Research School for Operations Management and Logistics

Deviations discovery using aligning event logs to business processes using A*

H. Yan, P.M.E. van Gorp, X. Lu, C.C. Chiau, U. Kaymak, H. Duan

Beta Working Paper series 520

BETA publicatie	WP 520 (working paper)
ISBN	
ISSN	
NUR	
Eindhoven	November 2016

Deviations discovery using aligning event logs to business processes using A*

Hui Yan*†, Pieter Van Gorp†, Xudong Lu*, Choo Chiap Chiau‡, Uzay Kaymak† and Huilong Duan*
*School of Biomedical Engineering and Instrumental Science, Zhejiang University, Hangzhou, P.R. China
{h yan,lvxd,duanhl}@zju.edu.cn
† School of Industrial Engineering, Eindhoven University of Technology, Eindhoven, The Netherlands
{p.m.e.v.gorp,u.kaymak}@tue.nl
‡ Philips Research, Shanghai, P.R. China
choo.chiap.chiau@philips.com

***Abstract*—In this study, we focus on *deviations discovery* for a whole business process in general. Meanwhile, we provides a detailed analysis and propose different theoretical foundation towards finding optimal alignments using A* search strategy and validate it in practice. In this study, we formalize the problem of finding optimal alignments between an action-space and an event trace using A* algorithm. We also propose several shortcuts for it.**

***Index Terms*—Best first, Replaying, Alignment, Optimization, Heuristic**

I. Introduction

Business processes play an important role in improving efficiency of individual organizations because business processes provide an effective way to document, understand and further analyze processes in practice. In reality, organizations often allow flexible behaviors and deviations from documented business models because of rapidly changing business environment. Information Technology (IT) systems embedding business processes leave "footprints", recording what happened when [1]. Event traces lay "happened" activities, namely events chronologically. What happened in reality is not always the same with what the business processes described. The unexpected events are called "deviations". Analyzing such deviations gives insights into redesign of processes or preventing deviations from happening again.

Prior researches have focused on defining specific compliance rules such as using Linear Temporal Logic (LTL) [2] to monitor deviations correctly and efficiently on runtime. However, it is not cost-effective in sense that in order to know the overall deviations, users need to define compliance rules exhaustively. Moreover, some unnoticeable deviations may be the cause of severe bottlenecks. Studies in [3], [4] align observed behavior and business processes in form of Petri Nets [5] for the purpose of conformance check. Optimal alignments reveal deviations as added or missing activities. However in industry domain, more efficient, user-friendly and intuitive modeling language have emerged and been widely used, such as Business Process Modeling and Notation (BPMN) [6]. Transforming BPMN into Petri Nets has been studied yet some advanced semantics in BPMN can not be transformed in Petri Nets, such as *General Synchronizing Merge* [5]. Optimal alignment based on a more general formalism is necessary.

In literature, there exist many process modeling formalisms, Petri Nets, YAWL [7], BPMN and so on. Though they have different execution semantics, the common feature is control-flow dimension, which is organizing activities and enabling synchronization and/or alternatives among activities. Study in [8] has used simple split and join construct to express control-flow dimension of a process model. Study in [9] adopts a flexible model to represent sequential, synchronization and alternatives relations of activities in a process.

In this study, with the assumption that control-flow aspect of process models is the same, we propose using a simple and general formalism to represent control-flow aspect: action-space. Further more, we bring forward a method for *deviations discovery* using action-space in theory and validate it in practice.

Control-flow aspect could be obtained using several approaches, such as process mining, converting existing process models, or extracting from the statespace. Statespace recorded all the execution information and sequences of activities executed, which is also referred to as reachability graphs [10] or marking graphs [11]. There are studies generating statespace for Petri Nets [11] and BPMN 2.0 process model [12]. Statespace can be regarded as a basic transition system including states during a process model's running and switching among them. From statespace, a pure singleton of activities (removing non-activity constructs) forms an action-space.

The research question is given an action-space representing a business process, how to discover deviations efficiently as well as simulating deviations. If we can replay an event trace in the business process simulating deviations as well, deviations becomes visible. Since there are various ways of replaying an event trace (adding or missing activities at different positions), a golden standard for the best is needed. Regardless of special cases, we assume that the less the number of deviations is, the closer it resembles the reality. Among all the ways of replaying, also called alignments, the optimal alignments are those with least number of deviations.

Finding optimal alignments is the main challenge. To achieve this goal, we searched in literature and inspired

by priori studies [13], [14]: A* algorithm, which adapted from one of the Best First Search strategies. However, we found some limitations in priori studies, for example, heuristic estimation of A* in [13] is stationary for all process models regardless of complex patterns and their approach can not deal with loops efficiently. In this study, we provide a detailed analysis on methods to overcome its limitations and formalize it in theory. To validate this method, we use a business process in clinical domain as an example.

The rest of the paper is organized as follows. Section II reviews the preliminaries needed in this study. Section III describes the methods. Section IV shows the result. Related work is shown in Section V. Section VI concludes this paper.

II. Preliminary

A. Event Traces

Information Technology (IT) systems supporting business processes record what happened when it happened by leaving their "footprints" [1]. An event trace is an ordered list of events according to the timestamps. Each event has a role, a timestamp and resource or other information associated with it. From a formal point of view, A_L denotes the set of activities that may be recorded in the log. A_L^* denotes the set of all possible sequences consisting of elements of A_L. $\sigma \in A_L^*$ denotes a trace. For example, $\sigma_0 = [acdeh]$ is an event trace, "a" represents an event such as "Start loading application by John Doe at 9:00 am, 1st July, 2008".

B. Control-flow representation of process models

A process model includes basic constructs such as transitions, roles, tasks, splits and joins and networks (subprocess) [8]. Constructs such as splits and joins affect control-flow behavior of a process. Splits allow defining the possible control paths through the process. Joins express the type of synchronization at a specific point in the process. Though different process modeling languages have different semantics, splits and joins can be represented in a general format as in [8]. We use Petri Nets and BPMN as examples to illustrate that.

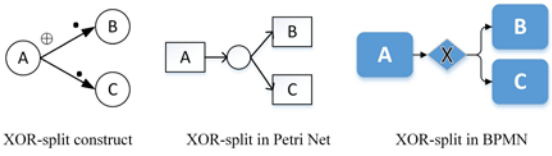


Fig. 1: XOR-split

XOR-split construct: A point in the process where, based on a decision or process control data, one of several alternatives is selected [8]. The leftmost graph in Fig.1 is the representation of XOR-split according to [8] using "⊕". Middle graph is the representation in Petri Net. The rightmost graph is the representation in BPMN.

AND-split construct: During the execution of a process, when an AND-split is reached the single thread of control splits into multiple treads of control which are executed in parallel, thus allowing activities to be executed at the same time or in any order [8]. It is assumed that all the alternatives are selected and executed. Fig.2 shows AND-split according to [8], in Petri Net and BPMN.

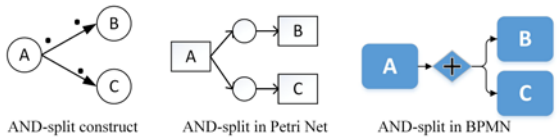


Fig. 2: AND-split

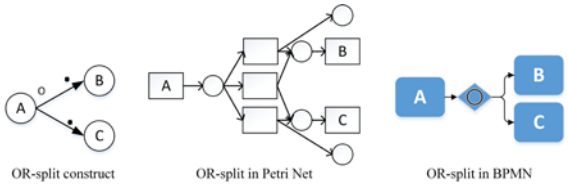


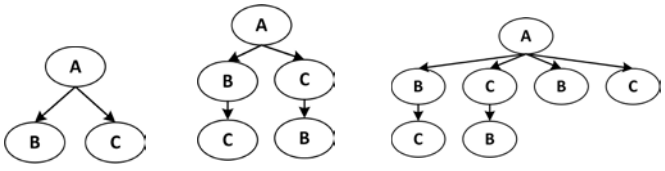
Fig. 3: OR-split

OR-split construct: A point in the process where, based on a decision or process control data, one or more alternatives are selected [8]. Fig.3 shows OR-split according to [8], in Petri Net and BPMN.

AND-join constructs start execution when all their incoming transitions are enabled. OR-join constructs start execution when a subset of their incoming transitions are enabled. XOR-join constructs are executed as soon as one of their incoming transitions is enabled. For the sake of space, we don't list all of them here. The idea is that control-flow perspective of process models can be represented using simple splits and joins.

C. Action-space of process models

Deviation discovery is basically comparing happened events and a process model which prescribes them and the order in between. Since event traces lay out events based on their order in time, it is necessary to have orders between activities in the process model as well. A graph with all the activities and their orders is called an *action space*. With simple representing process models in split and join constructs, it is possible to generate action spaces for all kinds of process models for control-flow perspective.



(a) Action-space of XOR-split (b) Action-space of AND-split (c) Action-space of OR-split

Fig. 4: Action-space of three splits

Fig.4 shows action-spaces of XOR-split, AND-split and OR-split constructs. Each action-space consists of nodes and edges in between. A node in action-space represents an activity in process model. Edges indicate sequential logics. What is worth mentioning is that outgoing edges are exclusive, meaning only one outgoing edge is valid at one time for each branch point. Take AND-split construct for example, the action-space of it (Fig.4b) expresses *parallel* by showing all the possibilities of execution orders between two outgoing activities.

Definition I An action-space of a process model M is a directed graph $G = (V, E)$ where:

- V is a finite set of actions, each action $v \in V$ is a tuple (A_M, T_{type}). A_M is a set of activities of M, $T_{type} = \{Start, Complete\}$ indicates the action type of an activity.
- $E \subseteq V \times V$ is a finite set of ordered pairs of V, called edges, directed edges, or arrows.

If there is a sequence of edges $< (v_0, v_1), (v_2, v_3), \ldots, (v_x, v_0) >$, it indicates a loop exist $v_0 \rightarrow v_0$.

Though this paper is not describing how to generate action-spaces for various process models, several approaches are available, such as process mining, converting existing process models, or extracting from the statespace. Statespace recorded all the execution information and sequences of activities executed, which is also referred to as reachability graphs [10] or marking graphs [11]. More specifically, statespace consists of states during a process model's running and switching among them. Switching between states are indeed actions of action-space, thus an action-space of a process model can be extracted from statespace of it. There are studies generating statespace for Petri Nets [11] and BPMN 2.0 process model [12].

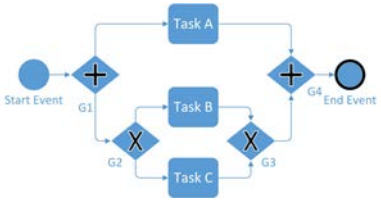


Fig. 5: BPMN 2.0 Process Model

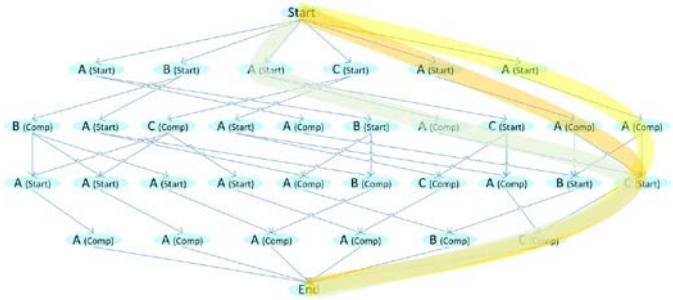


Fig. 6: Action-space of Process Model in Fig.5

Fig.5 shows a BPMN 2.0 process model describing exclusive relation (Exclusive Gateway G2) between Task B and Task C, and its parrel relation (Parallel Gateway G1) with Task A. Fig.6 is the corresponding action-space which is obtained using the tool in [12]. Three highlighted pathes in Fig.6 show a same sequence: Task A starts, Task A completes, Task C starts and Task C completes. The three pathes differ at the orders between Task A and G2 (since they are parallel, either one is preceding is possible).

Here we focus on the situation where an action-space as a directed graph has only one root node with zero incoming edges, even though some process language support two instances starting in one process.

D. Deviation Discovery using Alignment

In order to find deviations between an event trace and action-space of a process model, we use *optimal alignment technique* [13]. An alignment can be seen as a way of aligning them. In an alignment, for the events which can be tracked back to the activities in the action-space, we label as "matched" pairs; for those events which cannot be tracked back to the action-space and those activities in the action-space which are missing in the event trace, we label them as *deviations*. An alignment lays matches and deviations according to sequential logics respectively.

Supposing $\sigma_1 =< a, c, b, c, d >$ is a trace of events from A_L (A_L is a set of events) and $\sigma_2 =< a, b, c, d >$ is a sequence of activities from A_M (A_M is a set of activities in a process model). Three alignments between σ_1 and σ_2 are:

$\gamma_1 =$

a	c	b	c	d
a	⊥	b	c	d

,

$\gamma_2 =$

a	⊥	c	b	c	d
a	b	c	⊥	⊥	d

and

$\gamma_3 =$

a	c	b	c	d
a	b	⊥	c	d

.

The upper rows in these three are traces of events in σ_1 with or without "⊥" in between, and the lower rows are sequences of activities in σ_2 with or without "⊥" in between. Each collum is a pair (x, y) where $x \in A_L \cup \{\perp\}$ and $y \in A_M \cup \{\perp\}$ and it is called one match in an alignment. There are four different types:

1) Match in both with the same name, $x \in A_L$ and $y \in A_M$ and $x = y$, indicating a matched pair, e.g. (a, a).
2) Match in both with different names, $x \in A_L$ and $y \in A_M$ and $x \neq y$, e.g. (c, b) in γ_3.
3) Match in L, $x \in A_L$ and $y = \perp$ indicating a deviation, e.g. $(c, \perp)$ in γ_1.
4) Match in M, $x = \perp$ and $y \in A_M$ indicating a deviation, e.g. $(\perp, b)$ in γ_2.

Match in both with different names makes sense only within a given context, for example, two different activities are actually the same. But it needs specific illustration, so we do not include it in this paper. Regardless of special cases, we assume that the less the number of deviations is, the closer it resembles the reality. So the optimal alignments are those with least number of deviations. Comparing these three alignments, γ_1 is the optimal alignment.

III. Optimal Alignment Searching using A* algorithm

With an action-space and an event trace, the next step is to find an optimal alignment between them two. As we explained in Section II-D, an alignment is a way of "matching" the trace and a selected path in the action-space. Such "matching" allows adding fake activities or events (deviations, "$\perp$" in Section II-D). An optimal alignment should have the least number of deviations. Simply matching events in a trace and activities in the action-space according to their names is neither enough nor correct, for the reasons: (1) an activity in the process model could happen more than once; (2) sequential logics of activities in a action-space also needs being guaranteed.

A. Naive solution

A straightforward way is to align actions (in the action space) and events (in the trace) successively according to their sequential orders respectively. Whenever there is a mismatch, generate two situations: adding a fake event and a fake action. Fig.7 shows steps of aligning a trace of events ($a \rightarrow c \rightarrow b \rightarrow d$) and an action-space on the right. For the sake of simplicity, action type T_{type} and event type are omitted.

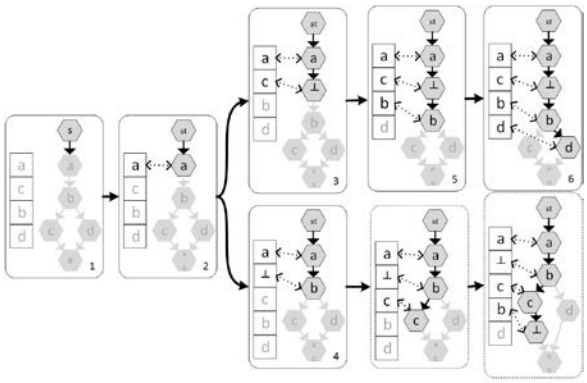


Fig. 7: Naive way of alignment

The first action a and event a make up a match ((a,a) shown in Node 2). Successors are "c" and "b" respectively. As "c" and "b" are not matched, we add a "$\perp$" as a fake action to match "c" ($(c,\perp)$ in Node 3); and a fake event "$\perp$" to match "b" ($(\perp,b)$ in Node 4). Following Node 3, match (b,b) continues (Node 5). Node 6 is similar to Node 5. Such an exploration ends when all the elements in the trace and in the action-space are matched.

The exploration results in a tree whose root is an empty match (Node 1) while leaves are pairs involving ends of both. Each path from the root to a leaf node is an alignment between the trace and a path in the action space. By comparing the number of matched pairs on each path, we select those with the least numbers as optimal alignments. However, we can see that the exploration tree could be very large since whenever there is a mismatch, one branch splits into two sub-branches.

B. Path-search Problem and A* algorithm

We are looking for a way of only generating branches that contribute to an optimal alignment. Of all search strategies, one of the most popular methods of exploiting heuristic information to cut down search time is the informed best-first strategy [15]. Our optimal alignment search technique uses informed best-first strategy. In order to understand that, a detailed example of a typical path-search problem is depicted here. A typical problem is given two nodes in a weighted directed graph, to find a path with the lowest weight (optimal path). Fig.8 shows a "coordinate system" with some parts are not connected. Assuming the problem is to find the shortest path from $s(0,0)$ to $e(4,4)$. Circles are nodes. Lines between nodes indicate they are connected and weight is length of the line. Best-first strategy works as the following: starting from node *s*, all 3 neighbors of *s* are evaluated; the one with highest "merit" is selected into the optimal path. In this example, $n2$ is better than $n1$ and $n3$ for making the shortest path. Best-first strategy avoids exploring less meritorious nodes thus improving efficiency.

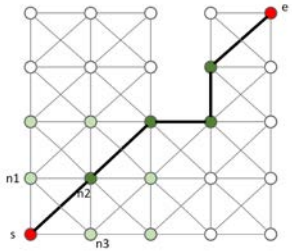


Fig. 8: Path-searching Problem

The key is the *evaluation function* for judging "merit". Evaluation function should always provide optimistic estimate of the final cost of the candidate node [15]. By far, the most studied version of best-first strategy is A* algorithm [16]. A* algorithm provides optimistic estimate by introducing an evaluation function ($f(n)$) taking into consideration both "current merit" and "future merit" (heuristic information). Let current merit $g(n)$ be *Euclidean Distance* from node n to start node s, so $g(n1) = Distance(s, n_1) = 1$, $g(n2) = \sqrt{2}$ and $g(n3) = 1$. Let future merit $h(n)$ be distance from n to end node e, thus $h(n1) = Distance(n1, e) = 5$, $h(n2) = 3\sqrt{2}$ and $h(n3) = 5$. A* adopts $f(n) = g(n) + h(n)$, so $f(n1) = 6, f(n2) = 4\sqrt{2}$ and $f(n3) = 6$. $f(n2)$ is smaller than $f(n1)$ and $f(n3)$, so at this step, $n2$ is a better choice than $n1$ and $n3$.

We investigate the difference between optimal alignment searching and path searching. There are two main challenges: 1. the "weighted directed graph" does not exist in optimal alignment searching; 2. evaluation function of optimal alignment searching problems should be created. The following sub-sections will elaborate on them.

C. Constructing the searching graph as weighted graph

Though there is no existing weighted directed graph in optimal alignment searching problem, we know clearly the

starting "node" and characteristics of the targeting "node". The starting "node" is before matching (Node 1 in Fig.7). The targeting "node" should have both ends of the action space and the trace matched (to the real or fake). Along the way from the starting "node" to a targeting "node", we create "nodes" between. The way of creating is just as matches are added one by one in Fig.7. Thus a directed graph is finished after a targeting "node" is found.

In this new directed graph, each "node" is its father node adding a new match. There are two kinds of matches: *true match* and *fake match*. For example, Node 2 in Fig.7 adds (a,a), which is a true match. Node 3 adds $(c, \perp)$, which is a fake match. Here, we use the term *Match* to represent a node as well as refer to matches in Fig.7. A Match (**Definition II**) should contain the basic information about which event in a trace $Event$ or action in action space $Action$ is matched.

Definition II A Match has 6 elements ($Event$, $Action$, $fScore$, $nDev$, $Type$, $PreMatchSet$), where:

- $Event$ is an event in a trace σ.
- $Action$ is an action in action space G_{ac}.
- $fScore$ is value of evaluate function.
- $nDev$ is the number of deviations in current match.
- $Type$ is the type of $Match$, which is $MatchInBoth$, $MatchInAction$, $MatchInEvent$ (Section II-D).
- $PreMatchSet$ contains a set of parent nodes of $Match$ in the constructed weighted directed graph.

$fScore$, $nDev$ and $PreMatchSet$ are important variables for further use in the algorithm. To give an examples, Node 2 is represented Match $2 = (a, a, 0, 0, MatchInBoth,$Node $1)$. In the illustration in Fig.7, each Node has one parent Node. In order to decrease the computation time, we come up with the idea of reusing Matches under the condition that two Matches share the same Event, Activity and fScore. So each Match may have more than one parent Match which are stored in $PreMatchSet$.

D. Algorithm explanation

Section III-C explains Matches and creating successive Matches after a given one. This section describes creating Matches selectively using idea of A* to find optimal alignment efficiently. Evaluation function for selecting is not detailed here for conciseness of algorithm itself.

The whole algorithm is shown in Algorithm 1. Inputs are a trace and an action-space. It starts from an empty Match (Line 2). OPEN stores Matches which are to be evaluated and explored. The Match n with the lowest $f(n)$ is selected to explore firstly in OPEN (Line 4). Once a Match is explored, it is removed from OPEN (Line 8). If n is a targeting one, the program ends and returns n (Line 5 to 7). If n is not targeting, *GenerateFollowMatches* computes n' - successive Matches of n (Line 9). Their $f(n')$ are calculated (Line 10). If there is a Match n'' equals n' (Line 11), it is not necessary to keep n' but add n as parent Match of n'' (Line 12). Otherwise, add n' into G (Line 14), $OPEN$ (Line 15) and create connection between n and n' (Line 16).

Algorithm 1 Optimal alignment searching with A* algorithm

```
1:  procedure REPLAYTECHNIQUE(σ, G_ac)   l> σ: a trace in log; G_ac: an action space;
2:      Add a new Match S_Empty into OPEN and G;  l> G: the created graph;
3:      while OPEN is not empty do
4:          Get Match n in OPEN that has the lowest f(n);
5:          if n is targeting then
6:              return n;                                  l> End while.
7:          end if
8:          Remove n from OPEN;
9:          for each n' = GenerateFollowMatches(n) do
10:             Calculate f(n').
11:             if there is a n'' in G fulfilling n'' == n' then
12:                 Add n into n''.PreMatchSet;
13:             else
14:                 Add n' into G;
15:                 Add n' into OPEN;
16:                 Add n into n'.PreMatchSet;
17:             end if
18:         end for
19:     end while
20: end procedure
```

Evaluation function decides priority of Matches in OPEN being explored. Each time, a Match with the lowest evaluated value in OPEN is explored and removed from OPEN. The algorithm ends when there are no Matches left in OPEN or one Match meets the requirement both with lowest $f(n)$ and a targeting condition.

E. Evaluation function formalization

Similar to what A* algorithm proposed, in optimal alignment searching problems, evaluation function $f(s)$ should also include "current merit" ($g(s)$) and "future merit" ($h(s)$). Here we simply formulate it as $f(s) = g(s) + a * h(s)$ to make explicit that the weight of $g(s)$ and $h(s)$ is not always the same. Since the final goal is to find a path with the least number of deviations, the golden standard is the number of deviations. At a single Match s, we can count the number of deviations as current "merit". So $g(s)$ is the number of mismatches in s. For future merit, because we can't predict the number of deviations in the final completed alignment of s accurately, we use an approximate estimate of the effort becoming a targeting alignment, in other words, "potential" of the candidate becoming a targeting alignment.

$h(s)$ could be an optimistic estimate of the "potential" for s to become a targeting alignment. [14] adopts remaining events in a trace as estimate for a targeting alignment heuristically. In this paper, we use the number of remaining steps to a targeting alignment indicating the effort. In other words, the number of remaining steps needs information not only from trace's aspect but also the model's aspect. A targeting alignment requires reaching ends of both the action space and the trace. So these two aspects should be combined. We define

$h(s) = h_L(s) + h_M(s)$, where $h_L(s)$ represents the trace's part, $h_M(s)$ represents the model's part, and they are equally important.

For $h_L(s)$, we use the number of events from $s.Event$ to the end of the trace. For $h_M(s)$, since the action space is a directed graph with loops **Definition I**, it is tricky to calculate the number of remaining actions along one or more paths. Since an action-space as a directed graph has only one root node (zero incoming edges), it make sense to compute depth information for the whole graph. Distance from depth of $s.Action$ to that of the end one in action-space indicates remaining "effort" from model's perspective. The depth of root node is 1 and it increases along one direction. In case of more than one incoming edges (may involving loops), the depth of action is determined by the smallest depth. $h_M(s)$ is the difference between the maximum depth and depth of $s.Action$.

F. GenerateFollowMatches *and weight value* a *of heuristic function*

In Algorithm 1, there is a method *GenerateFollowMatches* in Line 9. We use an example in Fig.9 to illustrate it. Fig.9 shows a trace is replayed to part of an action-space which has a loop from action node "a" to "b". In the graph, we use id to represent Entry and Action Node. Match s_0 has Entry 4 and Action Node 4 matched with type $MatchInBoth$ (s_0:(4,4,$MatchInBoth$,$f(s_0)$)). After s_0, next Entry is b; next action nodes are f and b. Three possible ways of aligning them are s_1, s_2 and s_3. s_2 is of type $MatchInEvent$, namely "$(b, \perp)$". The dark arrow in s_2 that points to action node in Layer 4 indicates the accomplished position in action space.

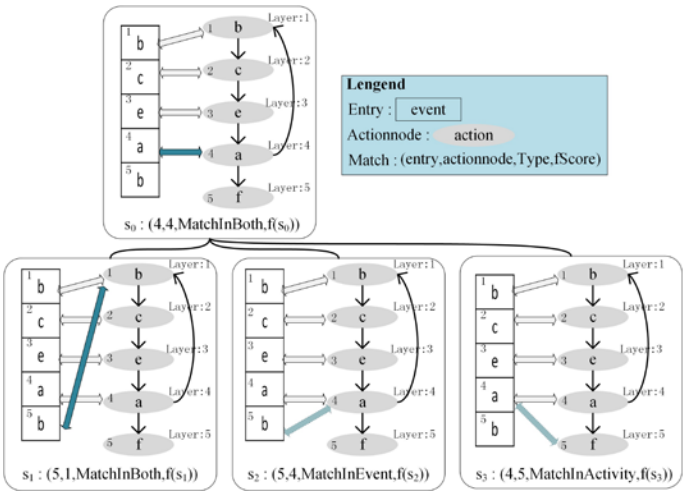


Fig. 9: a computation illustration

Among s_1, s_2 and s_3, s_1 should be the best one intuitively as it is of type $MatchInBoth$. $f(s_0)$, $f(s_1)$, $f(s_2)$ and $f(s_3)$ are calculated in TABLE I. $g(s)$ is defined as number of deviations, so $g(s_1)$ is the same as $g(s_0)$; $g(s_2)$ and $g(s_3)$ both add "1" to $g(s_0)$ since they introduce one deviation in current unfinished alignment.

$h_L(s)$ is defined as the number of events from $s.Event$ to the end of the trace. $h_L(s_1)$ and $h_L(s_2)$ are $h_L(s_0)$ subtracting "1", since both of them matches one event in a trace. For s_3, the position of matched Entry does not change, so it stays the same with $h_L(s_0)$. h_M relates to layer of current action node. For s_1, $h_M(s_1) = h_M(s_0) + 3$ as s_1' action node is in Layer 1 and s_0' action node is in Layer 4. $h_M(s_3) = h_M(s_0) - 1$ as s_1' action node is in Layer 5 and s_0' action node is in Layer 4. To make it general, we simply use x to indicate any positive integer. "+" in collum "h_M" shows the case where position of action node goes to a higher layer. "-" shows the case where position of action node goes to a lower layer. Row 6 to 9 calculate amount of change. A, B, C, D, E represent 5 different situations.

To assure that s_1 is more meritorious than s_2 and s_3, namely $f(s_1)$ is smaller than $f(s_2)$ and $f(s_3)$. It requires A and B are smaller than others: $A < C, A < D, A < E, B < C, B < D, B < E$. The value of a should fulfill: $\frac{1}{2x-1} > a > -\frac{1}{2x+1}$. Taking maximum of x guarantees $f(s)$ always provides a correct evaluation of each candidate alignment. x should be assigned the maximum depth difference between two actions in an action space. In this model, x is 3 and $0.2 > a > -0.14$. So the value of a differs with each model.

Introducing weight factor a is: single increment of h_L, h_M and g is "1", however g plays a dominant role. g represents number of deviations which is a golden standard for selecting an optimal alignment. Pre-calculating of a using TABLE I guarantees g is always dominant even in case of loops.

IV. Results

To evaluate our approach, we modeled diagnosis validation process for unstable angina in Catharina hospital in Fig.10. This process model is based on the results in [17]. There are 18 tasks when the sub-processes are expanded. We compute the action-space based on the statespace using the tool in [12]. The action-space is shown in Fig.11. It contains 36 Actions. We show each Task as one action instead of separating its Start or Complete type. Actions in this graph is displayed according to its "layer" from top to bottom.

Table II shows our experimental setting and the results. In total we constructed 11 log files each of which consists of 100 traces. Traces in Log 1 is of length 34 and fully match the longest path in action-space in Fig.11. From Log 2 to Log 11, we gradually inserted deviations of different ratio, from 5 to 50 percent (inserted events are also from the process itself). The positions of artificial deviations are random.

The fifth collum in Table II shows number of deviations detected using our method. We can see from Log 1 to Log 9, our method found the same number of deviations as we artificially inserted. However, when the noise ratio increases to 45% (Log 10), the number of deviations found is more than that we inserted. For Log 10, we found that 61 out of 100 traces are detected with 30 deviations; the other 39 traces have 28 deviations as they are supposed to. Log 11 has 6 traces with 37 deviations which are more than 35 as they are supposed to. The reason is that heuristic algorithms such as A* algorithm will often fall into local optimums. To guarantee the

TABLE I: Different situations to calculate a

	g	h_L	h_M	$f = g + a * (h_L + h_M)$
$f(s_0)$	$g(s_0)$	$h_L(s_0)$	$h_M(s_0)$	$f(s_0) = g(s_0) + a * (h_L(s_0) + h_M(s_0))$
$f(s_1)$	$g(s_0)$	$h_L(s_0) - 1$	$h_M(s_0) + x$ $h_M(s_0) - x$	$f(s_1)=g(s_0) + a * (h_L(s_0) - 1 + h_M(s_0) + x)$ $f(s_1)=g(s_0) + a * (h_L(s_0) - 1 + h_M(s_0) - x)$
$f(s_2)$	$g(s_0) + 1$	$h_L(s_0) - 1$	$h_M(s_0)$	$f(s_2)=g(s_0) + 1 + a * (h_L(s_0) - 1 + h_M(s_0))$
$f(s_3)$	$g(s_0) + 1$	$h_L(s_0)$	$h_M(s_0) + x$ $h_M(s_0) - x$	$f(s_3)=g(s_0) + 1 + a * (h_L(s_0) + h_M(s_0) + x)$ $f(s_3)=g(s_0) + 1 + a * (h_L(s_0) + h_M(s_0) - x)$
	Δg	Δh_L	Δh_M	Δf
$\Delta f(s_1)$	0	-1	x $-x$	A:$a * (x - 1)$ B:$-a * (x + 1)$
$\Delta f(s_2)$	1	-1	0	C:$1 - a$
$\Delta f(s_3)$	1	0	x $-x$	D:$a * x + 1$ E:$a * (-x) + 1$

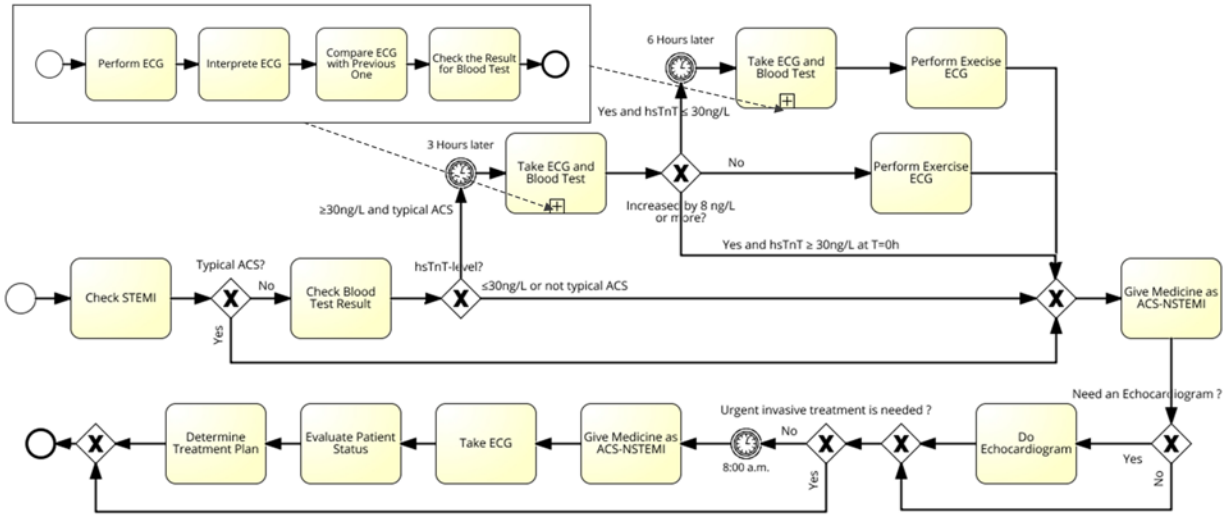


Fig. 10: Unstable Angina Diagnosis Validation Process in BPMN

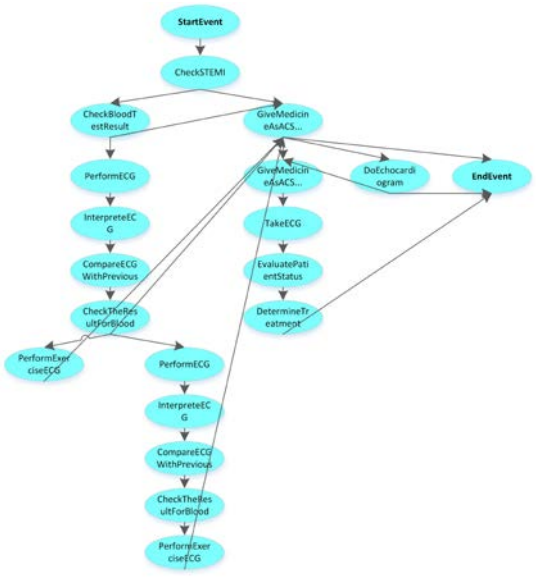


Fig. 11: Action-space of Process Model in Fig.10

correctness, users need to make sure the noise ratio is under 45% by filtering unrelated events with the process.

The sixth collum in TABLE II shows numbers of Matches constructed using A* algorithm for a trace with different deviation ratio. Deviations are inserted randomly. In order to show the improvement of efficiency, we theoretically compute the numbers of nodes using the "Naive Way" in Fig.12. Supposing deviations are inserted equidistantly in the traces from 5% to 50%. A trace of length l with k deviations results in a full binary tree to get the optimal alignment. The total number of nodes is $(2^{k+1} - 1) * \frac{l}{k+1}$. The number increase exponentially as the ratio of deviations increases as shown in Fig.12. Our method using A* algorithm generates less nodes to find the optimal alignment.

To evaluate its performance, we did the same experiment 20 times for each log. The seventh collum shows the average time it takes. The hardware is a computer with Intel i5-4590 and CPU of 3.30 GHz processor and 8.00 GB memory. For instance, Log 8 with 100 traces of 40% deviations takes 72 seconds, which is acceptable to users in all kinds of domains.

TABLE II: Deviation Discovery Evaluation Result

Log No. (100 traces)	Length of Trace	Noise Rate (%)	Artificial Deviations Num.	Found Deviations Num.	Matches Num.	Time(ms)
1	34	0	0	0	40	32.75
2	36	5	200	200	51	54.4
3	38	10	400	400	100	136.55
4	41	15	700	700	352	762.05
5	43	20	900	900	516	1367.8
6	46	25	1200	1200	1255	5743.65
7	49	30	1500	1500	1798	12908.75
8	53	35	1900	1900	2565	28870.95
9	57	40	2300	2300	4038	72453.1
10	62	45	2800	2922	6519	159391.7
11	69	50	3500	3512	11204	323838.05

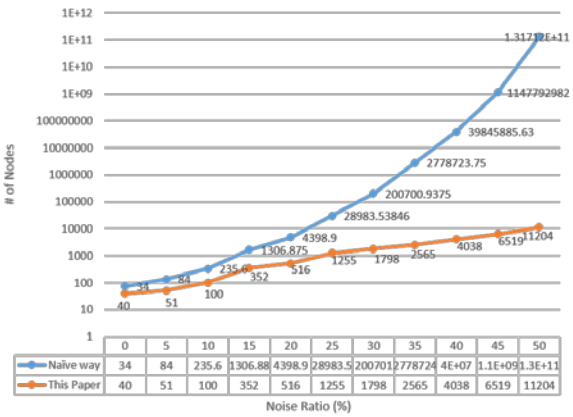


Fig. 12: The number of nodes to construct optimal alignments for the BPMN process in Fig.10 with corresponding different percent of noise according to the horizontal axis.

V. Related work

Researches in [3], [14] mentioned to use A* algorithm to limit the search by introducing a heuristic evaluation function to assess which branch has the most potential being the optimal alignment. When we applied their approaches, we find efficiency is not always good and sometimes there are errors especially involving loops. Approaches in [15] can not reuse nodes in exploration because each node is unique. Our approach changed this situation.

VI. Conclusion

Business processes play an important role in documenting and analyzing processes. Organizations often allow flexible behaviors and deviations from business models. This paper proposes a method discovering deviations on control-flow perspective for process models and event logs. Control-flow perspective exist in all process models and can be represented using action-space.

With action-spaces and event logs, we adapt A* algorithm in optimal alignment searching to efficiently discover deviations. We give a detailed analysis of setting the evaluation function as well as the algorithm itself. It enable deviations discovery for complex structures such as loops without compromising efficiency. We evaluate this method using a BPMN 2.0 process vs. 11 event logs. The performance is kept good. Further research direction could be analyzing distributions of deviations on different paths in action-space. Deviations also give insights into redesign of processes.

Acknowledgment

The research leading to these results has received funding from the Brain Bridge Project sponsored by Phillips Research.

References

[1] W. M. P. van der Aalst, *Process Mining - Discovery, Conformance and Enhancement of Business Processes.* Springer, 2011.

[2] H. T. D. Beer and B. F. V. Dongen, "Process mining and verification of properties: An approach based on temporal logic," in *On the Move to Meaningful Internet Systems 2005: CoopIS, DOA, and ODBASE: OTM Confederated International Conferences, CoopIS, DOA, and ODBASE 2005, volume 3760 of Lecture Notes in Computer Science*. Springer-Verlag, 2005, pp. 130–147.

[3] M. De Leoni, F. M. Maggi, and W. M. van der Aalst, "Aligning event logs and declarative process models for conformance checking," in *Business Process Management*. Springer, 2012, pp. 82–97.

[4] A. Adriansyah, B. F. van Dongen, and W. M. van der Aalst, "Memory-efficient alignment of observed and modeled behavior," *BPMcenter. org, Tech. Rep*, 2013.

[5] N. Lohmann, E. Verbeek, and R. Dijkman, *Transactions on Petri Nets and Other Models of Concurrency II-Petri Net Transformations for Business Processes A Survey*, K. Jensen and W. M. Aalst, Eds. Berlin, Heidelberg: Springer-Verlag, 2009.

[6] B. P. Model, "Notation (bpmn) version 2.0," *OMG Specification, Object Management Group*, 2011.

[7] W. M. Van Der Aalst and A. H. Ter Hofstede, "Yawl: yet another workflow language," *Information systems*, vol. 30, no. 4, pp. 245–275, 2005.

[8] J. Cardoso, "How to measure the control-flow complexity of web processes and workflows," *Workflow handbook*, vol. 2005, pp. 199–212, 2005.

[9] A. Adriansyah, B. F. van Dongen, and W. M. van der Aalst, "Towards robust conformance checking," in *Business Process Management Workshops*. Springer, 2011, pp. 122–133.

[10] G. J. Holzmann, "The model checker spin," *IEEE Transactions on software engineering*, vol. 23, no. 5, pp. 279–295, 1997.

[11] K. Wolf, "Generating petri net state spaces," in *Petri Nets and Other Models of Concurrency–ICATPN 2007*. Springer, 2007, pp. 29–42.

[12] P. Van Gorp and R. Dijkman, "A visual token-based formalization of BPMN 2.0 based on in-place transformations," *Information and Software Technology*, vol. 55, no. 2, pp. 365–394, Feb. 2013.

[13] W. Van der Aalst, A. Adriansyah, and B. van Dongen, "Replaying history on process models for conformance checking and performance analysis," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 2, no. 2, pp. 182–192, Mar. 2012.

[14] A. Adriansyah, B. F. van Dongen, and W. M. van der Aalst, "Conformance checking using cost-based fitness analysis," in *Enterprise Distributed Object Computing Conference (EDOC), 2011 15th IEEE International*. IEEE, 2011, pp. 55–64.

[15] R. Dechter and J. Pearl, "Generalized best-first search strategies and the optimality of a*," *Journal of the ACM (JACM)*, vol. 32, no. 3, pp. 505–536, 1985.

[16] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *Systems Science and Cybernetics, IEEE Transactions on*, vol. 4, no. 2, pp. 100–107, 1968.

[17] L. Vermeulen, "A process modelling method for care pathways," Master's thesis, Eindhoven University of Technology, 2013.