Received: June 17, 2020. Revised: July 17, 2020.

International Journal of
Intelligent Engineering & Systems

http://www.inass.org/

A New Similarity Method based on Weighted-Linear Temporal Logic Tree and Weighted Directed Acyclic Graph for Graph-based Business Process Models

Khairiyyah Nur Aisyah[1] **Kelly R. Sungkono**[2] **Riyanarto Sarno**[2]*

[1]*Department of Informatics Engineering, Institut Teknologi Sepuluh Nopember, Indonesia*
* Corresponding author's Email: riyanarto@if.its.ac.id

Abstract: A business process is a set of activities that needs to be considered in organizations or companies. Linear temporal logic (LTL) can models relationships of activities; however, the existing LTL does not consider occurrences probability of relationships of activities based on the event log. Weighted Linear Temporal Logic (W-LTL) extends the existing LTL by giving weights based on the occurrences probabilities. This paper proposes a new similarity method that combines Weighted-Linear Temporal Logic (W-LTL) Tree and Weighted Directed Acyclic Graph (wDAG) that modifies the original wDAG similarity, so it can distinguish the similarity value of two wDAGs that have two branches with opposite weight values. The proposed method (W-LTLDAG) will be verified by comparing with the original wDAG similarity, TPED, Cosine-TDP, and WGED. Based on the comparison, wDAG and WGED gives similarity value of 1 for all experiments, shows that both cannot distinguish weight between 2 graphs. TPED only concerns on relation without giving regards to the number of traces, Cosine-TDP and proposed method are able to distinguish parallel relations that have different occurrence probability of activity relations, but proposed method is proven to give a better calculation by giving a high similarity value, 0.976 for graphs with a small difference value of weights between branches, and low similarity value, 0.327 for graphs with a large difference value of weights between branches.

Keywords: Business process management, Graph database, Linear temporal logic, Similarity method, Weighted directed acyclic graph.

1. Introduction

A business process is a set of activities in a company that describes their logical order and the dependences between activities to produce the desired results [1, 2]. For this purpose, process models are widely utilized, addressing various issues such as fraud [3, 4], economy [5], and environmental problems [6]. The construction of a process model makes analysing these processes more easy [7]. The information in a business process can be analysed through a graph database, which is have seen growing popularity over the last few years [8]. Information inside a graph database can be represented in the form of a process model. Business process models can be represented in various ways, using for example BPMN, Petri net, or a graph [9, 10]. Nowadays, organizations or companies need to process hundreds or even thousands of process models in its repository. Organizing these repositories requires an efficient and effective methods to facilitate the business process analysis [11]. Doing a manual check on hundreds of business process models requires an enormous of effort and results in high costs [12]. Several techniques can be used for the reconstruction of the business processes quickly, such as process recommendation, process clustering, and process query which are all based on the business process similarity [13]. Calculation of similarity on process models being an important thing to do in business process management such as preventing duplication [14], reduce costs in expanding business [15], identified processes that no longer comply with the company [16], model repository management [17], and many more.

As mentioned above, many similarity measurements of business processes have been developed in recent years. Previous research [18] proposed a weighted DAG (wDAG) similarity algorithm for match-making in e-Business environments. The wDAG similarity algorithm compared the similarity calculation between 2 arc-weighted DAGs. [19] proposed Tree Declarative Pattern Edit Distance (TPED) and Cosine-Tree Declarative Pattern Similarity (Cosine-TDP). TPED is a modification of Graph Edit Distance (GED) [15] to measure a structural similarity, while Cosine-TDP is a modification of an original cosine to measure a behavioural similarity. Another previous work named Weighted Graph Edit Distance (WGED) [13] aimed to calculate minimal costs of transforming a graph to the other. The transformation is based on node substitution, node insertion/deletion, and edge insertion/deletion. However, most of previous reseraches are not paying attention in distinguish different parallel relations or same parallel relations that have different occurrence probability of activity relations.

This paper proposed a new similarity method based on Weighted-Linear Temporal Logic (W-LTL) Tree and Weighted Directed Acyclic Graph (wDAG). A previous research [20] developed a tree model based on linear temporal logic (LTL) patterns from an ordinary graph database without weight value. In this study, a weight is given to the tree on each relation between activities. The weights are obtained based on the probability of the occurrence of activities with the number of its incoming or outgoing activities. The major contributions of this paper are as follows:

1) Proposed an algorithm to discover Weighted-Linear Temporal Logic (W-LTL) Tree patterns from an original event log.
2) Extends the existing LTL by giving weights based on the occurrences probability of relationships of activities based on the event log.
3) Proposed a new similarity method based on Weighted-Linear Temporal Logic (W-LTL) Tree and Weighted Directed Acyclic Graph (wDAG) that able to distinguish different parallel relations or same parallel relations that have different occurrence probability of activity relations.

The proposed similarity method then will be compared with the existing similarity methods, i.e. wDAG algorithm, Tree Declarative Pattern Edit Distance (TPED), Cosine-Tree Declarative Pattern (Cosine-TDP), and Weighted Graph Edit Distance (WGED). Each method will be used to calculate similarity between graphs with parallel relations, XOR, OR, and AND. Similarity between graphs with the same relation but in different weights is also compared.

This rest of this research consists of several sections: Section 2 presents the basic concepts which underlie this research. The proposed method is discussed in Section 3. The experiment results will be discussed in Section 4, and the conclusion is presented in Section 5.

2. Research method

2.1 Parallel relationship of process model

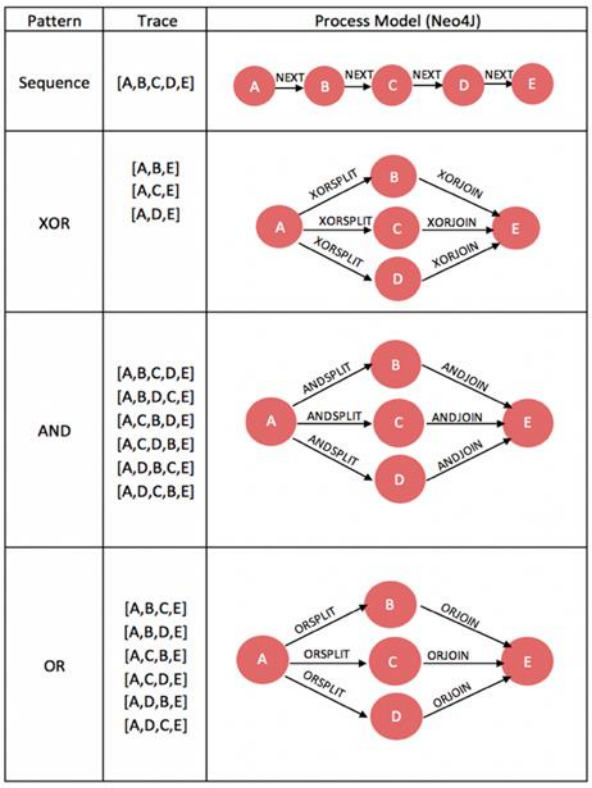
A great process model is a process model that has no redundant activity and the behaviour of its activities is clearly visible. Both of them can be applied with control flow patterns and concurrency. Control- flow patterns are used to build relations between activities, i.e. Sequence, XOR, AND, and OR [21]. Sequence is a relation that connects one activity to another. XOR occurs when an activity in the process model has branches and only one activity is executed. The OR relation is used when an activity has branches and several branches must be executed. AND is a relation that is used when an activity has several branches and all branches must be executed. If the chosen activity in XOR, OR and AND relations is the previous activity of the other activities, then ‘Split’ will be given in the control- flow pattern of the relation. If the chosen activity in the XOR, OR and AND relations is the next activity of the other activities, then ‘Join’ will be given in the control-flow pattern of the relation [21].

Examples of Sequence, XOR, OR, and AND relations are shown in Fig. 1. The first column of Fig. 1 describes the relationships between activities, the second column describes the log, while the third column represents the form of the graph model.

2.2 Graph model

A graph model is a database that consists of a state of graphs [22]. Graph model is a representation of the graph database that already contains one or several relations such as Sequence, XOR, OR, or AND. Graph models are used to solve problems that cannot be handled using tabular databases, such as data that have too many relationships. Handling data with a large number of relations requires complex SQL queries. Graph formation makes it easier for users to see the structure of the relations between activities in a process model. There are multiple places in information systems where data about process execution can be stored, i.e. system logs, databases, text files, and many more [23]. A graph

Received: June 17, 2020. Revised: July 17, 2020.



Pattern	Trace	Process Model (Neo4J)
Sequence	[A,B,C,D,E]	A -NEXT→ B -NEXT→ C -NEXT→ D -NEXT→ E
XOR	[A,B,E] [A,C,E] [A,D,E]	A -XORSPLIT→ B, C, D -XORJOIN→ E
AND	[A,B,C,D,E] [A,B,D,C,E] [A,C,B,D,E] [A,C,D,B,E] [A,D,B,C,E] [A,D,C,B,E]	A -ANDSPLIT→ B, C, D -ANDJOIN→ E
OR	[A,B,C,E] [A,B,D,E] [A,C,B,E] [A,C,D,E] [A,D,B,E] [A,D,C,E]	A -ORSPLIT→ B, C, D -ORJOIN→ E

Figure. 1 Process discovery

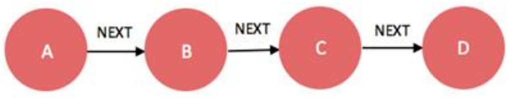


Figure. 2 Example of graph model

model has two components, namely nodes and arcs. Nodes are points that contain information about name of the activity, while arcs are lines that show the relationships between nodes. In the graph model of Fig. 2, Symbols A until D are nodes, while next relations are arcs.

2.3 Linear temporal logic tree model

Linear temporal logic (LTL) is a formal language that describes several temporal logics that refer to time [20]. Meanwhile, LTL tree model is a representation of a business process that is discovered based on patterns in linear temporal logic. A previous research [20] discovered control-flow patterns based on a declarative model. Each discovered linear temporal logic pattern is split into activities and symbols that are used in discovering the LTL tree. The form of the resulting Linear Temporal Logic patterns based on [20] is shown in Table 1. The first column describe the relationships between activities, meanwhile the second column shows the form of the relation in LTL.

Table 1. LTL patterns

Relation	Pattern
Sequence	LTL : act - > O (y)
AND Split	LTL : act - > <> ((y1 /\ y2 … /\ yn))
AND Join	LTL : <> ((y1 /\ y2 … /\ yn)) - > O (act)
XOR Split	LTL : act - > O ((y1 \/ y2 … \/ yn))
XOR Join	LTL : O ((y1 \/ y2 … \/ yn)) - > O (act)
OR Split	LTL : act - > <> ((y1 \/ y2 … \/ yn))
OR Join	LTL : <> ((y1 \/ y2 … \/ yn)) - > O (act)

2.4 Weighted-linear temporal logic (W-LTL)

Each relation between activities in the Linear Temporal Logic patterns is given a weight value. The value is obtained from the occurrence frequency of a sequence of activities in the process model. In Weighted-Linear Temporal Logic (W-LTL) model, ‘p’ is added to represent the weight value of the relation between two activities.

2.5 Similarity calculation using weighted Directed acyclic graph (wDAG)

Several previous research proposed a similarity calculation based on weight, for example AgentMatcher [24] and weighted Directed Acyclic Graphs (wDAG) [18]. Generally speaking, the wDAG similarity algorithm is used to traverse the two wDAGs (a and a') with a depth-first strategy. It calculates their similarity bottom-up [18]. The basis of the recursion is that if two nodes (n and n') are leaf nodes and they are identical, then the similarity is 1.0. Otherwise, the similarity is 0.0. The similarity calculation between non-leaf nodes is done by summing all similarity values of their sub-wDAGs and then multiplying the result by the average value of their arc weights. The result of the similarity is in the interval [0,1]. Generally speaking, the calculation of similarity of wDAG is shown in Eq. (1).

$wDAGsim(a, a')$ is similarity of 2 wDAGs a and a'

$wDAGsim(a_i, a_j)$: Intermediate similarity of i^{th} and j^{th} sub-wDAGs.

w_i and w'_j: Arc weights of the i^{th} and j^{th} of the root node.

$\in$ means an empty wDAG, i is increase from 1 to the breadth of a, and j is increase from 1 to the breadth of a’.

In some cases, Eq. (1) cannot distinguish the similarity value of two wDAGs that have two branches with opposite weight values. For example, there are 2 wDAGs, A and B, with opposite values of their branches, as shown as Fig. 3. Using Eq. (1), the similarity value of wDAG A and wDAG B of Fig. 3. is shown as below:

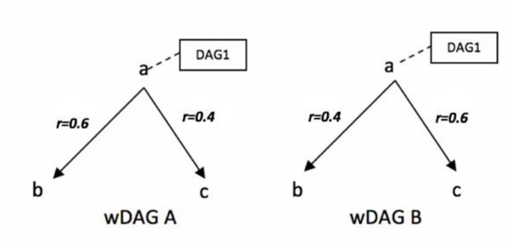


Figure. 3 wDAG A and wDAG B

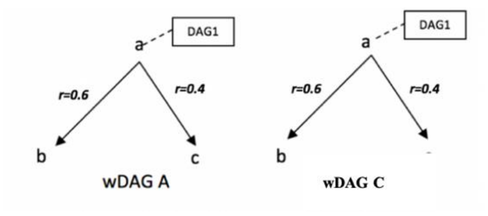


Figure. 4 wDAG A and wDAG C

$$wDAGsim(a,a') = \begin{cases} 0.0, \text{the root node labels of a and a}' \text{ are not identical} \\ 1.0, \text{a and a}' \text{are leaf nodes} \\ \sum \begin{cases} wDAGsim(a_i, a_j).\frac{w_i+w'_j}{2}, \\ \text{a}_\text{i} \text{ and a}'_\text{j} \text{ not missing} \\ wDAGsim(a_i, \in).\frac{w_i+0}{2}, \\ \text{a}_\text{i} \text{ is missing in a}' \\ wDAGsim(\in, a'_j).\frac{0+w'_j}{2}, \\ \text{a}'_\text{j} \text{ is missing in a} \end{cases} \\ \sum_{j=1}^{breadth_{of'_a}} wDAGsim(\in, a'_j).\frac{0+w'_j}{2}, \\ \text{a is a leaf node} \\ \sum_{i=1}^{breadth_{of_a}} wDAGsim(a_i, \in).\frac{w_i+0}{2}, \\ \text{a}' \text{ is a leaf node} \end{cases} \quad (1)$$

$$wDAGsim(A.DAG1, B.DAG1) = \left(\frac{0.6+0.4}{2}\right) \times 1.0 + \left(\frac{0.4+0.6}{2}\right) \times 1.0 = 1$$

In another case, two wDAGs A and C have the same branches with the same weight values, as shown in Fig. 4. The similarity of wDAG A and wDAG C of Fig. 4 is:

$$wDAGsim(A.DAG1, C.DAG1) = \left(\frac{0.6+0.6}{2}\right) \times 1.0 + \left(\frac{0.4+0.4}{2}\right) \times 1.0 = 1$$

Using Eq. (1), the similarity between wDAG A, B and wDAG A,C are same. To distinguish between the similarity of two wDAGs with two branches that have opposite weight values, a new similarity method is needed, which will be explained in the next section.

3. Proposed method

This section presents an algorithm to discover the W-LTL tree of a graph-based business process model and the new similarity method based on W-LTL Tree and wDAG. The first step in discovering the W-LTL tree is to convert the information from the execution process in the graph database into a graph model. The relations between the activities are used to discover W-LTL patterns. Each component in the W-LTL patterns is used to discover the W-LTL tree model. In this study, LTL is used because it uses text so that it requires less storage than other process models. Whereas W-LTL is applied so that it can be used in calculating similarity. In this study, the data used always begin and end with a single event, no overlapping activities, and no repetitive activities (looping).

3.1 Discovering a graph model based on a graph database

The data from the execution of a retail business process are stored in a graph database in .csv format. The first step is to convert the information in the graph database to a graph model. First, the .csv file is imported into Neo4j Graph Platform. Neo4j is a network-oriented database that stores data structured in networks rather than tables [25]. The basic data model in Neo4j consists of nodes, relations, and attributes. Nodes are similar to object instances and are connected by various relationships. The relations used in this research are Sequence, XOR, OR, and AND relation.

All the information of each Case Id in the graph database will be merged as an activity. In this step, all activities that have been discovered in the previous step are merged as case activities. After the case activities have been discovered, the next step is to specify the relations between the activities. The notation 'Split' will be added to activities that have branches, while the notation 'Join' indicates activities that were split previously. The Sequence relation is used if the number of outgoing values from an activity is 1. The AND Split relation is used if there is more than 1 outgoing value from an activity and all branches are executed. AND Join is used to reunite branches in AND Split relations. XOR Split relations are formed if there is more than 1 outgoing value from an activity and only 1 branch is executed. XOR Join is used to reunite branches in XOR Split relations. OR Split is a relationship that is formed if an activity has 2 or more outgoing values and only some branches are executed. OR Join is used to reunite branches in OR Split relations. The nodes and relations formed from the graph model are exported in the form of a .csv file. This file is then used in discovering the W-LTL patterns, which will be explained in the next section.

3.2 Discovering weighted-linear temporal logic (W-LTL) patterns from a graph model

To discover W-LTL patterns, two types of .csv files are needed. The first .csv file contains the nodes and relations that were exported in the previous step. The second file only contains the sequence relations between the nodes, which will be used in calculating the weight of each relation of the nodes. The two files are processed using the Python programming language to discover the W-LTL patterns. A step that needs to be done first is to calculate the weight of the relations of the nodes. The weight is obtained based on the probability of occurrence of an activity sequence. The calculation of the weight is done in two ways. The first pays attention to the outgoing value of an activity, while the second pays attention to the incoming value of the activity. The outgoing value is used in calculating the weight of Sequence or Split relations, while the incoming value is used in calculating the weight of Join relations. An example of calculating the weight of a relation between activities is shown in Table 2.

Hence, before discovering W-LTL, the symbols used in Table 2 that are not registered in ASCII are converted to ASCII symbols. The converted symbols are shown in Table 3. The symbols in the second column are the original symbols. All the symbols in column 2 are converted to symbols in column 3.

The weight values for each sequence relation of the activities are then used for discovering the W-LTL patterns. The symbols used in discovering the W-LTL patterns are registered in ASCII (American Standard Code for Information Interchange), an international standard for writing letters and symbols. The .csv file that contains the nodes and their relations that have been exported in the previous step is then used for discovering the W-LTL patterns. Each data in that file will be converted to Weighted-Linear Temporal Logic. The algorithm used to discover the W-LTL patterns is shown in Table 4. The relations discovered in this W-LTL are NEXT, XORSPLIT, XORJOIN, ORSPLIT, ORJOIN, ANDSPLIT, and ANDJOIN. For example, act A -> _O (act Bp = 1) means B is the next activity of A with "NEXT" relation and weight of A-B is 1. act A - > _O ((act Bp= weightA, B ⋁ … ⋁ act np= weightA, n)) means B until n are outgoing activities of A, with Bp= weightA, B means the weight value of activity A-B is weightA, B and np= weightA, n means the weight value of activity A-n is weightA, n. The complexity

Table 2. Example of calculating weight

Relation	Trace	Weight
Sequence	ABCD -5x	Target : A-B = 5 Weight of A-B = $\frac{5}{5} = 1$
Split	ABCD-3x ACBD-2x ADBC-4x	Target : A-B = 3 outgoing A = (3+2+4) = 9 weight A-B $\frac{3}{9} = 0.33$
Join	ABCD-3x ACBD-2x ADBC-4x	Target : C-D = 3 Incoming D = (3+2) = 5 Weight C-D = $\frac{3}{5} = 0.6$

Table 3. Conversion of symbols to ASCII

No	Symbol	Converted Symbol
1.	<>	< >
2.	∧	/ \
3.	∨	\ /
4.	O	_O

of the algorithm to discover W-LTL is O(n3). The general form of the W-LTL patterns is shown in Table 5. y1, y2, …, yn are the names of the activities, while the notation p = ... represents the weight of the relations between activities.

3.3 Discovering weighted-linear temporal logic (W-LTL) tree

After the W-LTL patterns have been discovered, the next step is copying the discovered W-LTL patterns to a .txt file adding Firstactivity (Name First Activity) and Lastactivity (Name Last Activity) at the top of the W-LTL patterns. The format of the wiring weight is changed by adding /\ between the name of the activity and the weight value. The first step in discovering the W-LTL tree model is choosing the first W-LTL pattern. This is the W-LTL pattern with the activity that is found in FirstActivity (activity). This pattern splits into several symbols and activities. For example, Login -> O ((Open discount \/ Open items data)) can be split as Part LTL = [Login, ->, O, ((Open discount, \/, Open items data))]. This part of the W-LTL pattern is then used in discovering the W-LTL tree.

The algorithm will check every part of the W-LTL pattern. An open parenthesis is used to make a sub-node and a closing parenthesis is used to return to the main node. 'x' symbol on tree describes XOR relations, '/\' for AND relations, '\/' for OR relations, and '->' for Sequence relations. The weight value is placed after the name of the activity added with the relation /\. Suppose the probability of the sequence of activities A to B is 0.5, then the name of the node in the tree model is B /\$p = 0.5$.

Table 4. Algorithm to discover weighted-linear temporal logic (W-LTL) patterns

Algorithm 2: Algorithm to discover Weighted-Linear Temporal Logic (W-LTL) patterns

```
 input  : $dataset$ = data in .csv file with all relations,
          $list_{sequence}$ = list of weight values of sequence relation,
          $data_{search}$ = $dataset$.
 output : W-LTL patterns

1  foreach $data$ in $dataset$ do
2    if the relation of data == "NEXT" then
3      print act A -> _O ( act Bp = 1 )

4    if the relation of data == "XORSPLIT" then
5      foreach $data_{search}$ in $dataset$ do
6        $list_{xorsplit}$ = all of XORSPLIT relation
         with outgoing act of data == outgoing act
         of $data_{search}$
7        foreach data in $list\_xorsplit$ do
8          print act A - > _O ( ( act $B_{p=}$
           $weight_{A,B}$ ∨ … ∨ act $n_p$= $weight_{A,n}$) )

9    if the relation of data == "XORJOIN" then
10     foreach $data_{search}$ in $dataset$ do
11       $list_{xorjoin}$ = all of XORJOIN relation with
         incoming act of data == incoming act of
         $data_{search}$
12       foreach data in $list\_xorjoin$ do
13         print _O ( ( act Ap = $weight_{A,B}$ ∨ ... ∨
           act $n_p$= weightn,B ) ) -> _O ( act B )

14   if the relation of data == "ANDSPLIT" then
15     foreach $data_{search}$ in $dataset$ do
16       $list_{andsplit}$ = all of ANDSPLIT relation
         with outgoing act of data == outgoing act
         of $data_{search}$
17       foreach data in $list\_andsplit$ do
18         print act A - > < > ( ( act $B_p$ =
           $weight_{A,B}$ ∧ … ∧ act $n_p$= $weight_{A,n}$ ) )

19   if the relation of data == "ANDJOIN" then
20     foreach $data_{search}$ in $dataset$ do
21     $list_{andjoin}$ = all of ANDJOIN relation with
       incoming act of data == incoming act of
       $data_{search}$
22       foreach data in $list\_andjoin$ do
23       print <> ( ( act $A_p$ = $weight_{A,B}$ ∧ ... ∧
         act $n_p$= $weight_{n,B}$ ) ) -> _O ( act B )

24   if the relation of data == "ORSPLIT" then
25     foreach $data_{search}$ in $dataset$ do
26       $list_{orsplit}$ = all of ORSPLIT relation with
         outgoing act of data == outgoing act of
         $data_{search}$
27       foreach data in $list\_orsplit$ do
28         print act A - > <> ( ( act $B_p$=
           $weight_{A,B}$ ∨ .. ∨ act $n_p$= $weight_{A,n}$ ) )

29   if the relation of data == "ORJOIN" then
30     foreach $data_{search}$ in $dataset$ do
31       $list_{orjoin}$= all of ORJOIN relation with
         incoming act of data == incoming act of
         $data_{search}$
32       foreach data in $list\_orjoin$ do
33         print <> ( ( act $A_p$ = $weight_{A,B}$ ∨ ... ∨
           act $n_p$= $weight_{n,B}$) ) -> _O ( act B )
34 end
```

Table 5. Form of weighted-linear temporal logic (W-LTL)

Pattern	Weighted-Linear Temporal Logic
Sequence	act -> _O ($y_{p = \ldots}$)
AND Split	act -> < > (($y1_{p =}$ ... ∧ $y2_{p = \ldots}$ ∧ $yn_{p = \ldots}$))
AND Join	< > (($y1_{p =\ldots}$ ∧ $y2_{p =\ldots}$ ∧ $yn_{p =\ldots}$)) -> _O (act)
XOR Split	act -> _O (($y1_{p = \ldots}$ ∨ $y2_{p = \ldots}$ ∨ $yn_{p = ..}$))
XOR Join	_O (($y1_{p =\ldots}$ ∨ $y2_{p =\ldots}$ ∨ $yn_{p =\ldots}$)) -> O (act)
OR Split	act -> < > (($y1_{p =}$ ... ∨ $y2_{p = \ldots}$ ∨ $yn_{p = \ldots}$))
OR Join	< > (($y1_{p = \ldots}$ ∨ $y2_{p =\ldots}$ ∨ $yn_{p = \ldots}$)) -> _O (act)

3.4 Weighted linear temporal logic tree and weighted directed acyclic graph (W-LTLDAG) similarity calculation

The original wDAG similarity [18] cannot distinguish the similarity value of two wDAGs that have two branches with opposite weight values, as explained in Section 2.5. This new similarity method based on Weighted Linear Temporal Logic tree and Weighted Directed Acyclic Graph (W-LTLDAG) proposed a modified wDAG similarity calculation that able to distinguish different parallel relations or same parallel relations that have different occurrence probability of activity relations. The general steps of calculating similarity between 2 wDAGs using W-LTLDAG similarity are shown as below:

1. Deternining the parallel relations between 2 wDAGs, e.g. AND – AND, AND – OR, etc.
2. Calculate the distance of the weights between each branch of the two wDAGs using standard deviation formula as shown in Eq. (2).

$$SD = \left|1 - \sqrt{\frac{\sum_{i=1}^{n} (x_i - \bar{x})^2}{n-1}}\right| \qquad (2)$$

Received: June 17, 2020. Revised: July 17, 2020.

n is the number of branch of wDAG, x_i is the weight value of branch i, and $\bar{x}$ is the mean of the weight values between 2 branches of 2 wDAGs.

3. The calculation of the distance will be done on each branch between 2 wDAGs. It calculates the distance bottom-up.
4. After all the distance of each branches of 2 wDAGs have been calculated, the average of distance values of all branches is calculated as a final similarity value between 2 wDAGs.

For example, Fig. 3 has 2 wDAGs, A and B, with 2 branches, b and c. The weight values of the branches of wDAG A, B in Fig. 3 are inverse, while Fig. 4 has wDAGs, A and C, with 2 branches, b and c, with the same value of weights. Using W-LTLDAG similarity, the first wDAG A, B in Fig. 3 and the second wDAG A, C in Fig. 4 will have different similarity values. The similarity values of wDAGs in Fig. 3 and Fig. 4 are shown as below:

$$W - LTLDAG_{sim}(A.DAG1, B.DAG1) = 1 - \sqrt{\frac{(0.6-0.4)^2+(0.4-0.6)^2}{2-1}} + \sqrt{\frac{(0.4-0.6)^2+(0.6-0.4)^2}{2-1}} = 0.84 \quad (3)$$

$$W - LTLDAG_{sim}(A.DAG1, C.DAG1) = 1 - \sqrt{\frac{(0.6-0.6)^2+(0.4-0.4)^2}{2-1}} + \sqrt{\frac{(0.4-0.4)^2+(0.6-0.6)^2}{2-1}} = 1 \quad (4)$$

4. Results and analysis

The dataset used in this study was obtained from retail companies with 5000 cases and 700 traces. The execution processes of the retail business process were stored in a graph database. The start activity is user login and the end of activity is logout. The information contained in the graph database is case id, activity, user id, company id, value, division id, created at, and updated at.

This section shows the discovered graph model, the W-LTL model, the W-LTL tree model based on the experiment. The discovered W-LTL tree model will be converted into wDAG model and compared with another wDAG using W-LTLDAG similarity calculation.

4.1 Discovering graph model based on graph database

Graph model discovered from the retail graph database is shown in Fig. 5. A to T symbols on Fig. 5 refers to user login, open selling, open discount, open purchase, choose item, choose payment method, choose customer type, set amount of discount, set discount type, set discount end time, set discount name, set discount start time, choose supplier, input expedition price, buy items, choose expedition, sell item, finish discount, finish transaction, and logout.

The name of the nodes and the relations between the activities are exported to a .csv file. The .csv file is then imported into Anaconda platform for discovering W-LTL patterns. An example of a .csv file exported from Neo4j Graph Platform is shown in Table 6. Besides that, a graph model that only contains sequence relations is also built in Neo4j. The nodes and their relations are exported to a .csv file. This sequence relation file is then used to calculate the weight of each sequence of activities.

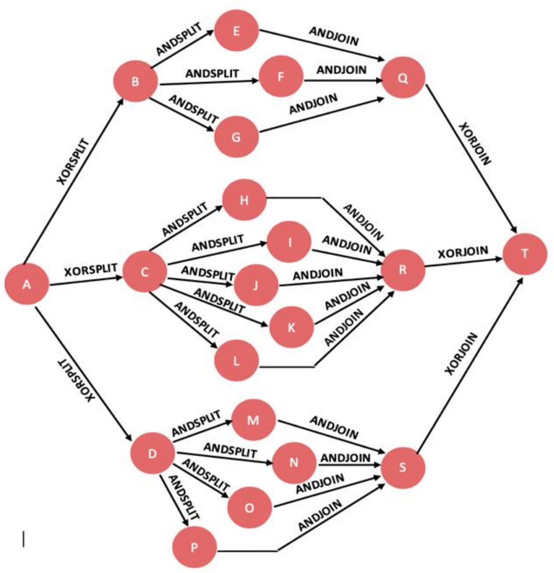


Figure. 5 Graph model based on graph database

Table 6. Snippet of .csv file with all exported from neo4j

1	user login, XORSPLIT, open discount
2	open discount, ANDSPLIT, set discount end time
3	open discount, ANDSPLIT, set discount name
4	open discount, ANDSPLIT, set discount type
5	open discount, ANDSPLIT, set discount start time
6	open discount, ANDSPLIT, set amount of discount
7	set discount start time, ANDJOIN, finish discount
8	set discount end time, ANDJOIN, finish discount
9	set discount name, ANDJOIN, finish discount
10	set discount type, ANDJOIN, finish discount

Received: June 17, 2020. Revised: July 17, 2020.

Table 7. Discovered weighted linear temporal logic (W-LTL)

1	user login -> _O ((open discount $^{P}{}_{=0.037}$ ∨ open selling $^{P}{}_{=0.852}$ ∨ open purchasing $^{P}{}_{=0.111}$))
2	open discount -> <> ((set discount end time $^{P}{}_{=0.346}$ ∧ set discount name $^{P}{}_{=0.192}$ ∧ set discount type $^{P}{}_{=0.154}$ ∧ set discount start time $^{P}{}_{=0.154}$ ∧ set amount of discount $^{P}{}_{=0.154}$))
3	<> ((set discount start time $^{P}{}_{=0.385}$ ∧ set discount end time $^{P}{}_{=0.154}$ ∧ set discount name $^{P}{}_{=0.038}$ ∧ set discount type $^{P}{}_{=0.154}$ ∧ set amount of discount $^{P}{}_{=0.269}$)) -> _O (finish discount)
4	open selling -> <> ((choose item $^{P}{}_{=0.032}$ ∧ choose payment type $^{P}{}_{=0.015}$ ∧ choose customer type $^{P}{}_{=0.953}$))
5	<> ((choose customer type $^{P}{}_{=0.032}$ ∧ choose payment type $^{P}{}_{=0.050}$ ∧ choose item $^{P}{}_{=0.918}$)) -> _O (sell item)
6	open purchasing -> <> ((choose supplier $^{P}{}_{=0.218}$ ∧ choose expedition $^{P}{}_{=0.231}$ ∧ buy items $^{P}{}_{=0.269}$ ∧ input expedition price $^{P}{}_{=0.282}$))
7	<> ((choose expedition $^{P}{}_{=0.218}$ ∧ choose supplier $^{P}{}_{=0.333}$ ∧ input expedition price $^{P}{}_{=0.269}$ ∧ buy items $^{P}{}_{=0.179}$)) -> _O (finish transaction)
8	_O ((finish discount $^{P}{}_{=0.037}$ ∨ sell item $^{P}{}_{=0.852}$ ∨ finish transaction $^{P}{}_{=0.111}$)) -> _O (logout)

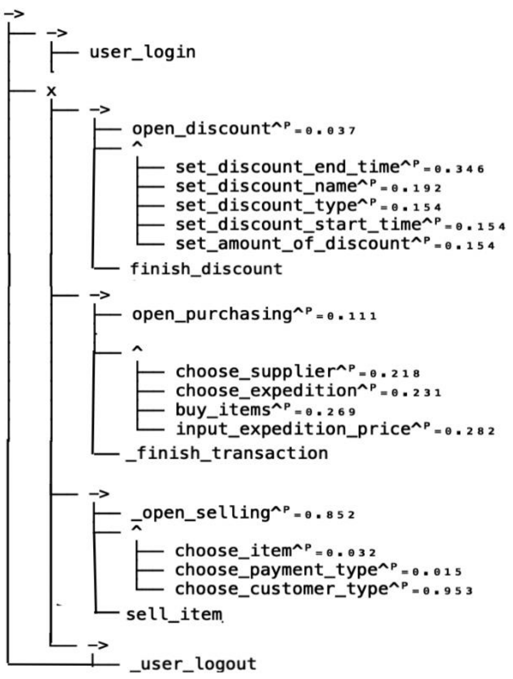
4.2 Discovering weighted-linear temporal logic (W-LTL) patterns from graph model

Both .csv files exported from Neo4j are then imported into Anaconda platform. The .csv file with only sequence relations is used to calculate the weights, while the .csv file with Sequence, AND, XOR, and OR relations is used to discover the W-LTL patterns. The discovered W-LTL patterns are shown in Table 7.

4.3 Discovering weighted-linear temporal logic (W-LTL) tree from W-LTL

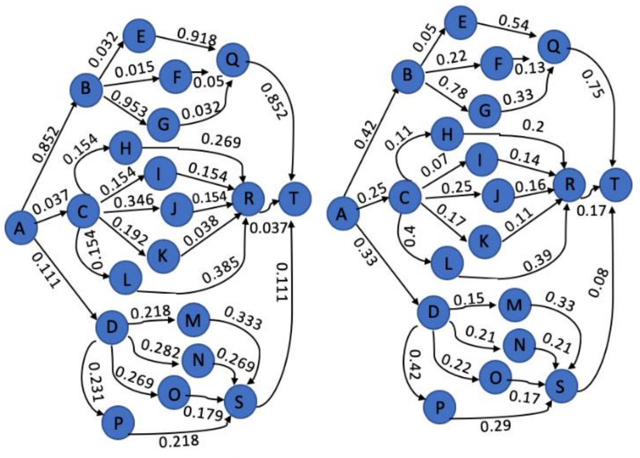
Data from W-LTL patterns are copied into a .txt file. Firstactivity (Name First Activity) and Lastactivity (Name Last Activity) are added at the top of the W-LTL patterns. The format of the wiring weight is changed by adding /\ before the weight value. For example, from finish transaction -> _O (logout p = 1.0) to finish transaction -> _O (logout/\\p = 1.0). Import the .txt file into the Anaconda platform to convert it into a W-LTL tree using the Python programming language. Each W-LTL pattern is split into symbols and activities. The W-LTL tree is shown in Fig. 6.

The discovered W-LTL tree then converted into wDAG A as shown in Fig. 7 (a) and will be compared with wDAG B as shown in Fig. 7 (b). Relation of each branches of both wDAGs can be shown in Fig.



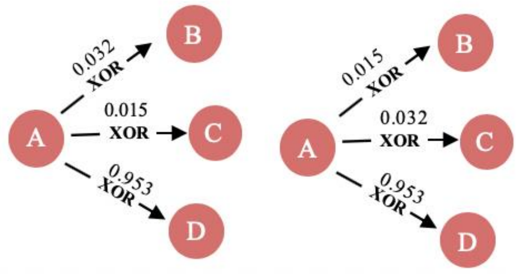
```
->
├── ->
│   ├── user_login
├── x
│   ├── ->
│   │   ├── open_discount^P=0.037
│   │   ├── ^
│   │   │   ├── set_discount_end_time^P=0.346
│   │   │   ├── set_discount_name^P=0.192
│   │   │   ├── set_discount_type^P=0.154
│   │   │   ├── set_discount_start_time^P=0.154
│   │   │   └── set_amount_of_discount^P=0.154
│   │   └── finish_discount
│   ├── ->
│   │   ├── open_purchasing^P=0.111
│   │   ├── ^
│   │   │   ├── choose_supplier^P=0.218
│   │   │   ├── choose_expedition^P=0.231
│   │   │   ├── buy_items^P=0.269
│   │   │   └── input_expedition_price^P=0.282
│   │   └── _finish_transaction
│   ├── ->
│   │   ├── _open_selling^P=0.852
│   │   ├── ^
│   │   │   ├── choose_item^P=0.032
│   │   │   ├── choose_payment_type^P=0.015
│   │   │   └── choose_customer_type^P=0.953
│   │   └── sell_item
│   └── ->
└── _user_logout
```

Figure. 6 The discovered W-LTL tree



(a) (b)

Figure. 7: (a) wDAG A and (b) wDAG B



(a) (b)

Figure. 8: (a) XOR relation - first graph A and (b) XOR relation - first graph B

5. Using W-LTLDAG similarity as describes in Section 3.4, the similarity of both wDAGs is shown as below:

1. $SD_{distance}$ (wDAG A.T, wDAG B.T) = 0.812

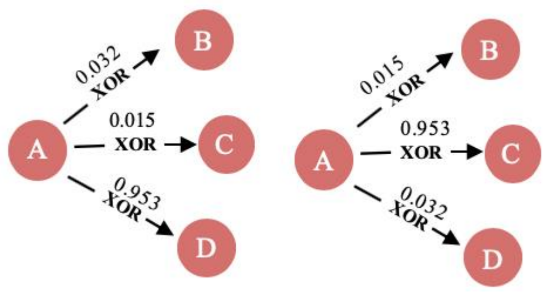


Figure. 9: (a) XOR relation - second graph A and (b) XOR relation - second graph B

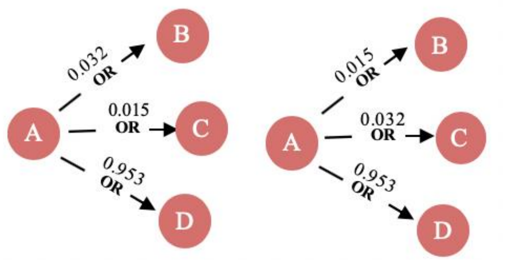


Figure. 10: (a) OR relation - first graph A and (b) OR relation - first graph B

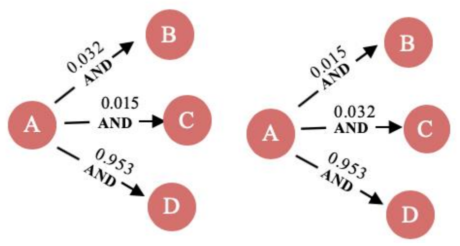


Figure. 11: (a) AND relation - first graph A and (b) AND relation - first graph B

2. $SD_{distance}$ (wDAG A.Q, wDAG B.Q) = 0.465
3. $SD_{distance}$ (wDAG A.R, wDAG B.R) = 0.882
4. $SD_{distance}$ (wDAG A.S, wDAG B.S) = 0.899
5. $SD_{distance}$ (wDAG A.B, wDAG B.B) = 0.72
6. $SD_{distance}$ (wDAG A.C, wDAG B.C) = 0.652
7. $SD_{distance}$ (wDAG A.D, wDAG B.D) = 0.732
8. $SD_{distance}$ (wDAG A.A, wDAG B.A) = 0.389

Hence, the similarity between wDAG A, B is

$$W - LTLDAG_{sim}(wDAG\ A, B) = 0.812 + 0.465 + 0.882 + 0.899 + 0.72 + 0.6 + 0.732 + 0.389 = \frac{5.551}{8} = 0.69 \quad (5)$$

From another event log, we have first graph A and graph B showed in Fig. 8 (a) and 8 (b), respectively. Then we compared the first graphs with the second

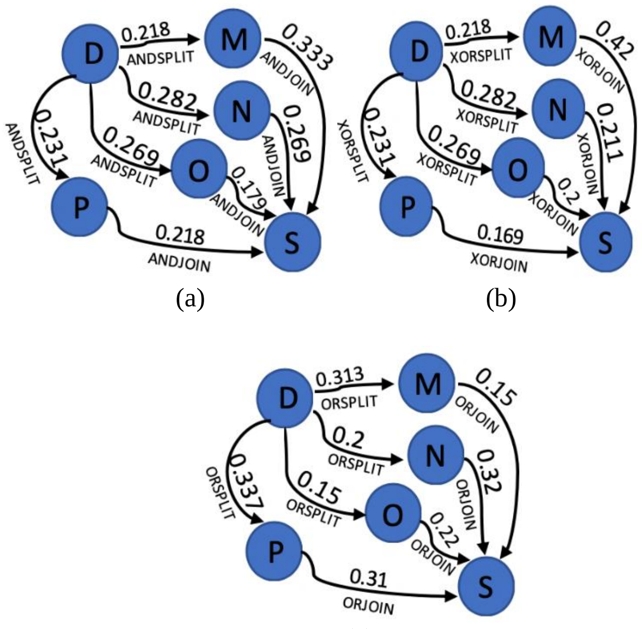


Figure. 12: (a) AND relation – graph A, (b) XOR relation - graph B, and (c) OR relation – graph C

graph A and graph B with the same weight value but with different order showed in Fig. 9 (a) and 9 (b).

Graphs in Fig. 8 (a) and 9 (a) are similar graphs with the same nodes, relations, and weights. graph A and graph B in Fig. 8 (a) and 8 (b) have the same nodes and relations, the same weight value but in a different order with a small difference value between branches on both graphs. While graph A and graph B in Fig. 9 (a) and 9 (b) have the same nodes and relations, the same weight value but in a different order with a large difference weight value. Fig. 10 and Fig. 11 has the same weights with Fig. 8 but the relation of Fig. 10 is OR and the relation of Fig. 11 is AND.

To test the reliability of the algorithm in calculating similarity between graphs with different relation, we used D-M-N-O-P-S graph with AND relations in Fig. 7 and compared with other 2 graphs with OR and XOR relations shown in Fig. 12. We compared the similarity between graphs with the existing methods: wDAG [18], WGED [13], TPED [19], and cosine-TDP [19]. The results of the similarity of each method are shown in Table 8.

Based on the test results in Table 8, all algorithms give a true similarity value of 1 for graphs with the same nodes, relations, and weight (Fig. 8 (a) and 9 (a)). wDAG algorithm [18] gives similarity value of 1 for all experiments. For experiment 1-5, wDAG algorithm cannot distinguish weight between 2 graphs, both in graphs with the similar values of weight (Fig. 8 (a) and Fig. 9 (a)) and graphs with the

Table 8. Similarity result of parallel relationships

Similarity	Existing Methods				W-LTLDAG
	wDAG [18]	WGED [13]	TPED [19]	Cosine-TDP [19]	
1. XOR relation (Fig. 8 (a)) and XOR relation (Fig. 9 (a))	1	1	1	1	1
2. XOR relation (Fig. 9 (a)) andXOR relation (Fig. 9 (b))	1	1	0.813	0.766	0,327
3. XOR relation (Fig. 8 (a)) and XOR relation (Fig. 8 (b))	1	1	0.813	0.999	0.976
4. OR relation (Fig. 10 (a)) andOR relation (Fig. 10 (b))	1	1	0.875	0.825	0.976
5. AND relation (Fig. 11 (a)) and AND relation (Fig. 11 (b))	1	1	0.938	0.691	0.976
6. AND relation (Fig. 12 (a)) and OR relation (Fig. 12 (c))	1	1	0.821	0.854	0.948
7. OR relation (Fig. 12 (c)) and XOR relation (Fig. 12 (b))	1	1	0.811	0.849	0.937
8. AND relation (Fig. 12 (a)) and XOR relation (Fig. 12 (b))	1	1	0.78	0.712	0.985

same weight values but in a randomize order (Fig. 8 (a) – Fig. 8 (b), Fig. 9 (a) – Fig. 9 (b), Fig. 10 (a) – Fig. 10 (b), and Fig. 11 (a) – Fig. 11 (b)). In experiment 6-8 wDAG algorithm also gives similarity value of 1, means that in these cases, wDAG algorithm cannot distinguish similarity value between AND – OR, OR – XOR, and AND - XOR relations. wDAG algorithm only gives a different similarity value if there are a different node between 2 graphs, but always give the same similarity value of 1 for graphs with the same nodes, although it has a different weight values. WGED [13] also gives similarity value of 1 for all experiments. Since it just concerned in inserted/deleted nodes, WGED gives the same similarity value for graphs with the same nodes, although with the different weight values of the branches. The next previous research, TPED [19], gives the same similarity value of 0.813 to the first graphs (Fig. 8 (a) and 8 (b)) and second graphs (Fig. 9 (a) and 9 (b)) although both have different values of weight. TPED cannot distinguish weight between nodes since it just only concerns on relations without giving regards to the number of traces. It only depends on the structure of the nodes, edges, and the relation operator value, make it always provide the same similarity value both on graphs with the same relation or different relations if the structure of the nodes and edges are same, even both have different weight values. Cosine-TDP [19] can distinguish two graphs with different values of weight by provide a different similarity value on graph A, B in Fig. 8 and graph A, B in Fig. 9. However, in Fig. 9, Cosine-TDP gives a high similarity value of 0.766 or 76% similar while the weights of both graphs is showing high different. Proposed method gives a more accurate calculation with the low similarity value, 0.327 or 33% similar. If we observe in row 3, 4, and 5 in Table 8, TPED and Cosine-TDP give different similarity values between XOR-XOR, OR-OR, and AND-AND relation although both graphs have the same nodes and weights. While proposed method can ensure to give the same similarity values on XOR-XOR, OR-OR, and AND-AND relation between graphs with the same nodes and weight values. For comparison between graphs with different relations in experiments 6-8, Cosine-TDP gives the highest similarity value in AND-OR. Whereas in Fig. 12, AND - XOR has the highest closeness since both have the same weight values in all of the split-edges. Proposed method proves that AND - XOR gives the highest similarity value (0.985) compared to AND – OR (0.948) and OR – XOR (0.937).

The proposed method (W-LTLDAG) calculates similarity based on the occurrences probability of relationships of activities. The more similar 2 graphs are, the higher the similarity value they have. Since it calculates the distance between weights of 2 branches using standard deviation, makes it able to distinguish 2 graphs with the same weight values but in a randomized order which cannot handled by wDAG similarity algorithm [18]. W-LTLDAG not only concern to insertion or deletion nodes, but also the occurrence probability of activities so make it able to distinguish graphs with the different weights that cannot be handled by WGED [13] and TPED [19]. Finally, W-LTLDAG is very concerned about the closeness of the occurences of activities between graphs, so that it can ensure that graphs with adjacent weight values will provide a higher similarity

Received: June 17, 2020. Revised: July 17, 2020.

compared to graphs with a large difference of weights that cannot handled by Cosine-TDP [19].

5. Conclusion

The proposed algorithm for discovering a Weighted-Linear Temporal Logic (W-LTL) tree from a graph-based business process model is implemented using Neo4j Graph Platform and Python programming language. The case study employed data from a complex retail business process containing Sequence, XOR, AND, and OR relations. The information from the retail business process in the form of a graph database was represented in a graph model using the Neo4j Graph Platform. The relations from the graph model were then processed using Python programming language to discover W-LTL patterns. Each W-LTL pattern is split into a partial W-LTL pattern containing activities and symbols, which is used as the basis for discovering the W-LTL tree model. Giving a weight to the relations of the activities is expected to be useful to obtain and solve various problems, such as similarity calculation, anomaly detection, and fraud detection.

The proposed similarity method (W-LTLDAG) is compared with other existing methods: wDAG algorithm, WGED, TPED, and Cosine-TDP. wDAG algorithm only gives a different similarity value if there are a different node between 2 graphs, but always give the same similarity value of 1 for graphs with the same nodes, although it has a different weight values. Since it just concerned in inserted or deleted nodes, WGED gives the same similarity value for graphs with the same node, although with the different weight values of the branches. TPED only depends on the structure of the nodes, edges, and the relation operator value, make it always provide the same similarity value both on graphs with the same relation or different relations if the structure of the nodes and edges are same, even both have different weight values. Cosine-TDP can distinguish graphs with parallel relationships that have different weight values, however proposed method gives more accurate in calculating similarity since it gives a lower similarity (0.327) between graphs with the large difference of weight, and gives a high similarity (0.976) on graphs with the small difference in weight. While Cosine-TDP gives a very high similarity (0.999) which is ≈ 1 although both graphs has different weight values, and also gives a high enough similarity (0.766) although both graphs have a large difference in weight. W-LTLDAG is very concerned about the closeness of the occurences of activities between graphs, so that it can ensure that graphs with adjacent weight values will provide a higher similarity compared to graphs with a large difference of weights that cannot handled by Cosine-TDP. For future works, discovering W-LTL Tree and similarity method that able to handle overlapping activities and repetitive activities (looping) need to be developed.

Conflicts of Interest

The authors declare no conflict of interest.

Author Contributions

Khairiyyah Nur Aisyah and Kelly R. Sungkono contributed in the formulation of methods, the implementation of algorithm, conduct of experiments, and the formation of the paper. Riyanarto Sarno as supervisors propose ideas of problems, supervised problem ideas and proposed the novelty contribution of the paper.

Acknowledgments

The authors would like to sincerely thank Institut Teknologi Sepuluh Nopember, the Directorate General of Higher Education, Indonesian Ministry of Education and Culture, and LPDP through RISPRO Invitation Program for funding the research.

References

[1] R. S. Aguilar-Savén, "Business process modelling: Review and framework," *International Journal of Production Economics*, Vol. 90, No. 2, pp. 129–149, 2004.

[2] R. Sarno, Y. A. Effendi, and F. Haryadita, "Modified Time-Based Heuristics Miner for Parallel Business Processes", *International Review on Computers and Software*, Vol. 11, No. 3, pp. 249, 2016.

[3] D. Rahmawati, M. A. Yaqin, and R. Sarno, "Fraud detection on event logs of goods and services procurement business process using Heuristics Miner algorithm", In: *Proc. of 2016 International Conf. on Information and Communication Technology and Systems (ICTS)*, pp. 249–254, 2017.

[4] R. Sarno, R. D. Dewandono, T. Ahmad, M. Naufal, and F. Sinaga, "Hybrid Association Rule Learning and Process Mining for Fraud Detection", *IAENG International Journal of Computer Science*, Vol. 42, No. 2, pp. 59-72, 2015.

[5] D. Huang, D. Mu, L. Yang, and X. Cai, "CoDetect: Financial Fraud Detection with Anomaly Feature Detection", *IEEE Access*, Vol. 6, pp. 19161–19174, 2018.

[6] A. Sanaa, S. B. Abid, A. Boulila, C. Messaoud, M. Boussaid, and N. B. Fadhel, "Modeling hydrochory effects on the Tunisian island populations of Pancratium maritimum L. using colored Petri nets", *BioSystems*, Vol. 129, pp. 19–24, 2015.

[7] R. Sarno and K. R. Sungkono, "Coupled Hidden Markov Model for Process Discovery of Non-Free Choice and Invisible Prime Tasks", *Procedia Computer Science*, Vol. 124, pp. 134–141, 2017.

[8] K. Rabuzin and M. Šestak, "Towards Inheritance in Graph Databases," In: *Proc. of International Conf. on Information Management and Processing (ICIMP)*, pp. 115–119, 2018.

[9] M. Weske, "Business Process Management: Concepts, Languages, Architectures", *Berlin: Springer*, 2007.

[10] R. Sarno, B. A. Sanjoyo, I. Mukhlash, and H. Maria, "Petri Net model of ERP business process variation for Small and Medium Enterprises", Vol. 54, No. 1, pp. 31-38, 2013.

[11] A. Schoknecht, T. Thaler, P. Fettke, A. Oberweis, and R. Laue, "Similarity of business process models - A state-of-the-art analysis", *ACM Computing Surveys*, Vol. 50, No. 4, pp. 1–33, 2017.

[12] J. Becker, P. Delfmann, H. A. Dietrich, M. Steinhorst, and M. Eggert, "Business process compliance checking – Applying and evaluating a generic pattern matching approach for conceptual models in the financial sector", *Information Systems Frontiers*, Vol. 18, No. 2, pp. 359–405, 2016.

[13] C. Zhou, C. Liu, Q. Zeng, Z. Lin, and H. Duan, "A Comprehensive Process Similarity Measure Based on Models and Logs", *IEEE Access*, Vol. 7, pp. 69257–69273, 2019.

[14] M. J. Amiri, S. Barbara, M. Koupaee, and S. Barbara, "Data-driven Business Process Similarity Data-driven Business Process Similarity", *IET Software*, Vol. 11, No. 6, 2017.

[15] R. Dijkman, M. Dumas, B. V. Dongen, R. Krik, and J. Mendling, "Similarity of business process models: Metrics and evaluation", *Information Systems*, Vol. 36, No. 2, pp. 498–516, 2011.

[16] B. van Dongen, R. Dijkman, and J. Mendling, "Measuring Similarity between Business Process Models", In: *Proc. of International Conf. on Advanced Information Systems Engineering*, pp. 450–464, 2008.

[17] Z. Dong, L. Wen, H. Huang, and J. Wang, "CFS: A behavioral similarity algorithm for process models based on complete firing sequences", In: *Proc. of OTM Conf*, pp. 202–219, 2014.

[18] J. Jin, B. Virendra, K. P. Sushil, K. S. Biplap, "Similarity of weighted directed acyclic graphs", University of New Brunswick, 2006.

[19] C. S. Wahyuni, K. R. Sungkono, and R. Sarno, "Novel parallel business process similarity methods based on weighted-tree declarative pattern models", *International Journal of Intelligent Engineering and Systems*, Vol. 13, No. 1, pp. 65-77, 2019.

[20] K. R. Sungkono and R. Sarno, "Constructing control-flow patterns containing invisible task and non-free choice based on declarative model", *International Journal of Innovative Computing, Information and Control*, Vol. 14, No. 4, pp. 1285–1299, 2018.

[21] V. Huser, "Process Mining: Discovery, Conformance and Enhancement of Business Processes", *Journal of Biomedical Informatics*, Vol. 42, 2012.

[22] J. Joishi, A. Sureka, and N. Delhi, "Graph or Relational Databases : A Speed Comparison for process mining algorithm", *ArXiv*, pp. 1–22, 2017.

[23] T. Savickas and O. Vasilecas, "Business process event log use for activity sequence analysis", In: *Proc. of 2015 Open Conf. of Electrical, Electronic and Information Sciences (eStream)*, pp. 1–4, 2015.

[24] R. Sarno, L. Yang, V. C. Bhavsar, and H. Boley, "The AgentMatcher Architecture Applied to Power Grid Transactions", In: *Proc. of the First International Workshop on Knowledge Grid and Grid Intelligence, Halifax,* pp. 92-99, 2003.

[25] H. Lu, Z. Hong, and M. Shi, "Analysis of film data based on Neo4j", In: *Proc. of 16th IEEE/ACIS International Conf. on Computer and Information Science, ICIS 2017*, pp. 675–677, 2017.