springer, 2010.
8. Buijs, J. C. A. M., van Dongen, B. F., van der Aalst, Wil M. P., "Discovering and Navigating a Collection of Process Models using Multiple Quality Dimensions", in Proceedings of the 9th International Workshop on Business Process Intelligence, 2013.
9. Weske, M., Business Process Management: Concepts, Languages, Architectures. Springer-Verlag, Berlin, 2007.
10. Medeiros, A. K. A. d., and G¨unther, C. W., Process mining: Using CPN tools to Create Test Logs for Mining Algorithms, in Proceedings of the Sixth Workshop on the Practical Use of Coloured Petri Nets and CPN Tools, vol. 576. Aarhus, Denmark, 2005.
11. Stocker, T., and Accorsi, R., "Secsy: Security-aware Synthesis of Process Event Logs," in Proceedings of the 5th International Workshop on Enterprise Modelling and Information Systems Architectures, St. Gallen, Switzerland, 2013.
12. Shugurov, I., Alexey A. Mitsyuk. Generation of a Set of Event Logs with Noise, Proceedings of the 8th Spring/Summer Young Researchers' Colloquium on Software Engineering, M.: 2014. P. 88-95, 2014.
13. J. C. A. M. Buijs, Flexible Evolutionary Algorithms for Mining Structured Process Models, Ph.D thesis, 2014.
14. Li, C., Mining process model variants: challenges, techniques, examples. Phd thesis. University of Twente, The Netherlands, 2010.
15. Pawlik, M., Augsten, N.: RTED: A Robust Algorithm for the Tree Edit Distance. CoRR, abs/1201.0230, 2012.