IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received November 22, 2021, accepted December 3, 2021, date of publication December 13, 2021,
date of current version January 5, 2022.

Digital Object Identifier 10.1109/ACCESS.2021.3135298

Improving Heuristic Process Discovery Methods Through Determining the Optimal Split/Join Patterns of Dependency Graphs

MARYAM TAVAKOLI-ZANIANI AND MOHAMMAD REZA GHOLAMIAN
School of Industrial Engineering, Iran University of Science and Technology, Tehran 16844, Iran
Corresponding author: Mohammad Reza Gholamian (gholamian@iust.ac.ir)

ABSTRACT Identifying the split and join patterns of dependency graphs is an essential step in Heuristics Mining process discovery methods. The existing methods determine the split/join patterns (consisting of AND and XOR relations) according to the event log information about the activities involved in the splits and joins. Hence, they neglect the event log information available for the other activities on the paths from split points to join points. On the other hand, the current methods determine the patterns of each split/join separately and do not aim to select the best set of patterns. Therefore, the outputs of the existing methods can be non-optimal. Furthermore, the current methods cannot guarantee that there is a matching And-join for each AND-split, and vice versa. This can make some split/join patterns incapable of being activated. To handle these issues, this paper, for the first time, presents an integer linear programming model which identifies the optimal patterns of splits/joins with regard to all succession information that is available in the event log; simultaneously, it ensures that for each AND-split there is at least one matching AND-join, and vice versa. The objective function of the proposed model is inspired by replay fitness and precision dimensions of process model quality. According to the assessments, the process models obtained by the proposed method are superior to the results of the most prominent methods of determining split/join patterns in terms of replay fitness, precision, simplicity, and matching AND-splits with AND-joins.

INDEX TERMS Heuristics Miner, integer linear programming, process discovery, process mining, mining split/join patterns, optimization.

I. INTRODUCTION

Process discovery is a branch of process mining and a field of research that uses the recorded events of process execution to analyze business processes. A process discovery method takes the recorded event log of process execution as input and produces a process model that best describes the behavior in the event log. The discovery of process models from event logs can be performed for numerous purposes. For example, identifying the models of real processes running in an organization can be an essential step in process improvement since it can provide an accurate insight into the real processes and existing issues. Furthermore, discovering the real process models can be employed for other applications such as documentation, performance analysis, configuring systems, etc. [1].

The associate editor coordinating the review of this manuscript and approving it for publication was Pasquale De Meo.

Heuristic mining methods are among the most popular process discovery techniques due to their numerous advantages, such as robustness to noise and effectiveness even in the presence of unstructured process models [2], [3]. These algorithms have two main steps. At the first step, the dependency graph is discovered; this graph describes the pre-requisite and post-requisite relations between activities. If there is an arc from activity a to activity b in the dependency graph, it means that a is a direct pre-requisite of b and b is a direct post-requisite of a. However, when an activity has multiple post-requisite activities (i.e., there is a split) or when an activity has multiple pre-requisite activities (i.e. there is a join) the dependency graph cannot describe the relationship between the post-requisite or pre-requisite activities. For example, supposing the dependency graph depicted in Fig. 1(a), it can be understood that activity A is the direct pre-requisite of activities B, C, and D. However, it cannot be inferred that whether after occurring A, only one of B, C, or D can occur

(i.e., they are involved in an XOR relation) or all of them can occur in parallel (i.e., they are involved in an AND relation) or any other possibility can occur (i.e., there is a combination of XOR and AND relations). Hence, to clarify the mentioned issue, the dependency graph split and join patterns are determined in terms of AND and XOR relations in the next step of heuristic mining methods.

To the best of our knowledge, all existing methods in determining the split/join patterns of dependency graphs utilize only the event log information about the activities involved in the splits/joins (i.e., activities that have the same pre-requisite/post-requisite activity). In consequence, they neglect the information about the other activities. However, the patterns of splits/joins directly affect the behaviors that the output process model can/cannot replay. For example, all activities on a path from an AND-split to an AND-join can concurrently occur with all other activities on a different path from the AND-split to the AND-join. Thus, considering the successions of activities that are on different paths from a split to a join point can lead to a better decision about split/join patterns. Moreover, the existing methods determine the patterns of each split/join without considering their effects on the overall quality of the process model; in consequence, their results are likely to be non-optimal. On the other hand, none of the existing methods can guarantee the matching of AND-splits and AND-joins. i.e., none of them can ensure that: 1) for each AND-split, there are some paths that are branched from the AND-split and are joined in an AND-join 2) for each AND-join, there are some paths that are joined in the AND-join and are branched from an AND-split. This can lead to the inability of some split/join patterns to be activated.

To handle the issues mentioned above, this paper suggests benefiting from the potentials of mathematical programming. Mathematical programming has been used in some process mining applications; nevertheless, it has not been used to determine the split/join patterns of dependency graphs to the best of our knowledge. Hence, in this paper:

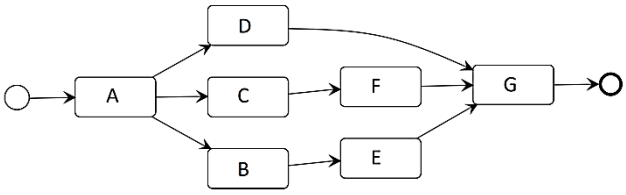
- A novel integer linear programming (ILP) model is proposed for determining the dependency graph split/join patterns. The objective function of the proposed ILP model focuses on finding the optimal combination of concurrent activities considering the event log occurred successions for all activities.
- The proposed Objective function of the ILP model employs the concept of replay fitness and precision measures which are among the most prominent measures in the literature of evaluating the quality of process models. (The replay fitness measure describes whether the behavior in the event log can be reproduced by the process model. Whereas the precision measure describes whether the behaviors that the process model can produce are present in the event log [1]).
- The ILP model constraints guarantee the accordance of all concurrencies, as well as the matching of AND-splits and AND-joins. i.e., they ensure that if two activities are on an AND-split, there are some paths started from them that join in an AND-join, and vice versa.
- After determining the concurrent activities, a modified version of the method has been used by Heuristics Miner is presented to convert the dependency graph into a Causal net by describing the split/join patterns in terms of output/input bindings of Causal nets.

The rest of this paper is organized as follows. A review of the literature and formal definitions are presented in Sections 2 and 3. Section 4 discusses and introduces the proposed model. The method applied for converting the ILP model outputs to process models is presented in Section 5. Section 6 presents the experimental results, and finally, Section 7 concludes the paper and suggests some future studies topics.

II. LITERATURE REVIEW

To the best of our knowledge, there is no study in the literature dedicated only to determining the type of dependency graph splits/joins, and this topic has always been discussed as a part of studies that propose/modify a heuristic mining method. Heuristics Miner [4] is the first heuristic mining method. The outputs of this method are in Heuristic nets notation. After constructing the dependency graph, this method calculates a measure of concurrency between each pair of activities. The mentioned measure is calculated based on the frequency of the occurred successions in the event log. The split/join patterns are determined based on a minimum concurrency measure threshold for activities involved in the splits/joins. Hence, Heuristics Miner neglects the concurrency measures available for all other activities that are on a path from a split to a join point. Consequently, its results are likely to be non-optimal and can be improved. In addition, the mentioned method does not utilize any solution to guarantee the matching of AND-splits and AND-joins. i.e., it is possible that for an AND-split, there are no paths to an AND-join, and vice versa.

For example, suppose the event log and the dependency graph presented in Fig. 1. The result of the determination of the dependency graph split/join patterns by the Heuristics Miner method is illustrated in Fig. 2(a). The output process model was converted to BPMN notation to make the result more visually understandable. As it can be seen, according to the output process model, activities F and E can occur in parallel. However, in the event log, they never occurred in the same trace; hence, the process model is not precise and can replay the behaviors that are not seen in the event log. This happened because the method employed by Heuristics Miner only considers the concurrency measures for the activities B and C that are involved in a split, and it does not consider the concurrency measures for activities E and F that are on a path from the split to a join. Moreover, in the obtained process model, there is no matching AND-join for the AND-split that activities B and C are involved in. The obtained process model can also replay some infrequent traces, such as $< A, D, G >$; this leads to reducing the precision and

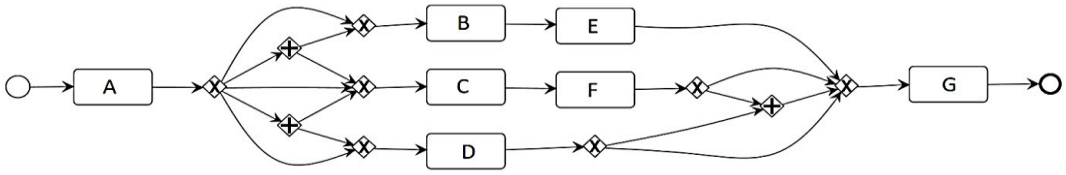


(a)

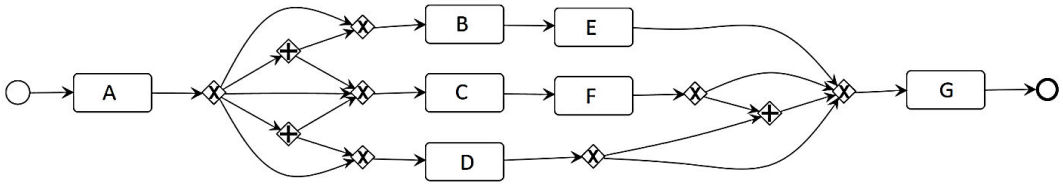
Trace	Frequency
<A,C,B,E,G>	3
<A,B,C,F,G>	3
<A,C,D,F,G>	4
<A,D,C,F,G>	3
<A,C,F,D,G>	4
<A,D,G>	1
<A,C,F,G>	1
<A,B,E,G>	11

(b)

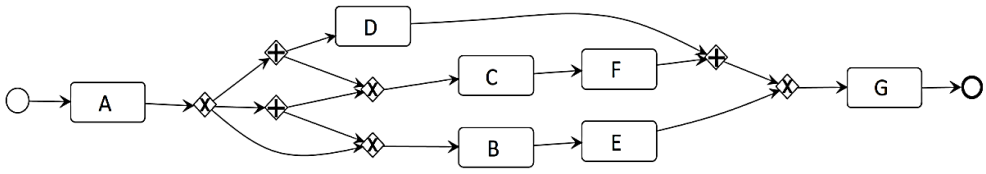
FIGURE 1. (a) An example of a dependency graph, and (b) the event log related to the dependency graph.



(a)



(b)



(c)

FIGURE 2. The result of applying (a) Heuristics Miner, (b) Flexible Heuristics Miner, and (c) Fodina methods to determine the split/join patterns of the dependency presented in Fig. 1 (to make the results more visually understandable, the output process models were converted to BPMN notation).

increasing the visual complexity (and as a result decreasing the understandability) of the process model.

Flexible Heuristics Miner [5] is another heuristic mining method. The outputs of this method are in Causal nets notation which, compared to Heuristic nets, uses a different representation for the split/join patterns. Using a specific procedure, for each split point, this method counts the frequency of appearance of each possible pattern (i.e., each subset of activities that are involved in the split) after the split point. For each join point, the same approach is used to calculate the frequency of occurrence of each possible pattern (i.e., each subset of activities involved in the join) before the join point. Then, based on the achieved values, the method describes the split/join patterns in terms of AND and XOR relations. The method also does not guarantee the matching of AND-splits and AND-joins. Back again to the previous example, Fig. 2(b) shows the result of the determination of the dependency graph split/join patterns by employing the Flexible Heuristics Miner method. It can be seen that the obtained process model suffers from issues that are similar to the problems that exist in the process model obtained by applying the Heuristics Miner method.

In 2012, [6] introduced a version of Heuristics Miner that was dedicated to discovering process models from streaming event data. This method uses an approach similar to Heuristics Miner in determining split/join patterns; however, to make the algorithm stream-aware, it utilizes a different method for achieving the frequency of activities and

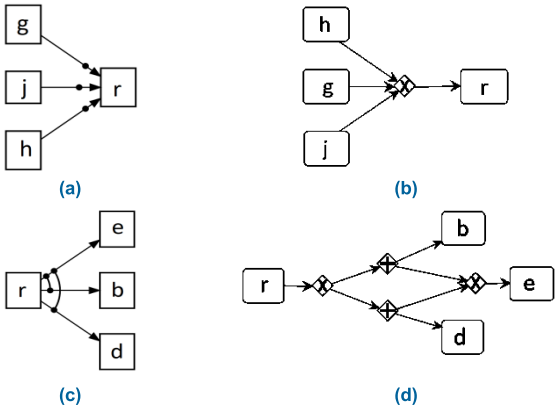


FIGURE 3. (a) Representation of input bindings of activity r: $I(r) = \{\{j\}, \{h\}, \{g\}\}$ in Causal net notation, (b) representation of input bindings of activity r: $I(r) = \{\{j\}, \{h\}, \{g\}\}$ in BPMN notation (c) representation of output bindings of activity r: $O(r) = \{\{b, e\}, \{d, e\}\}$ in Causal net notation, and (d) Representation of output bindings of activity r: $O(r) = \{\{b, e\}, \{d, e\}\}$ in BPMN notation.

number of occurred direct successions. In 2015, [7] proposed a version of Heuristics Miner that uses the information about the time interval (i.e., start and end time) of occurrence of events. This study uses the approach comparable to Heuristics Miner for determining the split/join patterns; nonetheless, it introduced a new concurrency measure that utilizes the information about the time interval of events.

In 2017, [8] introduced another heuristic mining method called Fodina. This method uses an approach similar to Flexible Heuristics Miner for mining spit/join patterns. However, it introduced a technique to filter less frequent patterns. It also presented some configurable options to make the method more robust to noise. Reference [9] and [10] also introduced another improvement to Heuristics Miner; nevertheless, their methods did not present any innovation in detecting the split/join patterns of dependency graphs. They presented a method to make the process models extracted by Heuristic Miner structured and sound. Considering the previously mentioned example, Fig. 2(c) depicts the result of employing the Fodina method to determine the dependency graph split/join patterns. Accordingly, it encountered issues similar to the problems that exist in the results of Heuristics Miner and Flexible Heuristics Miner methods. Nevertheless, due to the utilized filtering technique, it cannot replay some infrequent traces such as $< A, D, G >$ and as a result, it is visually simpler (and in consequence more understandable) and enjoys higher precision.

To address the existing issues in determining the split/join patterns of dependency graphs, this paper suggests utilizing the potentials of mathematical programming, especially ILP. Employing ILP for process mining applications is not a new topic, and there are numerous studies in this regard, for instance, [11]–[18]. However, to the best of our knowledge, there is no study in the literature dedicated to employing ILP for determining the split/join patterns of dependency graphs. Therefore, the mentioned studies are not related to this study.

III. FORMAL DEFINITIONS

Definition 1 (Event Log (L)): An event log L is a multiset of traces, each mapped onto one process instance. Supposing T_L as the set of all traces in L, each trace $t \in T_L$ is an ordered set of events occurring for the process instance corresponding to t, and $|T_L|$ is the total number of traces in the event log. Each event is mapped to an activity $a \in A_L$, where A_L is the finite set of all distinct activities that are present in L and $|A_L|$ is the size of the set A_L.

Definition 2 (Causal Net (C)): A Causal net is a multiset like $C = (A_C, a_s, a_e, D, I, O)$, in which A_C is a limited set of activities modeled by the causal net. Supposing $\rho(A_C) = \{A' \mid A' \subseteq A_C\}$ as a power set of A_C, $I : A_C \longmapsto \{X \subseteq \rho(A_C) \mid X = \{\emptyset\} \vee \emptyset \notin X\}$ is the set of possible input bindings for each activity, whereas $O : A_C \longmapsto \{X \subseteq \rho(A_C) \mid X = \{\emptyset\} \vee \emptyset \notin X\}$ is the set of possible output bindings for each activity (input and output bindings are sets of the subsets of activities). The Causal net should have an initial activity $a_s \in A_C$ and a final activity $a_e \in A_C$, such that $I(a_s) = \{\emptyset\}$ and $O(a_e) = \{\emptyset\}$. The dependency relation is expressed as $D \subseteq A_C \times A_C$, such that $D = \{(a_1, a_2) \in A_C \times A_C \mid a_1 \in \cup_{as \in I(a_2)} as \wedge a_2 \in \cup_{as \in O(a_1)} as\}$, and all of the activities in Graph (A_C, D), should be placed on a path from a_s to a_e.

The output bindings of each activity create certain obligations, which are resolved by input bindings. For instance, if $r \in A_C$ is an activity with $I(r) = \{\{j\}, \{h\}, \{g\}\}$ (illustrated in Fig. 3(a)), it means that r can be executed if g, h, or j has already been executed. In other words, it means that g, h, and j are involved in an XOR-join. If $O(r) = \{\{b, e\}, \{d, e\}\}$ (illustrated in Fig. 3(c)), it means that after execution of r either both of b and e, or both of d and e should be executed in a parallel (concurrent) manner. In other words, this binding expresses that b and e are involved in an AND-split, it also expresses that d and e are involved in an AND-split, whereas both of the AND-splits are involved in an XOR-split. The equivalents of the bindings mentioned above in BPMN notation are presented in Fig. 3(b) and Fig. 3(d).

Definition 3 (Dependency Graph (G)): assuming a Causal net $C = (A_C, a_s, a_e, D, I, O)$, the graph $DG = (A_C, D)$ is called the dependency graph of C. The function $\theta : a \in A_L \longmapsto A_C$ maps each activity in the event log to an activity in the Causal net/dependency graph.

Definition 4 (Succession and Direct Succession): Assume a log L, activities $a, b \in A_L$, and a trace $t \in T_L$. For trace t, if b occurred after occurring a, it means that b is a successor of a, while, if b occurred immediately after occurring a, it means that b is a direct successor of a. $|a > b|$ is the total number of direct successions of a by b that occurred for all $t \in T_L$.

IV. PROPOSED ILP MODEL

This section introduces the proposed ILP model. The ILP model takes a dependency graph, and its corresponding event log as inputs and aims to find the optimal patterns for the dependency graph splits/joins, considering all successions

TABLE 1. The notations used in the rest of the paper.

Notation	Description
A_L	• The set of all activities that are present in L. (see Definition 1)
A_G	• The set of all activities that are present in DG (see Definition 3)
$\theta(i)$	• The mapping of activity $i \in A_L$ to a member of A_G (see Definition 3)
D_G	• The set of all arcs (dependencies) that are present in DG (see Definition 3)
$ar_{a,b}$	• A binary parameter that if in DG there is an arc from $a \in A_G$ to $b \in A_G$ (in other words if $(a,b) \in D_G$) it should be equal to one; otherwise, it is equal to zero.
a_s	• The initial activity of DG ($a_s \in A_G$)(see Definitions 2 and 3)
$p_{a,b}$	• A binary parameter that if in DG there is a path from $a \in A_G$ to $b \in A_G$ it should be equal to one; otherwise, it is equal to zero.
$plf_{a,b}$	• A binary parameter that if in the graph obtained after removing loop closure arcs of original input dependency graph DG there is a path from $a \in A_G$ to $b \in A_G$, it is equal to one; otherwise, it is equal to zero. (the calculation is described in the appendix)
$APen_{a,b}$	• The penalty dedicated to allowing the direct succession of $a \in A_G$ by $b \in A_G$ in the output process model (it is calculated using Equation (3))
$NAPen_{e,f}$	• The penalty dedicated to not allowing the direct succession of $a \in A_G$ by $b \in A_G$ in the output process model (it is calculated using Equation (4)).
$Support(i)$	• The frequency of occurring $i \in A_L$ in L (see Definition 12).
$Support(i > j)$	• The total number of direct successions of $i \in A_L$ by $j \in A_L$ that are occurred for all $t \in T_L$.(see Definition 12).
$Confidence(i > j)$	• An indicator of causal dependency between $i \in A_L$ and $j \in A_L$ (it is calculated using Equation (2)).
$S(a)$	• The set of activities that are divergent from $a \in A_G$ (see Definition 6).
$J(a)$	• The set of activities that are joined to $a \in A_G$ (see Definition 7).
$MS(a,b)$	• The set of the ordered pair of activities that are corresponding divergent activities of $a \in A_G$ and $b \in A_G$ (see Definition 8)
$MJ(a,b)$	• The set of the ordered pair of activities that are corresponding joined activities of $a \in A_G$ and $b \in A_G$ (see Definition 9)
$PS(a,b)$	• The set of ordered pair of activities that are on the different paths from divergent activities $a \in A_G$ and $b \in S(a)$ (See Definition 10)
$PJ(a,b)$	• The set of ordered pair of activities that are on the different paths to joined activities $a \in A_G$ and $b \in J(a)$ (See Definition 11)
M	• A sufficiently large number.
th	• A user-defined threshold that is used in Equations (3) and (4). It determines the end-user's preference for replay fitness and precision of the output model.

that can be extracted from the event log. The proposed model also guarantees the matching of the AND-splits and AND-joins. In the rest of this section, first, Assuming the dependency graph $DG = (A_G, D_G)$ and event log L as the problem inputs, the utilized notations and variables are defined in Table 1 and Table 2. Then after presenting the definitions and objective function, the model constraints are discussed. Finally, the model constraints are presented in mathematical terms.

It should be noted that according to Definition 2, we assume that DG is connected and all activities in DG are on a path from the initial to the final activity. However, DG can contain cycle subgraphs/ loops (according to [19], a cycle graph is a connected graph consisting of a single cycle/loop, and the number of graph nodes is equal to the number of graph arcs).

The proposed model identifies the optimal combination of concurrent activities regarding the mentioned considerations. The obtained result can be converted to Causal nets input/output bindings by the method described in Section V.

A. DEFINITIONS UTILIZED IN THE ILP MODEL

In order to explain the proposed objective function and constraints, the following definitions are utilized.

Definition 5 (Concurrent Activities): in Causal net $C = (A_G, a_s, a_e, D_G, I, O)$ containing the dependency graph $DG = (A_G, D_G)$ activities $a, b \in A_G$ are supposed as concurrent (i.e., they can occur concurrently) if:

- There is at least one AND-split from which there are paths to both a and b such that at least one of the paths from the AND-split to a has no common activities with any of the paths from the AND-split to b.

TABLE 2. The variables used in the proposed ILP model.

Variable	Description
$h_{a,b}$	• A binary variable that if it is equal to one in the output process model $a \in A_G$ is concurrent with $b \in A_G$. If it is equal to zero a is not concurrent with b.
$r_{(a,b),(c,d)}$	• A binary variable that for divergent activities $a \in A_G$, $b \in S(a)$, and their corresponding joined activities $c \in A_G$ and $d \in J(c)$ it can be equal to one only when c and d are concurrent.
$q_{(a,b),(c,d)}$	• A binary variable that for joined activities $c \in A_G$, $d \in J(c)$, and their corresponding divergent activities $a \in A_G$ and $b \in S(a)$, it can be equal to one only when a and b are concurrent
$u_{(a,b),(e,f)}$	• A binary variable that for activities $e, f \in A_G$ and their corresponding divergent activities $a \in A_G$ and $b \in S(a)$, it can be equal to one only when a and b are concurrent
$v_{(c,d),(e,f)}$	• A binary variable that for activities $e, f \in A_G$ and their corresponding joined activities $c \in A_G$ and $d \in J(c)$, it can be equal to one only when c and d are concurrent.

- There is at least one AND-join that from both a and b there are paths to it, such that at least one of the paths from a to the AND-join has not any common activities with any of the paths from b to the AND-join.

Definition 6 (Divergent Activities): for each activity $a \in A_G$, the set of activities that is divergent from a is defined as follows:

$$S(a) = \{b \in A_G \mid \exists x \in A_G : (x, a) \in D_G \wedge (x, b) \in D_G\} \tag{1}$$

For example, supposing the dependency graph presented in Fig. 4, $S(B) = \{E, H\}$.

Definition 7 (Joined Activities): for each activity $a \in A_G$, the set of activities that are joined to a is defined as follows:

$$J(a) = \{b \in A_G \mid \exists x \in A_G : (a, x) \in D_G \wedge (b, x) \in D_G\} \tag{2}$$

For example, supposing the dependency graph presented in Fig. 4, $J(D) = \{G, H\}$.

Each set of the divergent or joined activities is the set of activities that are respectively involved in a split or a join. However, to express the proposed conditions and determine the activities that can be concurrent more definitions are required that are introduced as follows.

Definition 8 (Corresponding Divergent Activities): for activities $e, f \in A_G$, the set of the ordered pair of activities that are their corresponding divergent activities is defined as follows:

$$\begin{aligned} MS(e,f) = \{(a,b) \in A_G \times A_G | a \in S(b) \\ \wedge \left(plf_{a,e} = 1 \vee a = e\right) \wedge plf_{a,f} = 0 \wedge plf_{b,e} = 0 \\ \wedge \left(plf_{b,f} = 1 \vee b = f\right) \end{aligned} \tag{3}$$

For example, supposing the dependency graph presented in Fig. 4, $MS(C, F) = \{(B, E)\}$ and $MS(H, G) = \{(H, E)\}$.

For activities $e, f \in A_G$, The above definition determines the pair of activities $(a, b) \in A_G \times A_G$ that are involved in a split, such that: 1) there are paths from a to e, and from b to f. 2) In the graph obtained after removing the loops of the original input dependency graph DG, there should be no paths from a to f, or from b to e. The second condition is applied to ensure that none of the paths that are from a to e have common activity with any of the paths from b to f. This condition needs to be guaranteed since this definition is intended to be utilized in the constraints determining the activities that can be concurrent, and according to Definition 5, it is one of the required conditions that determine whether two activities can be concurrent or not. For this purpose, the paths in the graph obtained after removing the loops of the dependency graph DG are used instead of the paths in the original dependency graph DG, because in the original dependency graph, the existence of a path from a to f, or from b to e can be due to loops; thus, it is not necessarily an indicator of the presence of a common activity in at least one of the paths from a to e with at least one of paths from b to f. Deriving the loop-free graph from the original dependency graph is performed so that only the loop closure arcs are removed, and no other arcs are removed. The details about the employed procedure for removing the loop closure arcs are presented in the appendix. After obtaining the graph, detecting the graph paths is simply possible by various methods in the literature for achieving reachability matrix, like Warshall's Algorithm [20]. Similarly, the following definitions are proposed.

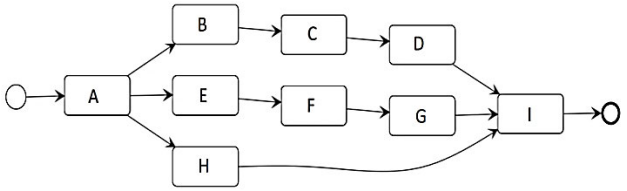


FIGURE 4. An example of a dependency graph.

Definition 9 (Corresponding Joined Activities): for activities $e, f \in A_G$ the set of the ordered pair of activities that are their corresponding joined activities is defined as follows:

$$\begin{aligned} MJ(e,f) = \{(c,d) \in A_G \times A_G | c \in J(d) \\ \wedge \left(plf_{e,c} = 1 \vee e = c\right) \wedge plf_{e,d} = 0 \wedge plf_{f,c} = 0 \\ \wedge \left(plf_{f,d} = 1 \vee f = d\right) \end{aligned} \tag{4}$$

For example, supposing the dependency graph presented in Fig. 4, $MJ\,(C,F)=\{(D,G)\}$ and $MJ\,(H,C)=\{(H,D)\}$.

Definition 10 (Activities on the Different Paths from Divergent Activities): for divergent activities $a\in A_G$ and $b\in S\,(a)$, the set of ordered pair of activities that are on the different paths from a and b is defined as follows:

$$PS\,(a,b)=\{(e,f)\in A_G\times A_G|\,(plf_{a,e}=1\vee e=a)\wedge plf_{a,f}=0\wedge plf_{b,e}=0\wedge(plf_{b,f}=1\vee f=b)\tag{5}$$

For example, supposing the dependency graph presented in Fig. 4, $PS\,(B,E)=\{(B,E),(B,F),(B,G),(C,E),(C,F),(C,G),(D,E),(D,F),(D,G)$.

Definition 11 (Activities on the Different Paths to Joined Activities): for joined activities $c\in A_G$ and $d\in J\,(c)$, the set of ordered pair of activities that are on different paths to c and d is defined as follows:

$$PJ\,(c,d)=\{(e,f)\in A_G\times A_G|\,(plf_{e,c}=1\vee e=c)\wedge plf_{e,d}=0\wedge plf_{f,c}=0\wedge(plf_{f,d}=1\vee f=d)\tag{6}$$

For example, supposing the dependency graph presented in Fig. 4, $PJ\,(D,G)=\{(D,E),(D,F),(D,G),(C,E),(C,F),(C,G),(B,E),(B,F),(B,G)$.

Definition 12 (Support): Support (i) is the frequency of occurring $i\in A_L$ in L, whereas, *Support* $(i>j)$ is the total number of direct successions of $i\in A_L$ by $j\in A_L$ that happened for all $t\in T_L$. In fact, it is equal to $|i>j|$ (see Definition 4).

B. OBJECTIVE FUNCTION

The proposed objective function is inspired by two of the most prominent measures in assessing the quality of process models, namely, replay fitness and precision. The replay fitness describes to what extent the process model can reproduce the behaviors that exist in the event log. On the other hand, the precision describes to what extent the behaviors that the process model can reproduce are present in the event log. Replay fitness and precision are complementary measures, and a model with high replay fitness but low precision (and vice versa) is not an appropriate descriptor for the behaviors in the event log. Thus, the proposed objective function considers both measures.

To consider replay fitness and precision, two ideas are utilized. For event log L and process model C, the more C allows the direct successions that exist in L, the higher the replay fitness. Similarly, the more C allows the direct successions that do not exist in L, the lower the precision. To apply these ideas, first, the direct successions allowed by a process model should be identified. The following rules are employed for this purpose:

- For $i,j\in A_L$,if in the process model there is an arc from $\theta\,(i)$ to $\theta\,(j)$, it means that the model allows the direct succession of $\theta\,(i)$ by $\theta\,(j)$.
- For $i,j\in A_L$, if in the process model $\theta\,(i)$ and $\theta\,(j)$ are concurrent, it means that the model allows the direct succession of $\theta\,(i)$ by $\theta\,(j)$, and it also allows the direct succession of $\theta\,(j)$ by $\theta\,(i)$.

Hence, the formula presented in (7) is proposed for the objective function. By optimizing this objective function, it can be ensured that maximum precision is achieved while attaining a certain level of replay fitness.

$$Min\sum_{e\in A_G}\sum_{f\in A_G}h_{e,f}\times APen_{e,f}+\sum_{e\in A_G}\sum_{f\in A_G}(1-ar_{e,f}-h_{e,f})\times NAPen_{e,f}\tag{7}$$

where, $ar_{e,f}$ is a binary parameter that if in the input dependency graph DG, there is an arc from e to f, it is equal to one; otherwise, it is equal to zero. $h_{e,f}$ is a binary variable that if it is equal to one, it means that in the output process model e and f are concurrent, otherwise, e and f cannot be concurrent. $APen_{e,f}$ and $NAPen_{e,f}$ are penalties assigned to respectively allowing and not allowing the direct succession of e by f in the output process model. Therefore, the first part of the objective function relates to optimizing the model precision, whereas the second part pertains to obtaining a certain level of fitness. Prior to explaining the calculation of $APen_{e,f}$ and $NAPen_{e,f}$, the term *Confidence*, which is borrowed from the data mining (association rule learning) literature [21], should be introduced. For event log L and activities $i,j\in A_L$, *Confidence* $(i>j)$ is an indicator of causal dependency between i and j. It can be calculated as follows:

$$Confidence(i>j)=Support(i>j)/Support(i)\tag{8}$$

$Support(i)$ and $Support(i>j)$ are previously described in Definition 12. It is intended that the higher the $Confidence(i>j)$, the lower the penalty assigned to allowing the direct succession of $\theta\,(i)$ by $\theta\,(j)$, and the higher the penalty assigned to not allowing the direct succession of $\theta\,(i)$ by $\theta\,(j)$.Thus, the proposed formulation for $APen_{e,f}$ and $NAPen_{e,f}$ are as follows:

$$Apen_{\theta(i),\theta(j)}=\begin{cases}MConfidence(i>j)=0\\1-Confidence(i>j)0\\\quad<Confidence(i>j)<th\\0Confidence(i>j)\geq th\end{cases}\tag{9}$$

$$NAPen_{\theta(i),\theta(j)}=\begin{cases}0Confidence(i>j)<th\\MConfidence(i>j)\geq th\end{cases}\tag{10}$$

where, th is a user-defined threshold. Considering the formulation of $APen_{e,f}$ and $NAPen_{e,f}$, the first part of the objective function guarantees that only for allowing direct successions with $Confidence(i>j)$ higher than th, no penalty is included in the objective function. (Since in the dependency graph, the arcs are predetermined, allowing direct succession in the output process model is possible only through setting activities as concurrent). For allowing the other direct successions, a penalty will be included in the objective function, especially when $Confidence(i>j)$ is equal to zero, a very

high penalty is included in the objective function for allowing the direct succession of $\theta(i)$ by $\theta(j)$. The second part of the objective function ensures that for each $\theta(i), \theta(j) \in A_G$ with $Confidence(i > j)$ higher than th, the direct succession of $\theta(i)$ by $\theta(j)$ should be allowed, i.e., there should be an arc from $\theta(i)$ to $\theta(j)$, or $\theta(i)$ and $\theta(j)$ should be concurrent; otherwise, a very high penalty will be included in the objective function.

Therefore, users can determine their preference for fitness or precision by utilizing the th threshold. The higher the th, the higher the preferred precision, and the lower the preferred replay fitness.

C. DISCUSSING CONSTRAINTS

In this section, the model constraints are discussed and explained. According to the aim of the proposed model that is mentioned above, we proposed three sets of required conditions. In Section 3.5, the conditions are formulated in mathematical terms. The proposed required conditions are as follows:

Conditions Related to the Matching of AND-Splits and AND-Joins:

Condition 1 The divergent activities $a \in A_G$ and $b \in S(a)$ can be concurrent only if for them, there exist at least a pair of corresponding joined activities $c \in A_G$ and $d \in J(c)$ that are concurrent.

Condition 2 The joined activities $c \in A_G$ and $d \in J(c)$ can be concurrent only if for them, there exist at least a pair of corresponding divergent activities $a \in A_G$ and $b \in S(a)$ that are concurrent.

Conditions Related to the Accordance of Split/Join Patterns With the Activities That Are Identified as Concurrent

Condition 3 If divergent activities $a \in A_G$ and $b \in S(a)$ are concurrent and their corresponding joined activities $c \in A_G$ and $d \in J(c)$ are also concurrent, then all activities that are on a path from a to c should be concurrent with all activities that are on a path from b to d.

Condition 4 The activities $e, f \in A_G$ can be concurrent only if they have at least one pair of corresponding divergent activities $a \in A_G$ and $b \in S(a)$ that are concurrent.

Condition 5 The activities $e, f \in A_G$ can be concurrent only if they have at least one pair of corresponding joined activities $c \in A_G$ and $d \in J(c)$ that are concurrent.

Condition 6 If for the activities $e, f \in A_G$ there exist no corresponding divergent activities or corresponding joined activities, then e and f cannot be concurrent.

Condition Related to the Accordance of the Activities That Are Identified as Concurrent With Each Other

Condition 7 If for the activities $e, f \in A_G$, e is identified as concurrent with f, then f should also be identified as concurrent with e.

D. CONSTRAINTS IN MATHEMATICAL TERMS

The conditions discussed in Section 3.4 are formulated as follows:

$$\forall a, b \in A_G : b \in S(a), \forall c, d : (c, d) \in MJ(a, b) :$$
$$h_{c,d} - r_{(a,b),(c,d)} \geq 0 \tag{11}$$
$$\forall a, b \in A_G : b \in S(a) :$$
$$h_{a,b} \leq \sum_{(c,d) \in MJ(a,b)} r_{(a,b),(c,d)} \tag{12}$$
$$\forall c, d \in A_G : d \in J(c), \forall a, b : (a, b) \in MS(c, d) :$$
$$h_{a,b} - q_{(a,b)(c,d)} \geq 0 \tag{13}$$
$$\forall c, d \in A_G : d \in J(c) :$$
$$h_{c,d} \leq \sum_{(a,b) \in MS(c,d)} q_{(a,b),(c,d)} \tag{14}$$
$$\forall a, b \in A_G : b \in S(a), \forall c, d \in A_G : d \in J(c),$$
$$\forall e, f : (e, f) \in PS(a, b) \wedge (e, f) \in PJ(c, d) :$$
$$h_{e,f} \geq h_{a,b} + h_{c,d} - 1 \tag{15}$$
$$\forall e, f \in A_G, \forall a, b : (a, b) \in MS(e, f) :$$
$$h_{a,b} - u_{(a,b),(e,f)} \geq 0 \tag{16}$$
$$\forall e, f \in A_G :$$
$$h_{e,f} \leq \sum_{(a,b) \in MS(e,f)} u_{(a,b)(e,f)} \tag{17}$$
$$\forall e, f \in A_G, \forall c, d : (c, d) \in MJ(e, f) :$$
$$h_{c,d} - v_{(c,d),(e,f)} \geq 0 \tag{18}$$
$$\forall e, f \in A_G :$$
$$h_{e,f} \leq \sum_{(c,d) \in MJ(e,f)} v_{(c,d),(e,f)} \tag{19}$$
$$\forall e, f \in A_G : MS(e, f) = \emptyset \vee MJ(e, f) = \emptyset :$$
$$h_{e,f} = 0 \tag{20}$$
$$\forall e, f \in A_G :$$
$$h_{e,f} = h_{f,e} \tag{21}$$
$$\forall a, b, c, d, e, f \in A_G :$$
$$Binary : h_{a,b}, r_{(a,b),(c,d)}, q_{(a,b),(c,d)}, u_{(a,b),(e,f)}, v_{(c,d),(e,f)} \tag{22}$$

Equations (11) and (12) ensure that Condition (1) is met, while; (13) and (14) guarantee that Condition (2) is met. Equation (15) pertains to the trueness of Condition (3). Meeting Condition (4) is guaranteed by (16) and (17), whereas (18) and (19) guarantee meeting Condition (5). Equations (20) and (21) pertain respectively to the trueness of Conditions (6) and (7), and finally, (22) determines the types of the variables.

V. CONVERTING THE ILP MODEL VARIABLES TO CAUSAL NET INPUT/OUTPUT BINDINGS

The proposed ILP model determines the optimal combination of concurrent activities considering several conditions. However, to make the ILP model result applicable, it should be converted to the split/join patterns representation of a process modeling notation. This allows converting the input

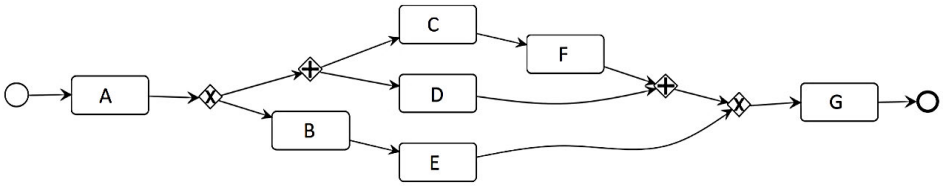


FIGURE 5. The result of applying the proposed method to determine the split/join patterns of the dependency graph presented in Fig. 1 (to make the result more visually understandable, the output process model was converted to BPMN notation).

dependency graph to a process model. For this purpose, we decided to convert the result of the ILP model to the split/join patterns representation of Causal nets since it is the most popular process modeling notation among heuristic mining methods. According to Definition 2, in this notation, the split/join patterns are presented as input/output bindings of activities. Therefore, this section proposes a method to infer the input/output bindings from the identified concurrent activities. By adding the input/output bindings to the input dependency graph, the dependency graph is simply converted to a Causal net.

The proposed method for converting the ILP model results to Causal net input/output bindings is similar to the technique used by Heuristics Miner. Heuristics Miner method identifies the concurrent divergent/joined activities and then based on it; the split/join patterns are built. This is comparable to the approach that we intend to utilize; however, the Heuristics Miner representation for split/join patterns is somehow different from Causal net notation. Therefore prior to applying the procedure used by Heuristics Miner, we need to modify it to make it able to build Causal net input/output bindings. The proposed modified procedure is presented in Algorithm 1.

Supposing the dependency graph and the event log presented in Fig. 1 as the inputs of the proposed ILP model, Fig. 5 shows the process model achieved after applying Algorithm 1 to the outputs of the ILP model. To make the result more visually understandable, the output process model was converted to BPMN notation. It can be seen that in contrast to the results of Heuristics Miner, Flexible Heuristics Miner, and Fodina methods, the process model obtained by the proposed method cannot replay the behaviors that are infrequent or not observed in the event log. Therefore, the result of the proposed method has higher precision than the mentioned methods. In addition, for the AND-split in the process model, there is a matching AND-join and vice versa. Hence the proposed method can handle the previously mentioned deficiencies that exist in the current methods.

Algorithm 1 The Modified Version of the Procedure Used by Heuristics Miner for Building Input/Output Bindings From Activities Identified as Concurrent and the Input Dependency Graph

```
Input: dependency graph G = (A_G, D_G),
    and the optimal values of h_{a,b} : ∀a, b ∈ A_G
Output: I(a), O(a) : ∀a ∈ A_G
For each a ∈ A_G do:
    For dir ∈ {InBindings, OutBindings} do:
        B(a) ← {}
        If dir = InBindings then:
            X ← {b ∈ A_G | (b, a) ∈ D_G}
        If dir = OutBindings then:
            X ← {b ∈ A_G | (a, b) ∈ D_G}
        For index ∈ {1, 2} do:
            For x_i ∈ X do:
                flag1 ← 0
                For B_j(a) ∈ B(a) do:
                    flag2 ← 1
                    For a_k ∈ B_j(a) do:
                        If h_{x_i,a_k} = 0 then:
                            flag2 ← 0
                    If flag2 = 1 then:
                        B_j(a) ← B_j(a) ∪ {x_i}
                        flag1 ← 1
                    If flag1 = 0 then:
                        B(a) ← B(a) ∪ {{x_i}}
        If dir = InBindings then:
            I(a) ← B(b)
        If dir = OutBindings then:
            O(a) ← B(a)
```

VI. EXPERIMENTAL RESULTS

This section presents the evaluations made to assess the proposed method. To the best of our knowledge, determining the split/join patterns of dependency graphs has been done only as a step of heuristic mining process model discovery. Therefore, in this paper, we compare the performance of the proposed method with the methods of detecting the dependency graph split/join patterns used by the most prominent heuristic mining algorithms, including Heuristics Miner (HM), Flexible Heuristics Miner (FHM), and Fodina.

To make the assessments, 27 event logs were utilized which their specifications are presented in Table 3. Rows 1 to 12 of the table correspond to all publicly available event logs used in the experiments performed by the review and benchmark paper [22]. This set of event logs consists of 10 BPI Challenge (BPIC) event logs (called $\text{BPIC}Y_X$, where Y is the year that the related challenge of the event log took place), as well as "*real events in a hospital for patients with sepsis*" (called SepsisCases) [23] and "*road traffic fine management process)*" (called RTFMP) [24] event logs.

Moreover, the event log of "*BPI Challenge 2011*" [25] was also used in the experiments of this section. This event log consists of the actual events that occurred for the patients in a Dutch hospital. However, the event log is very heterogeneous; hence, according to the suggestion of the challenge

winner [26], the event log was decomposed into some smaller logs by filtering the original log according to the ''diagnosis code'' attribute of instances. There were some infrequent activities in the obtained event logs, and even some activities appeared only once. Thus, using a ProM6.6 plugin, namely ''Filter Log Using Simple Heuristics'', 2% of the less frequent activities were removed for each obtained event log. The achieved event logs are called $BPIC11_X$(rows 13 to 20 of Table 3), where $BPIC11_X$ is derived from the event log obtained by considering only the events that took place for the patients with ''diagnosis code=X''. For instance, $BPIC11_{M16}$ is the event log achieved after removing 2% of the less frequent activities of the events that occurred for the patients with ''diagnosis code=M13''.

Four other real event logs were also utilized, including: ''*event log obtained from the financial modules of the ERP system of a regional hospital*'' (called HospBill) [27], ''*Production''* (called production) [28], ''*Receipt phase of an environmental permit application process''* (called Receipt) [29], and ''*edited_hh104_labour* (from *Activities of daily living of several individuals* dataset)'' (called DailyActs) [30] (row 21 to 24 of Table 3). In addition, to further test the performance of the proposed method in dealing with highly noisy event logs, three synthetic event logs that have been used by [8] were also utilized (rows 25 to 27 of Table 3). The event logs are called ''randsAlBmCaD'', where, B and C are respectively, average length and standard deviation of the event log traces. The traces are created by random selection of activities out of a set of activities with size D.

Hence, the event logs are in different sizes and cover a wide area of applications such as finance, production, IT management, healthcare, daily activities of people, and government services. All real-world event logs are publicly available at 4TU Centre for Research Data.[1]

In order to make fair comparisons for each event log, it is required that the same dependency graphs be used as the input of all methods. Hence, in the first step, after adding artificial initial/final activity to all traces of the event logs without unique initial/final activity, the method used by Heuristics Miner for dependency graph construction was applied to achieve dependency graphs for each event log. This was done using the ''Mine for a Heuristic Net Using Heuristic Miner'' plugin for ProM 6.4 software. To perform evaluations with more input data for each event log, two dependency graphs were constructed. For this purpose, two different configurations of the HM algorithm were employed.

The first and second dependency graphs are achieved by setting the ''*dependency threshold*'' parameter to respectively 80 and 100. These values were used because the dependency threshold levels lower than 80 generally had not enough efficiency in dealing with noises; while, the dependency graphs constructed by using the dependency threshold levels between 80 and 100 were usually identical with the dependency graphs achieved by setting the dependency threshold to 80 or 100. For event log E, the dependency graphs achieved by setting the dependency threshold parameter to 80 and 100 are called E(dt80) and E(dt100) respectively.

TABLE 3. Characteristics of the event logs utilized in the experiments.

No.	Event Log	Total number of traces	Total number of events	Number of activities
1.	BPIC12	13,087	262,200	36
2.	$BPIC13_{cp}$	1,487	6,660	7
3.	$BPIC13_{inc}$	7,554	65,553	13
4.	$BPIC14_{f}$	41,353	369,485	9
5.	$BPIC15_{1f}$	902	21,656	70
6.	$BPIC15_{2f}$	681	24,678	82
7.	$BPIC15_{3f}$	1,369	43,786	62
8.	$BPIC15_{4f}$	860	29,403	65
9.	$BPIC15_{5f}$	975	30,030	74
10.	$BPIC17_{f}$	21,861	714,198	41
11.	RTFMP	150,370	561,470	11
12.	Sepsis	1,050	15,214	16
13.	$BPIC11_{106}$	113	6149	155
14.	$BPIC11_{821}$	29	13572	118
15.	$BPIC11_{822}$	22	4955	143
16.	$BPIC11_{839}$	14	5380	96
17.	$BPIC11_{M11}$	60	4961	99
18.	$BPIC11_{M13}$	195	23647	138
19.	$BPIC11_{M14}$	128	6075	135
20.	$BPIC11_{M16}$	1050	14012	112
21.	DailyActs	43	2,100	19
22.	HospBill	100,000	451,359	18
23.	Production	225	4,543	55
24.	Receipt	1434	8577	27
25.	rands100l5m2a3	100	665	3
26.	rands1000l10m4a5	1000	11574	5
27.	rands10000l20m8a10	10000	215259	10

For each of the dependency graphs, the different methods were applied to determine split/join patterns and convert the dependency graph to a process model (For the proposed method, the *th* threshold was set to 0.3). Then, the obtained process models were compared through different experiments considering three different dimensions in assessing the quality of process models consisting of replay fitness, precision, and simplicity.

The replay fitness and precision have been described in Section 3.3. To evaluate the replay fitness of process models, the ''*ContinuosParsingMeasure (CPM)*'' measure was utilized and implemented as it is described in [4]. This measure is among the most well-known methods in assessing the replay fitness of Causal/Heuristic nets. However, since there is no prominent method for evaluating the precision of Causal/Heuristic nets to the best of our knowledge, we converted the obtained process models to their equivalent Petri nets prior to assessing their precision. Then, the precision measure proposed by [31] was utilized to evaluate the precision of obtained Petri nets. This method is popular in the literature and has been cited by many studies.

In the experiments, at first, all attained process models were saved in ''.flex'' format. Next, by means of the ''Convert Flexible Models to Petri Nets'' plugin for ProM6.6, the equivalent Petri nets for the process models were achieved, and their extra invisible transitions were removed by using the ''Reduce Silent Transitions'' plugin for ProM 6.6. Finally,

[1] https://data.4tu.nl/

TABLE 4. The quality measures obtained by the methods for each input dependency graph and its corresponding event log.

	F-Score(CPMF,ETCP)				CFC				NMAND			
	HM	FHM	Fodina	Proposed Method	HM	FHM	Fodina	Proposed Method	HM	FHM	Fodina	Proposed Method
BPIC12(dt80)	**0.77(0.89,0.67)**	0.60(0.93,0.45)	0.57(0.92,0.41)	0.76(0.90,0.65)	112	143	**98**	101	16	122	6	**0**
BPIC12(dt100)	0.64(0.86,0.51)	0.60(0.88,0.46)	0.81(0.86,0.77)	**0.90(0.85,0.94)**	62	64	**33**	36	2	52	4	**0**
$BPIC13_{cp}$(dt80)	0.74(0.93,0.61)	0.32(1.00,0.19)	0.74(0.96,0.60)	**0.86(0.81,0.92)**	19	50	18	**15**	4	24	6	**0**
$BPIC13_{cp}$(dt100)	0.75(0.92,0.63)	0.40(0.97,0.25)	0.82(0.94,0.74)	**0.85(0.81,0.89)**	17	36	16	**14**	4	26	6	**0**
$BPIC13_{inc}$(dt80)	**0.89(0.91,0.86)**	0.29(0.98,0.17)	0.60(0.96,0.43)	0.80(0.89,0.73)	39	120	42	**38**	4	72	12	**0**
$BPIC13_{inc}$(dt100)	0.70(0.94,0.55)	0.28(0.97,0.16)	0.57,0.96,0.41)	**0.79(0.89,0.72)**	45	94	35	**31**	4	36	4	**0**
$BPIC14_{f}$(dt80)	0.28(0.95,0.17)	0.20(0.95,0.11)	0.41(0.93,0.26)	**0.81(0.87,0.45)**	61	70	32	**21**	24	14	10	**0**
$BPIC14_{f}$(dt100)	0.26(0.89,0.15)	0.18(0.92,0.10)	0.41(0.89,0.26)	**0.78(0.84,0.73)**	53	67	29	**17**	18	14	10	**0**
$BPIC15_{1f}$(dt80)	0.86(1.00,0.76)	**0.87(1.00,0.78)**	**0.87(1.00,0.78)**	**0.87(1.00,0.78)**	90	**89**	**89**	**89**	12	**0**	**0**	**0**
$BPIC15_{1f}$(dt100)	**0.93(0.89,0.97)**	0.90(0.90,0.91)	**0.93(0.89,0.97)**	**0.93(0.89,0.97)**	39	47	**33**	**33**	4	14	4	**0**
$BPIC15_{2f}$(dt80)	0.74(0.99,0.59)	0.70(1.00,0.54)	0.72(0.99,0.57)	**0.75(0.99,0.60)**	156	166	150	**149**	10	20	**0**	**0**
$BPIC15_{2f}$(dt100)	0.83(0.86,0.81)	0.71(0.87,0.60)	0.85(0.85,0.84)	**0.86(0.85,0.87)**	52	51	34	**33**	8	22	4	**0**
$BPIC15_{3f}$(dt80)	**0.79(0.97,0.67)**	0.72(0.99,0.56)	0.51(0.97,0.34)	**0.79(0.97,0.66)**	153	187	156	**151**	0	48	14	**0**
$BPIC15_{3f}$(dt100)	0.83(0.87,0.79)	0.83(0.87,0.78)	0.79(0.85,0.73)	**0.90(0.85,0.97)**	47	46	30	**28**	4	24	10	**0**
$BPIC15_{4f}$(dt80)	0.63(0.98,0.46)	**0.73(1.00,0.58)**	0.71(1.00,0.55)	0.72(1.00,0.57)	127	131	124	**123**	8	10	4	**0**
$BPIC15_{4f}$(dt100)	**0.87(0.85,0.88)**	0.82(0.86,0.79)	0.85(0.85,0.85)	0.86(0.85,0.88)	36	42	**28**	29	2	14	8	**0**
$BPIC15_{5f}$(dt80)	0.65(1.00,0.49)	**0.80(1.00,0.67)**	**0.80(1.00,0.67)**	**0.80(1.00,0.67)**	124	**123**	**123**	**123**	2	**0**	**0**	**0**
$BPIC15_{5f}$(dt100)	0.90(0.88,0.93)	0.72(0.89,0.61)	0.75(0.88,0.65)	**0.91(0.87,0.94)**	33	42	**27**	28	6	30	4	**0**
$BPIC17_{f}$(dt80)	0.81(0.95,0.71)	0.77(0.98,0.63)	0.61(0.98,0.44)	**0.88(0.94,0.82)**	85	94	73	**72**	8	34	10	**0**
$BPIC17_{f}$(dt100)	0.77(0.90,0.68)	0.75(0.93,0.62)	0.43(0.92,0.28)	**0.91(0.88,0.93)**	66	68	**38**	**38**	6	40	10	**0**
RTFMP(dt80)	0.62(0.94,0.46)	0.49(0.99,0.32)	**0.94(0.98,0.90)**	0.89(0.88,0.91)	31	37	**22**	**22**	18	32	2	**0**
RTFMP(dt100)	0.73(0.95,0.59)	0.71(0.95,0.57)	**0.97(0.93,1.00)**	0.91(0.84,1.00)	15	17	**9**	**9**	0	4	0	0
Sepsis(dt80)	**0.72(0.96,0.58)**	0.66(0.99,0.49)	0.06(0.97,0.03)	0.28(0.96,0.16)	38	44	**24**	26	10	40	12	0
Sepsis(dt100)	**0.68(0.93,0.53)**	0.60(0.96,0.43)	0.04(0.94,0.02)	0.67(0.91,0.53)	32	36	**16**	20	6	20	6	0
$BPIC11_{106}$(dt80)	0.59(0.87‹0.45)	0.30(0.90‹0.18)	0.74(0.87‹0.65)	**0.75(0.87‹0.66)**	276	490	211	**198**	6	63	4	**0**
$BPIC11_{106}$(dt100)	0.59(0.84‹0.46)	0.33(0.87‹0.20)	0.77(0.85‹0.71)	**0.78(0.84‹0.74)**	267	431	197	**177**	0	35	1	**0**
$BPIC11_{821}$(dt80)	0.31(0.93‹0.18)	0.06(0.96‹0.03)	0.43(0.94‹0.28)	**0.45(0.93‹0.30)**	306	679	231	**217**	4	194	4	**0**
$BPIC11_{821}$(dt100)	**0.66(0.90‹0.52)**	0.09(0.95‹0.05)	0.65(0.92‹0.50)	0.65(0.90‹0.50)	261	560	179	**166**	0	104	3	**0**

to apply the mentioned precision measure, the ''Check Conformance Using ETC Conformance'' plugin for ProM6.6 was employed. The mentioned replay fitness and precision measures are called *CPMF* and *ETCP* in the rest of this paper. However, as mentioned before, replay fitness and precision are complementary measures and a process model with high replay fitness and low precision is not a suitable descriptor for the event log; this is also true of a process model with high precision but low replay fitness. Hence, as is prevalent in the literature, instead of using both replay fitness and precision, the process models were compared according to the F-score measure, which is the harmonic mean of *CPMF* and *ETCP*. The F-score of a process model takes a very low value if only one of the *CPMF* or *ETCP* measures of the process model is very low, even when the other measure is very high. Using the F-score measure allows simultaneous comparison of the replay fitness and the precision of outputs by only one measure.

The other quality dimension that was assessed in the experiments was simplicity. According to the simplicity dimension, the process models should be as simple as possible. In other words, the simplest process model that can properly describe the behaviors in the event log is desired [1]. Since the same dependency graphs were used as the input of all methods, the Control Flow Complexity (CFC) [32] measure was utilized to compare the simplicity of the process models. This measure assesses the complexity of process models based on their split patterns; hence, it is suitable for our experiments. Obviously,

TABLE 4. *(Continued.)* **The quality measures obtained by the methods for each input dependency graph and its corresponding event log.**

	F-Score(CPMF,ETCP)				CFC				NMAND			
	HM	FHM	Fodina	Proposed Method	HM	FHM	Fodina	Proposed Method	HM	FHM	Fodina	Proposed Method
BPIC11$_{822}$(dt80)	0.53(0.92,0.37)	0.19(0.94,0.11)	0.55(0.92,0.39)	**0.56(0.92,0.41)**	266	414	186	**181**	4	80	2	**0**
BPIC11$_{822}$(dt100)	0.54(0.90,0.39)	0.23(0.93,0.13)	0.57(0.91,0.41)	**0.58(0.90,0.43)**	259	372	178	**171**	4	78	4	**0**
BPIC11$_{839}$(dt80)	0.50(0.93,0.34)	0.12(0.96,0.07)	0.49(0.94,0.33)	**0.50(0.93,0.34)**	198	382	147	**139**	**0**	4	4	**0**
BPIC11$_{839}$(dt100)	0.57(0.91,0.41)	0.15(0.94,0.08)	0.57(0.92,0.41)	**0.57(0.91,0.41)**	178	325	126	**114**	**0**	22	**0**	**0**
BPIC11$_{M11}$(dt80)	0.36(0.89,0.23)	0.58(0.91,0.42)	0.57(0.89,0.42)	**0.61(0.89,0.47)**	191	323	136	**132**	23	34	4	**0**
BPIC11$_{M11}$(dt100)	0.54(0.87,0.39)	0.25(0.89,0.14)	0.73(0.87,0.63)	**0.74(0.87,0.65)**	171	279	126	**119**	9	24	2	**0**
BPIC11$_{M13}$(dt80)	0.35(0.92,0.21)	0.05(0.96,0.03)	0.51(0.93,0.35)	**0.53(0.90,0.38)**	397	1029	288	**266**	48	88	8	**0**
BPIC11$_{M13}$(dt100)	0.48(0.89,0.33)	0.09(0.93,0.05)	0.55(0.90,0.40)	**0.61(0.89,0.47)**	308	699	204	**183**	14	62	10	**0**
BPIC11$_{M14}$(dt80)	0.47(0.87,0.32)	0.16(0.90,0.09)	0.58(0.88,0.44)	**0.61(0.87,0.47)**	280	414	188	**174**	9	78	1	**0**
BPIC11$_{M14}$(dt100)	0.57(0.84,0.43)	0.18(0.87,0.10)	0.61(0.85,0.48)	**0.63(0.84,0.51)**	256	377	168	**155**	5	11	3	**0**
BPIC11$_{M16}$(dt80)	0.34(0.92,0.21)	0.10(0.94,0.05)	**0.48(0.93,0.33)**	**0.48(0.91,0.32)**	294	554	200	**189**	1	10	**0**	**0**
BPIC11$_{M16}$(dt100)	0.58(0.88,0.43)	0.17(0.91,0.10)	0.62(0.89,0.48)	**0.64(0.88,0.50)**	242	403	152	**143**	4	1	4	**0**
DailyActs(dt80)	**0.44(0.88,0.29)**	0.12(0.94,0.06)	0.35(0.90,0.21)	**0.44(0.88,0.29)**	78	240	80	**62**	0	136	28	**0**
DailyActs(dt100)	**0.44(0.88,0.29)**	0.12(0.94,0.06)	0.35(0.90,0.21)	**0.44(0.88,0.29)**	77	243	80	**61**	0	136	28	**0**
HospBill(dt80)	0.84(0.98,0.74)	0.37(1.00,0.23)	**0.94(0.99,0.89)**	**0.94(0.99,0.89)**	70	145	**61**	**61**	38	126	**0**	**0**
HospBill(dt100)	0.64(0.86,0.51)	0.24(0.91,0.14)	0.59(0.90,0.44)	**0.87(0.86,0.89)**	71	92	**36**	**36**	10	74	6	**0**
Production(dt80)	0.41(0.89,0.27)	0.11(0.92,0.06)	0.47(0.90,0.32)	**0.87(0.84,0.91)**	195	302	133	**108**	64	100	50	**0**
Production(dt100)	0.42(0.89,0.27)	0.11(0.92,0.06)	0.48(0.90,0.33)	**0.87(0.84,0.91)**	197	302	133	**108**	64	100	50	**0**
Receipt(dt80)	**0.96(0.97,0.95)**	0.46(1.00,0.30)	0.85(0.99,0.74)	**0.96(0.97,0.95)**	31	58	27	**26**	12	42	4	**0**
Receipt(dt100)	**0.84(0.98,0.73)**	0.62(0.98,0.45)	0.82(0.98,0.70)	0.82(0.98,0.70)	31	48	**22**	**22**	0	16	0	**0**
rands100l5m2a3(dt80)	0.52(1.00,0.35)	0.52(1.00,0.35)	0.59(0.97,0.42)	**0.84(0.89,0.79)**	17	17	12	**9**	**0**	**0**	**0**	**0**
rands100l5m2a3(dt100)	0.52(1.00,0.35)	0.52(1.00,0.35)	0.59(0.97,0.42)	**0.84(0.89,0.79)**	17	17	12	**9**	**0**	**0**	**0**	**0**
rands1000l10m4a5(dt80)	0.23(1.00,0.13)	0.23(1.00,0.13)	0.21(0.99,0.12)	**0.86(0.86,0.85)**	50	50	22	**15**	**0**	**0**	**0**	**0**
rands1000l10m4a5(dt100)	0.23(1.00,0.13)	0.23(1.00,0.13)	0.21(0.99,0.12)	**0.86(0.86,0.85)**	50	50	22	**15**	**0**	**0**	**0**	**0**
rands10000l20m8a10(dt80)	0.11(1.00,0.06)	0.11(1.00,0.06)	0.15(0.99,0.08)	**0.87(0.82,0.92)**	784	784	132	**30**	**0**	**0**	**0**	**0**
rands10000l20m8a10(dt100)	0.11(1.00,0.06)	0.11(1.00,0.06)	0.15(0.99,0.08)	**0.87(0.82,0.92)**	784	784	132	**30**	**0**	**0**	**0**	**0**

the lower the *CFC*, the higher the simplicity of process models, and vice versa.

In addition to the measures mentioned above, for each obtained process model, the number of cases each pair of concurrent divergent/joined activities have no matching pairs of concurrent joined/divergent activities was also calculated (called *NMAND*). To further assessments, in our experiments, the *NMAND* measures attained for each method were also compared. The quality measures obtained by each method for each input dependency graph and its corresponding event log are presented in Table 4. For each input, the best-obtained measure is boldfaced. According to the table, generally, the proposed method outperformed the other methods in terms of *F-score*, *CFC*, and *NMAND* measures. For each input, considering the outputs with the best result in all terms of *F-score*, *CFC*, and *NMAND* measures as the overall best output, the cases in which the proposed method attained the overall best output are highlighted in gray. Accordingly, the proposed method obtained the overall best output for 42 out of 54 cases (i.e., 78%). This is an impressive performance, especially when it is considered that HM, FHM, and Fodina methods attained the overall best output in respectively 0, 2, and 4 cases (i.e., 0%, 4%, and 7%). Therefore, in this regard, there is a huge gap in the performance of the proposed method with the performance of the other methods.

The number of cases that each method attained the best/second-best result for a quality measure is presented in Table 5. In addition, for each input and quality measure, the distance between the measure attained by each method and the best-attained measure was calculated. For each method, the averages of the distances are presented in Table 6. In both Tables 5 and 6, the best result for each measure is boldfaced.

Accordingly, the FHM method showed the best performance in attaining the outputs with high CPMF, since it

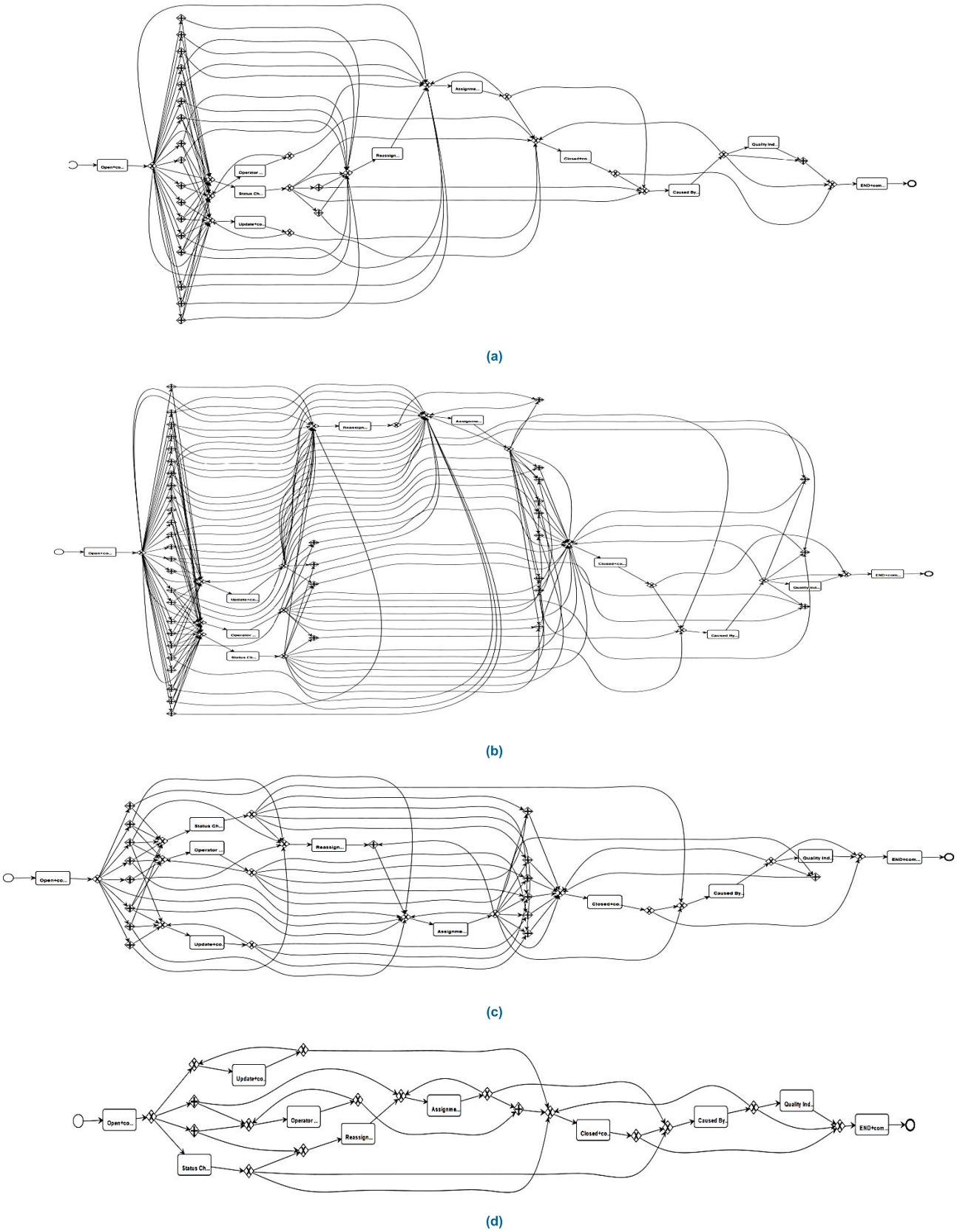


FIGURE 6. The process models attained by applying (a) Heuristics Miner, (b) Flexible Heuristics Miner, (c) Fodina, (d) the proposed method to BPIC14$_f$ (dt80) dependency graph.

TABLE 5. The frequency of the cases each method achieved the best/second-best quality measure.

		CPMF	ETCP	F-score	CFC	NMAND
The frequency of the cases achieved the best quality measure	HM	12	14	14	0	15
	FHM	**54**	3	3	2	8
	Fodina	5	6	7	15	14
	Proposed method	4	**46**	**43**	**48**	**54**
The frequency of the cases achieved the second-best quality measure	HM	14	10	10	4	14
	FHM	0	4	5	0	1
	Fodina	**23**	**25**	**25**	**35**	**24**
	Proposed method	5	7	9	6	0
Total	HM	26	24	24	4	29
	FHM	**54**	7	8	2	9
	Fodina	28	31	32	50	38
	Proposed method	9	**53**	**52**	**54**	**54**

TABLE 6. The average distance of the quality measures obtained by each method from the best-obtained measures.

	CPMF	ETCP	F-score	CFC	NMAND
HM	0.03	0.22	0.17	66.41	9.24
FHM	**0.00**	0.40	0.36	127.79	43.15
Fodina	0.02	0.22	0.17	10.22	6.78
Proposed method	0.06	**0.01**	**0.01**	**0.26**	**0.00**

obtained the best CPMF measure for all cases. However, the FHM method achieved this performance at the cost of attaining the *ETCP*, *F-score*, *CFC*, and *NMAND* measures far lower than the other methods. Especially, the F-score measures achieved by the FHM method were generally lower than the result of the other methods and had the largest average distance to the best result. It means that at the cost of obtaining precision measures significantly lower than the other methods, the FHM method achieved fitness measures slightly higher than the other methods. Hence, despite attaining high fitness, the outputs achieved by the FHM method were not a good descriptor for their corresponding event logs.

On the other hand, the proposed method showed the worst performance for the *CPMF* measure. Nevertheless, according to Table 6, on average, the proposed method attained *CPMF* measures 0.06 lower than the best value, whereas it showed the best performance for *ETCP*, *F-score*, *CFC*, and *NMAND* measures in both terms of frequency of attaining the best result and average distance from the best result. In both terms, there was a large gap between the performance of the proposed method and the other methods. Thus, at the cost of a slight reduction in *CPMF* measure, the proposed method successfully attained far higher *ETCP*, *F-score*, *CFC*, *NMAND* measures. Especially, the F-score measures attained by the proposed method had the lowest average distance from the best result, and they were generally higher than the other method. Therefore, compared to the other methods, the slightly lower *CPMF* measures achieved by the proposed method are justifiable because this led to achieving far higher *ETCP* measures. Hence, considering all measures of replay fitness, precision, simplicity, and matching of AND-splits and AND-joins, the proposed method showed a significantly better performance than the other methods. To enable a visual comparison of the results of each method, as an instance, the process model obtained by applying each method to

TABLE 7. Some statistics about the problem-solving times.

Maximum	Minimum	Median	Standard deviation
202.92	0.00	11	51.03

BPIC14$_f$(dt80) dependency graph is presented in Fig. 6. The results were converted to BPMN notation to make the results more visually understandable. Accordingly, it can be seen that the result of the proposed method enjoys far less complexity than the results of the other methods and, in consequence, is more understandable; At the same time, according to Table 4, it has the highest F-score.

Finally, since the application of ILP may raise a concern about the problem-solving time, Table 7 presents some statistics about the time spent on solving the proposed ILP model for the above-mentioned real-world inputs. All experiments were performed on a PC with Intel(R) Core(TM) i5 CPU @2.40GHz and 8GB RAM.

According to Table 7, in the worst case, the problem was solved in 202.92 seconds which can be considered as an acceptable time, especially considering the fact that some of the utilized event logs are of large size (in terms of the number of unique activities, events, and traces). The median and minimum of problem-solving times were far lower. The real-world event logs used in the experiments of this paper cover a wide range of application areas and event log sizes; thus, it is anticipated that the ILP model can be solved in a short or at least acceptable time for most real-world applications.

VII. CONCLUSION AND FUTURE WORKS

The Heuristic mining methods are among the most widely used methods of process discovery due to several advantages,

such as their ability to deal with noises and unstructured processes. This category of methods is mainly composed of two main steps 1) constructing dependency graph, 2) determining the split/join patterns of the dependency graph. There are some methods in the literature to mine the split/join patterns of dependency graphs; nevertheless, all of them use only the event log succession information about the activities involved in the splits/joins. Whereas the types of splits/joins determine whether the activities on the paths between splits and joins can occur concurrently or not. In addition, existing methods determine the split/join patterns in a local manner, and when detecting the pattern of a split/join, the effect it has on the overall quality of the process model is neglected. Therefore, the result of the existing methods is prone to be non-optimal.

On the other hand, the existing methods cannot guarantee that there is at least a matching AND-join for each AND-split, and vice versa. Consequently, some of the split/join patterns can be unable to be activated. To address the mentioned issues, in this paper, for the first time, an ILP model for determining the split/join patterns of dependency graphs is introduced, which optimizes the split/join patterns regarding all successions that exist in the event log. The proposed objective function is inspired by two of the process model quality dimensions named replay fitness and precision. At the same time, the constraints of the ILP model ensure that the dependency graph AND-splits match the AND-joins. Furthermore, by means of introducing appropriate constraints, the proposed ILP model can be made more flexible, and it can also be capable of using domain knowledge. The input of the proposed ILP model is a dependency graph and its corresponding event log, whereas its output is the activities identified as concurrent. To convert the outputs of the ILP model to the split/join patterns, a modified version of the method has been used by the Heuristics Miner algorithm is employed.

The proposed method is evaluated against the most prominent methods of determining split/join patterns of dependency graphs. Accordingly, it outperformed the other methods in terms of fitness, precision, simplicity, and the matching of AND-splits and AND-joins.

Some constraints can be introduced in future works to make the method more flexible. In addition, some constraints for utilizing domain knowledge can also be introduced. Another potential research topic can be improving the technique employed for converting the ILP model outputs to split/join patterns.

SUPPLEMENTARY MATERIAL

In this paper, GAMS and MATLAB were employed to implement the proposed method and perform experiments. The codes utilized in this study are available in the following repository: https://github.com/MaTavakoli/ ILP_DetermineSplitJoinPatterns.git, to allow other researchers to utilize and make comparisons in future studies.

Algorithm 2 The Proposed Procedure to Remove Loop Closure Arcs From the Original Input Dependency Graph DG

Input: dependency graph $DG = (A_G, D_G)$,
the parameters $P_{a,b} : \forall a, b \in A_G$ which represents paths in DG, and a_s as the initial activity of DG
Output: $LoopFreeD_G$
$DistFromInitial \leftarrow DetecDistFromInitial(A_G, D_G, a_s)$
$LoopFreeD_G \leftarrow D_G$
For $(a, b) \in A_G \times A_G$ **do**:
  If $P_{a,b} = 1$ **and** $P_{b,a} = 1$ **then**:
    If $DistFromInitial(a) > DistFromInitial(b)$ **then**:
      $LoopFreeD_G(a, b) \leftarrow 0$
Function $DetectDistFromInitial\ (A_G, D_G, a_s)$
  $DistFromInitial\ (a_s) \leftarrow 0$
  $DeterminedDistSet \leftarrow \{a_s\}$
  $CurrentDistSet \leftarrow \{a_s\}$
  $CurrentDist \leftarrow 0$
  While $DeterminedDistSet \neq A_G$ **do**:
    $CurrentDist \leftarrow CurrentDist + 1$
    $NewCurrentDistSet \leftarrow \{\}$
    For $a \in CurrentDistSet$ **do**:
      $X \leftarrow \{b \in A_G | (a, b) \in D_G\} - DeterminedDistSet$
      For $x_i \in X$ **do**:
        $DistFromInitial(x_i) \leftarrow CurrentDist$
        $NewCurrentDistSet \leftarrow NewCurrentDistSet \cup \{x_i\}$
        $DeterminedDistSet \leftarrow DeterminedDistSet \cup \{x_i\}$
    $CurrentDistSet \leftarrow NewCurrentDistSet$
Return $DistFromInitial$

APPENDIX: APPLIED PROCEDURE TO DERIVE THE LOOP-FREE GRAPH FROM THE ORIGINAL DEPENDENCY GRAPH

This appendix describes the procedure applied for deriving the loop-free graph from the original dependency graph. For this purpose, an assumption is made that all activities are reachable from the initial activity of the dependency graph. This assumption is also included in the definitions of Causal nets and dependency graphs (Definitions 2 and 3). If there is no unique initial activity in the event log, it can be added artificially to all traces in the event log, and then the dependency graph can be constructed.

In the proposed procedure at the first step, the minimum distance of each activity $a \in A_G$ from the initial activity of DG (a_s) is calculated. Next, the existence of paths between activities in the original dependency graph is detected by Warshall's Algorithm. For activities $a, b \in A_G$, $P_{a,b}$ is equal to one if in DG there is a path from a to b; otherwise, it is equal to zero. If for activities $a, b \in A_G$, both $P_{a,b}$ and $P_{b,a}$ are equal to one, it means that in DG, a and b are involved in a loop. We assume that the origin activity of each loop closure arc is more distant from the initial activity than the destination activity of the arc. Hence, in the final step, whenever two activities are identified as being involved in a loop, and there is/are arcs/arcs between them, the proposed procedure removes the arc that its origin activity is more

distant from the initial activity than its destination activity (if there exist such an arc in DG). Supposing the new set of arcs as D'_G,the loop-free graph is $DG' = (A_G, D'_G)$. The pseudo-code of the proposed procedure is presented in

Algorithm 2. Here again, the existence of paths between activities in $DG' = (A_G, D'_G)$ can be detected by Warshall's method.

REFERENCES

[1] W. M. P. Aalst, *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. Berlin, Germany: Springer-Verlag, 2011.

[2] C. D. S. Garcia, A. Meincheim, E. R. Faria Junior, M. R. Dallagassa, D. M. V. Sato, D. R. Carvalho, E. A. P. Santos, and E. E. Scalabrin, "Process mining techniques and applications—A systematic mapping study," *Expert Syst. Appl.*, vol. 133, pp. 260–295, Nov. 2019, doi: 10.1016/j.eswa.2019.05.003.

[3] E. Rojas, J. Munoz-Gama, M. Sepúlveda, and D. Capurro, "Process mining in healthcare: A literature review," *J. Biomed. Inform.*, vol. 61, pp. 224–236, Jun. 2016, doi: 10.1016/j.jbi.2016.04.007.

[4] A. Weijters, W. M. P. Aalst, and A. Medeiros, "Process mining with the Heuristics Miner-algorithm," Eindhoven Univ. Technol., Eindhoven, The Netherland. Tech. Rep. BETA publicatie: Working Papers, Jan. 2006, vol. 166. [Online]. Available: https://research.tue.nl/en/publications/process-mining-with-the-heuristicsminer-algorithm

[5] A. J. M. M. Weijters and J. T. S. Ribeiro, "Flexible Heuristics Miner (FHM)," in *Proc. IEEE Symp. Comput. Intell. Data Mining (CIDM)*, Paris, France, Apr. 2011, pp. 310–317, doi: 10.1109/CIDM.2011.5949453.

[6] A. Burattin, A. Sperduti, and W. M. P. van der Aalst, "Heuristics miners for streaming event data," 2012, *arXiv:1212.6383*.

[7] A. Burattin, "Heuristics Miner for time interval," in *Process Mining Techniques in Business Environments: Theoretical Aspects, Algorithms, Techniques and Open Challenges in Process Mining*, A. Burattin, Ed. Cham, Switzerland: Springer, 2015, pp. 85–95.

[8] S. K. L. M. vanden Broucke and J. De Weerdt, "Fodina: A robust and flexible heuristic process discovery technique," *Decis. Support Syst.*, vol. 100, pp. 109–118, Aug. 2017, doi: 10.1016/j.dss.2017.04.005.

[9] A. Augusto, R. Conforti, M. Dumas, M. La Rosa, and G. Bruno, "Automated discovery of structured process models: Discover structured vs. discover and structure," in *Conceptual Modeling*, I. Comyn-Wattiau, K. Tanaka, I.-Y. Song, S. Yamamoto, and M. Saeki, Eds. Cham, Switzerland: Springer, 2016, pp. 313–329.

[10] A. Augusto, R. Conforti, M. Dumas, M. L. Rosa, and G. Bruno, "Automated discovery of structured process models from event logs: The discover-and-structure approach," *Data Knowl. Eng.*, vol. 117, pp. 373–392, Sep. 2018, doi: 10.1016/j.datak.2018.04.007.

[11] M. Prodel, V. Augusto, X. Xie, B. Jouaneton, and L. Lamarsalle, "Discovery of patient pathways from a national hospital database using process mining and integer linear programming," in *Proc. IEEE Int. Conf. Autom. Sci. Eng. (CASE)*, Gothenburg, Sweden, Aug. 2015, pp. 1409–1414.

[12] V. S. J. Zelst, V. B. F. Dongen, and W. M. P. Aalst, "ILP-based process discovery using hybrid regions," presented at the Int. Workshop Algorithms Theories Anal. Event Data (ATAED), Brussels, Belgium, Jun. 2015. [Online]. Available: https://research.tue.nl/nl/publications/ilpbased-process-discovery-using-hybrid-regions(dab52c29-3467-463c-a62e-74c73c092221).html

[13] M. Prodel, "Process discovery, analysis and simulation of clinical pathways using health-care data," Ph.D. dissertation, École des Mines de Saint-Étienne, Université de Lyon, Lyon, France, 2017.

[14] M. Prodel, V. Augusto, B. Jouaneton, L. Lamarsalle, and X. Xie, "Optimal process mining for large and complex event logs," *IEEE Trans. Autom. Sci. Eng.*, vol. 15, no. 3, pp. 1309–1325, Jul. 2018, doi: 10.1109/TASE.2017.2784436.

[15] S. J. van Zelst, B. F. van Dongen, W. M. P. van der Aalst, and H. M. W. Verbeek, "Discovering workflow nets using integer linear programming," *Computing*, vol. 100, no. 5, pp. 529–556, May 2018, doi: 10.1007/s00607-017-0582-5.

[16] R. Conforti, M. L. Rosa, and A. H. T. Hofstede, "Filtering out infrequent behavior from business process event logs," *IEEE Trans. Knowl. Data Eng.*, vol. 29, no. 2, pp. 300–314, Sep. 2017, doi: 10.1109/TKDE.2016.2614680.

[17] J. M. E. M. van der Werf, B. F. van Dongen, C. A. J. Hurkens, and A. Serebrenik, "Process discovery using integer linear programming," in *Applications and Theory of Petri Nets*, K. M. van Hee and R. Valk, Eds. Berlin, Germany: Springer, 2008, pp. 368–387.

[18] B. N. Yahya, M. Song, H. Bae, S.-O. Sul, and J.-Z. Wu, "Domain-driven actionable process model discovery," *Comput. Ind. Eng.*, vol. 99, no. 1, pp. 382–400, Sep. 2016, doi: 10.1016/j.cie.2016.05.010.

[19] M. Authman, H. Q. Mohammad, and N. H. Shuker, "Vertex and region colorings of planar idempotent divisor graphs of commutative rings," *Iraqi J. Comput. Sci. Math.*, pp. 71–82, Jan. 2022. [Online]. Available: https://journal.esj.edu.iq/index.php/IJCM/article/view/72

[20] S. Warshall, "A theorem on Boolean matrices," *J. ACM*, vol. 9, no. 1, pp. 11–12, Jan. 1962, doi: 10.1145/321105.321107.

[21] J. Han, M. Kamber, and J. Pei, *Data Mining: Concepts and Techniques*. Waltham, MA, USA: Morgan Kaufmann, 2011.

[22] A. Augusto, R. Conforti, M. Dumas, M. L. Rosa, F. M. Maggi, A. Marrella, M. Mecella, and A. Soo, "Automated discovery of process models from event logs: Review and benchmark," *IEEE Trans. Knowl. Data Eng.*, vol. 31, no. 4, pp. 686–705, Apr. 2019, doi: 10.1109/TKDE.2018.2841877.

[23] F. F. Mannhardt, "Sepsis cases—Event log," distributed by 4TU.ResearchData, Dataset, Dec. 2016, doi: 10.4121/uuid:915d2bfb-7e84-49ad-a286-dc35f063a460.

[24] M. de Leoni and F. Mannhardt, "Road traffic fine management process," distributed by 4TU.ResearchData, Dataset, Feb. 2015, doi: 10.4121/uuid:270fd440-1057-4fb9-89a9-b699b47990f5.

[25] B. van Dongen, "Real-life event logs—Hospital log," distributed by 4TU.ResearchData, Dataset, 2011, doi: 10.4121/uuid:d9769f3d-0ab0-4fb8-803b-0d1120ffcf54.

[26] R. P. J. C. Bose and W. Aalst, "Analysis of patient treatment procedures: The BPI challenge case study," BPM Center, Eindhoven Univ. Technol., Eindhoven, The Netherlands, BPM Rep., 2011, vol. 1118. [Online]. Available: https://research.tue.nl/en/publications/analysis-of-patient-treatment-procedures-the-bpi-challenge-case-s, doi: 10.4121/uuid:d9769f3d-0ab0-4fb8-803b-0d1120ffcf54.

[27] F. Mannhard, "Hospital billing—Event log," distributed by 4TU.ResearchData, Dataset, Aug. 2017, doi: 10.4121/uuid:76c46b83-c930-4798-a1c9-4be94dfeb741.

[28] D. Levy, "Production analysis with process mining technology," distributed by 4TU.ResearchData, Dataset, Jan. 2014. [Online]. Available: https://data.4tu.nl/articles/dataset/Production_Analysis_with_Process_Mining_Technology/12697997/1, doi: 10.4121/uuid:68726926-5ac5-4fab-b873-ee76ea412399.

[29] B. Joos, "Receipt phase of an environmental permit application process ('WABO'), CoSeLoG project," distributed by 4TU.ResearchData, Dataset, Aug. 2014. [Online]. Available: https://data.4tu.nl/articles/dataset/Receipt_phase_of_an_environmental_permit_application_process_WABO_CoSeLoG_project/12709127/1, doi: 10.4121/uuid:a07386a5-7be3-4367-9535-70bc9e77dbe6.

[30] T. Sztyler and J. Carmona, "Activities of daily living of several individuals," distributed by 4TU.ResearchData, Dataset, Nov. 2015, doi: 10.4121/uuid:01eaba9f-d3ed-4e04-9945-b8b302764176.

[31] J. Muñoz-Gama and J. Carmona, "A fresh look at precision in process conformance," in *Proc. 8th Int. Conf. Bus. Process Manage. (BPM)*. Berlin, Germany: Springer-Verlag, 2010, pp. 211–226.

[32] J. Cardoso, "How to measure the control-flow complexity of web processes and workflows," in *Workflow Handbook 2005*. Lighthouse Point, FL, USA: Future Strategies, 2005, pp. 199–212.

MARYAM TAVAKOLI-ZANIANI is currently pursuing the Ph.D. degree with the School of Industrial Engineering, Iran University of Science and Technology, Tehran, Iran. Her main research interests include process mining, data mining, optimization, operations research, and medical informatics.

MOHAMMAD REZA GHOLAMIAN received the Ph.D. degree in industrial engineering from the Amirkabir University of Technology, Tehran, Iran, in 1996. He is currently an Associate Professor and a Faculty Member of industrial engineering at the Iran University of Science and Technology. His research interests include optimization, operations research, inventory models in supply chain, supply chain networks design, and multiple criteria decisions making.