In this paper, we propose an evolutionary computing approach based on Genetic Algorithms for composing an efficient trace given a desirable utility function based on the observations made in the event logs of several peer-organizations. Our proposed approach works with a set of event logs from different peer-organizations and generates an efficient trace according to a utility function. The main advantage of our approach is that we primarily work with event logs that are more accurate representations of the actual execution of a process within an organization. Furthermore, we generate efficient traces that are put together through the identification of sub-parts of the observed traces that are locally optimal. We report on our experiments on the BPIC’15 dataset that show improvement in terms of the optimality of the generated traces.

Additional Key Words and Phrases: business process families, process improvement, optimization algorithms, event logs

1. INTRODUCTION

Many organizations are replacing their traditional processes with so-called Process Aware Information Systems (PAIS) [1, 2] and even more recently moving into the domain of cognitive business process management systems [3]. Enabling continuous process improvement lies at the core of such efforts, in part requiring the identification of best practice processes. Best practice processes refer to the recommended execution model that has already proven successful [4]. Currently, best practice processes (aka best practices) are, not exclusively, but mainly extracted manually and with the involvement of domain experts. However, given the limited cognitive capacity of the domain experts, as the number and complexity of organizational processes increase, the likelihood of identifying the best possible process model for a given process decreases [5]. Moreover, due to the emergence of agile and ad hoc business process models, processes are often updated very frequently and therefore, the identification of best practices would need to be an ongoing activity.

In addition to the manual identification, an additional source for best practices could be the observation of similar comparative processes in other peer-organizations. For example, there are many peer-organizations such as municipalities, hospitals, insurance companies, and car rental companies, among others that execute very similar variants of the same process model. As reported in the CoSeLoG project [5], which involved several Dutch municipalities, many of the processes in these municipalities were driven by legislation, e.g., citizen registration, building permit and others. Such processes are substantially regulated and therefore share many similarities. However, although legislation enforces that the organizations conform to major principles, some degree of freedom is often allowed regarding the concrete implementation of such processes resulting in different customizations of the same process [5]. It is clear that various ways of executing the same process will certainly have effects on the properties of the process, e.g., execution time, cost and complexity. The valuable data stored in peer-organizations’ event logs can be used for automatically discovering best practices based on utility functions such as execution time. The reason that we emphasize on peer-organizations in our work is that we are interested in extracting best experiences from organizations that have similar goals, rules, organizational culture and structure. Our work is therefore geared towards mining and generating desirable execution traces for peer-organizations.

Process improvement approaches can be categorized into three main groups: i) human expert based methods, ii) best practice based approaches, and iii) automatic or semi-automatic approaches that use event logs. In the first group, process analysts and
design experts improve an organization’s process models based on a standard methodology. In [6], a complete evaluation of these approaches is presented. The second group of approaches is also based on the involvement of process analysts and design experts. In addition, there is the need for domain experts as well. These experts investigate different processes in existing systems from within the same domain. Then, according to the requirements of the target organization, a suitable process model is selected which is referred to as a best practice [7].

The third group of approaches is based on event logs. These approaches process existing logs of peer-organizations for identifying desirable reusable best practices.

While existing techniques in the literature for identifying best practices have shown noticeable results, there are some areas that can still be improved. Among these, we specifically focus on two areas where more work is warranted:

- Most of the prominent approaches that identify best practices primarily focus on the formal business process models of peer-organizations. In other words, the formal representation of the business process is taken into consideration, as opposed to the actual executed model that is reflected in the event logs. Given the fact that actual and formal models of a process can be different for various reasons, a more accurate representation of the real executed model would be the one obtained from the event logs. Therefore, in our work, we identify best practices from event logs as opposed to the formal business process models.

- Furthermore, even within the few works that have already taken on the challenge of identifying best practices from event logs, their results might not necessarily be desirable. The main reason for this is that such approaches primarily opt for identifying the best practice from amongst the set of executed traces. However, the desirable trace may not have been directly observed in the traces but could be configured through the composition of desirable subsets of different traces.

The main purpose of this paper is to propose an approach that would analyze the event logs of multiple peer-organizations in order to compose a desirable execution trace from the event logs that would satisfy a utility function such as execution time, or cost. In order to enable this, we provide the following core contributions:

- Given the focus of our work is to compose a desirable execution trace from a set of event logs and an utility function, it is imperative that the produced solution respects the constraints of the input process models. We propose techniques that are able to extract such constraints from the traces of event logs, as they pertain to the execution order of activities.

- We propose a Genetic algorithm-based approach that is able to enforce the constraints identified from event logs, and compose an execution trace with regards to the desirable utility function.

The final outcome of our proposed approach is an execution trace that respects all of the constraints that are observed in the event logs and also optimizes an utility function. Such identified execution trace can be viewed as a best practice for the desired objective. For instance, based on the event logs of two peer-organizations and with the objective of minimizing time-to-completion, our proposed approach would compose an execution trace that satisfies all of the constraints observed in the two event logs, e.g., co-occurrence or precedence relations, and has the minimal time-to-completion.
1.1 PRELIMINARIES

Before providing a concrete problem statement, we would like to include some preliminary definitions that would clearly define the terminology used across the paper. Our work is focused on processing event log data, which is formalized as follows:

Definition 1 (Event). An event \( e \) is a record of the execution of an activity (task) within a business process instance of an information system. Each event has a label (activity name), a case Id and a timestamp. A case Id is an identifier, which specifies that the event belongs to a particular case, i.e., a process instance. A timestamp shows either the completion or starting time of the event.

Definition 2 (Event trace). A trace \( t \) is a finite sequence of activities whose corresponding events belong to the same execution of a business process.

Definition 3 (Event log). An event log \( \mathcal{L} \) is a set of event traces.

Definition 4 (Fragment) [29]. A fragment of a trace is a sub-sequence of the trace, which can be shown as a tuple \( \langle a_0, S, a_n \rangle \), where \( a_0 \) is a starting activity, \( a_n \) is an ending activity and \( S = \langle a_1, a_2, ..., a_k \rangle \) is a sequence of intermediate activities (where \( a_1 \) is followed by \( a_2 \), \( a_2 \) is followed by \( a_3 \) and in the same way \( a_{k-1} \) is followed by \( a_k \)).

A trace \( t \) can be rewritten as \( t = \langle E_1, E_2, ..., E_n \rangle \) where \( E_i \) are activities or fragments (elements).

Definition 5 (Trace efficiency). For any given trace \( t \), we measure efficiency in terms of inverse of the execution time of the trace, i.e., the amount of time required to transit from the starting activity to the ending activity.

It should be noted that in this paper, we assume that utility functions will be defined based on Definition 5, which concerns execution time. However, future work can focus on benefiting from other metadata in event logs, if available, for defining extended models of trace efficiency.

Definition 6 (Trace desirability). A desirable trace among a set of traces is one that has the highest trace efficiency value. Therefore, a desirable trace would have the least execution time.

1.2 THE PROBLEM STATEMENT

In this paper, we are interested in composing a desirable trace from a collection of peer-organization event logs. The input of our problem is a collection of event logs from peer-organizations and the output is one desirable trace according to user’s utility function (in our experiments, we consider execution time as the utility function). In peer-organizations such as municipalities, there exist processes that have commonalities. However, there might also be some degree of difference between them as well. Some of these differences are due to reasons such as demographic differences. However, in many cases, the reasons are related to managerial decisions or local customizations made to the business process. The process of developing a desirable trace would need to include the identification of the most efficient segments of a business process from each peer-
organization’s event logs such that they can be subsequently composed into a new desirable trace. For example, in Figure 1.a, four different variants of the same process are shown. Each variant has an execution time, which is shown in terms of the number of days. As can be seen, organization $S_1$ has the most efficient execution time, when minimum execution time is desirable (10 days). So, this can be recommended as a best practice; however, this choice overlooks the fact that in the real world, there may be cases when none of the peer business processes are optimal on their own but each might have desirable sub-processes that if put together would create the desirable solution. For example, in Figure 1.b, each variant can be separated into two fragments. In this case, $S_4$ has the minimum execution time for the first segment (4 days) and $S_1$ for the second segment (3 days). Therefore, if $S_4$ and $S_1$ are composed, the resulting process will have a lower execution time (7 days).

The goal of this paper is to propose an approach that helps users (business analysts) to improve a business process using previous execution experiences of the same processes within different peer-organizations. This is done by mining desirable subsequences of event traces from event logs and composing them together such that both process constraints and user requirements are respected and at the same time the composed trace has a minimum execution time. The functional requirements of the generated desirable trace are twofold: 1) the desirable trace should include a set of user-defined elements; and 2) the desirable trace should respect the constraints of the original business process. We will discuss in the paper how these constraints can be mined from event logs. The non-functional requirement of the generated desirable trace is its optimality as defined in Definition 6.

The rest of this paper is organized as follows. Section 2 reviews the related works while Section 3 presents a running example. In Section 4, we present the details of the proposed approach. Section 5 outlines our evaluation plan and the details of the results. In Section 6, we present the details of the developed tool support and the paper is finally concluded in Section 7.

2. RELATED WORK

In this section, we group the related works into two sections: first, the works related to configurable process model discovery are presented and then, the works, which are related to Key Performance Indicator (KPI) analysis are reviewed.

2.1 CONFIGURABLE PROCESS MODEL DISCOVERY

One of the common approaches that has recently been adopted for addressing the common process needs of similar peer organizations is the development of reference business processes. Such reference processes are viewed as the representation of the valid variants of a process [9]. As discussed in [10, 11], process variants must have at least one common and one variable aspect. In [12], the authors review the challenges, scenarios and algorithms of

<table>
<thead>
<tr>
<th>Method</th>
<th>Automation</th>
<th>Activity-Level Configuration</th>
<th>Fragment-Level Configuration</th>
<th>Automatic Elicitation of Fragments/Activities’ Relationships</th>
<th>Consideration of Objective Function</th>
<th>Based on Event Logs</th>
</tr>
</thead>
<tbody>
<tr>
<td>[5]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>[13]</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>[14]</td>
<td>✓</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>[16]</td>
<td>x</td>
<td>✓</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>[17]</td>
<td>✓</td>
<td>x</td>
<td>✓</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>[18]</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>[19]</td>
<td>✓</td>
<td>x</td>
<td>✓</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>

Table 1. Comparison of existing process configuration methods.
mining a collection of process variants. They also proposed two scenarios for discovering a reference model. In the first, they provide a reference model and a collection of related process variants and try to improve the reference model in such a way that it better fits the process variants. In the second, they compose a reference model from among a collection of process variants. They use a matrix representation for merging process variants. This approach is not based on event logs. In [20], a complete survey of methods that allow for the configuration of process variants has been presented. Furthermore, Valenca et al. [21] report a mapping study on process configuration research through which more than 700 works are assessed from different perspectives.

In Table 1, we systematically compare eight of the most highly cited works on process configuration using the criteria introduced in [20, 21]. These criteria are as follows: (1) **automation**, which explores whether the technique can be automatically performed or not, (2 and 3) **activity and fragment level configuration**, which specify whether the configuration is performed on activities or fragments, (4) **Automated elicitation of activity/fragment’s relationships** that discusses whether the approach can identify relations or constraints that might have been imposed on the process elements, (5) **consideration of an objective function**, which explores whether the method can work with any objective function or is limited to a certain type or class of objective functions, and last, (6) **based on event logs or not** that shows whether the algorithm or technique operates on the business process structure or the business process event logs. All of the configuration approaches except [5, 19] are based on the structure of the process model where a set of variation points are extracted and each of the variation points is assigned to a set of configuration alternatives. Each configuration can result in one or more variants.

In contrast, in [5] and [19], the authors have used event logs instead of process model structure to mine configurable business process models. Using event logs as input enables the proposed techniques to more closely capture the real behavior of the business process. For instance, in many cases the event logs, which represent the actual course of actions that were taken when executing a process, do not fully align with the actual business process model. This can be due to many reasons such as ad-hoc changes or manual circumvention of lengthy activities. In [5], a conceptual model for the configuration of process models based on event logs has been proposed, which uses Genetic Algorithms [22] for extracting a configurable process model from amongst a collection of process event logs (also see [23]). The authors consider tree-based change operators to act as the mutation function and propose a fitness function based on both optimization and process discovery criteria. The algorithm would terminate when the quality of the best candidate reaches a minimum threshold. This approach is focused on discovering a business process model.

In [19], the authors propose an approach that works primarily with event logs and has been implemented as a plugin for the ProM framework in order to assist designers in the process of building configurable fragments. In order to elicit a configurable process fragment, the authors first extract a neighbourhood context log for a specific activity (determined by the designer) from each event log. The neighbourhood context log of a specific activity represents the portions of traces in the event logs that contain that activity and its neighbours. Then, they merge all of the extracted neighbourhood context logs into one sub-log and use this sub-log as an input for an existing process mining algorithm in order to discover the corresponding process fragment. Finally, they propose an intuitive approach for identifying configurable gateways from the extracted fragments using shared activities (the activities that belong to multiple neighbourhood context logs). There are some other more recent works such as [24, 25] that propose configurable models based on such neighbourhood context. However, these approaches do not consider user preferences, e.g., shortest time, in their configuration process.
2.2 KPI AND PERFORMANCE METRICS ANALYSIS BASED APPROACHES

The reason that we present this class of papers in the related work is twofold: first, some of these approaches consider performance metrics or KPIs during process discovery or have the capability of discovering business processes with regards to performance indicators. Second, our proposed approach for composing desirable traces has some similarities with these methods. There can be several factors influencing process performance in organizations. In [26], the authors have proposed an approach to detect causal direct/indirect factors of business process performance from event logs. They first extract all possible process characteristics from event logs, and decompose events based on shared properties (single or combinations of properties). Then, they analyze the causal effects of each event group on other groups using time series analysis. However, one factor that we are interested to explore in this paper is whether the way in which a process is executed in other peer-organizations can be used to enhance process execution in other organizations. As previously said, there can be several performance indicators such as cost, duration, and time among others. A complete literature review on process performance measures can be found in [27]. In our paper, we consider the execution time of a process as its performance indicator.

In [28], the authors proposed an approach for extracting best processes according to some quality criteria. For this purpose, a comparison table, which compares event logs and process models, is used. The rows of this table are event logs and the columns are process models and for each combination of event log and process model, a comparison metric is used. Some quality criteria for event logs and process models are also used. Using this table, the best process is identified based on users' preferences and the quality of the event logs. However, in many cases, other peer-organizations only perform some similar parts of a process and not necessarily all of the process in the same way and therefore, finding two completely similar organizations can prove to be a difficult task. So, selecting one process from among the existing ones as the best practice process might not necessarily be desirable.

In [29], the authors used sequence classification to identify differences in the process control flow among different peer-organizations (municipalities). For this purpose, they detect high value and low value cases based on execution time. Then, they extract sequential patterns that are more frequent in each class (extract the effective sequence patterns). Most of the sequence patterns that they detect have a length of one, two or three activities. However, their approach does not provide a solution for composing the most effective sequences.

In [30], the authors have proposed a technique for flow performance analysis from event logs. This approach breaks down a business process into a series of queues corresponding to stages (each stage is related to some performance characteristics calculated at each time point) defined by the user. Then, they visualize and analyze the evolution of these characteristics over time. This approach analyzes the overall performance of a business process over time. A key limitation of this approach is the assumption that the log is divided into user-defined stages. Automated identification of candidate stages from a log can improve this approach.

In [31], a fuzzy clustering based technique for process mining and performance analysis is introduced. The authors first cluster events and derive a simplified process model. Then the event logs are replayed to measure performance metrics. However, this approach, similar to most of the other techniques in the literature, is designed to analyze only one event log as opposed to several event logs from different peer organizations.
2.3 OTHER RELATED WORK

Some other work, including [15], have adopted a questionnaire-based variability modeling approach for configuration. In such work, the users’ answers are employed to bind variation points; therefore, feedback is elicited from the users, which is in the form of alternative-choice questions. It is noteworthy to mention that most of the automated approaches support activity-level configuration. Activity-level process configuration is more complex compared to fragment-level configuration [9]. Because in this case, the consolidated model grows rapidly, especially, when the input variants grow.

3. A RUNNING EXAMPLE

In this paper, we employ a running example from a “purchasing order” process in order to explain the various aspects of our proposed approach. Figure 2 presents the structure of the purchasing order process. The process starts by “filling a request form” and ends with “sending material” activity or request rejection. After the “filling a request form” activity is executed, two activities can be performed simultaneously: “stock will be checked” and “price calculations”. If the requested material is not available in stock, the process will terminate, else, the invoice will be sent (activity D). Afterwards, there are two parallel activities: “Other material recommendation” and “payment”. There are two choices for payment: cash and online payment. Finally, the receipt and material will be sent (activities H and I). This process is a simplified purchasing order process and is used just for the purpose of demonstrating the details of our work and does not necessarily show the complexity of real world process models. Figure 2-b shows some sample event log entries for the business process shown in the same figure.

4. THE PROPOSED APPROACH

In this paper, we propose an approach for composing a desirable execution trace of several peer-organizations according to a utility function such as cost or execution time by analyzing their event logs. We assume that there are several peer-organizations such as municipalities that execute different variants of the same process. We address the problem of composing the desirable execution trace by identifying the best subset of each peer-organization’s process from its event logs and composing the extracted subsets together in such a way that the structural constraints of the processes are respected and the utility function is optimized. The benefit of our proposed approach is that it can find desirable results, which may not have been observed in any of the business process event logs, but would be obtained only if the best subsets of each of the event logs are put together.
The overall process of our proposed work is shown in Figure 3. The process consists of four main steps, namely Steps A to D. As a preprocessing step for our work, we employ an existing business process fragment extraction technique, namely [32], to identify the common fragments from various peer organizations’ event logs. Once the common fragments are identified, we rewrite the event logs using the identified common fragments in the first step (A). In step B, we identify possible relationships that might exist between the fragments/activities as observed in the pre-processed event logs. As already discussed in [33], if certain relations are always observed between activities or fragments, they could be an indication of possible constraints imposed from the business process. For example, if the event logs show that one activity always appears before another activity, then that could be a sign that such a constraint needs to be enforced on any trace that is to be composed. For this reason, once the relationships between the activities and/or fragments are identified, we will employ this information to model activity/fragment constraints in step C. Finally, in step D, we consider the identified constraints and a given utility function in order to generate a desirable execution trace through a Genetic algorithm. We first randomly generate a trace set (initial population) and iteratively apply crossover and mutation operators to generate new solutions until an acceptable execution trace is generated.

**4.1 PRE-PROCESSING OF EVENT LOGS**

It is typically the case that event logs consist of thousands of traces, most of which are similar. Processing all of the redundancies when processing an event log can prove to be inefficient. Process-oriented event logs might contain many executions of a business process, many of which exhibit themselves as the repeat of the exact same trace in the event log. While it is important to consider the frequency of these traces in some problems, in our case, frequency is not considered when extracting relation constraints between elements. Therefore, as a preprocessing step, we extract distinct subsets of the traces, which are frequently observed throughout the event logs before extracting constraints; however, when calculating the execution time of fragments we consider all traces.

Loop regions are also considered. We use the proposed solution in [8] for detecting loops. We classify “loop regions” into three classes:
• **Single loop:** for example suppose that we have a trace such as “ABCCCCCCCDD”. In this trace, the activity “C” has a single loop. In such cases, since the repetition of “C” does not affect the ordinal or co-occurrence relations, we remove the repetition, which leads to “ABCD” in this example. However, it should be noted that we do consider the execution time of these repetitions for calculating execution time. Indeed, although we do not know the logic and reasons behind these repetitions because we do not have access to the actual business process, however we consider the impact of these repetitions by updating execution time of the repeated activity.

• **Loops with the same repeated patterns:** in some cases there are traces that include loops with the same repeated patterns such as: “ABCDCCDCDCH”. In this example, “CD” is repeated several times. Same as single loops, we remove repetitions, which leads to “ABCDH”. We calculate the execution time similar to the previous case.

• **Loops which have different patterns:** in some cases there might be traces that have loops with different patterns such as: “ABCDCECDH”. In this example, “CDCECD” is a loop region. However, because of the existence of “E”, the repeated patterns are different. In such cases, we do not make any changes to the traces.

We employ the method that we have proposed earlier in [8] to extract common fragments. The added advantage of this method is that it is able to identify fragments that are not completely structurally the same but perform similar tasks, referred to as **morphological fragments**. Morphological fragments are more formally defined as follows [8]:

**Definition 7.** Two fragments $f_1 = \langle a_s, S, a_e \rangle$ and $f_2 = \langle \hat{a}_s, \hat{S}, \hat{a}_e \rangle$ are behaviorally similar, called morphological fragments, iff:

- $a_s = \hat{a}_s$
- $a_e = \hat{a}_e$
- $\text{set}(S) = \text{set}(\hat{S})$.

Based on this definition, two fragments are morphologically identical if they have the same start and end activities and also have the same middle activities set. In the above definition, the function “set” returns the set of activities of the input sequence. For example, suppose fragments $f_1 = \langle a, < b, c >, d \rangle$, $f_2 = \langle a, < c, b >, d \rangle$, $f_3 = \langle a, < b, c >, h \rangle$ and $f_4 = \langle a, < b, e, c >, d \rangle$. Here, $f_1$ and $f_2$ are morphological fragments (have the same start activity “a” and same end activity “d” and also set($< b, c >$)=set($< c, b >$)). However, $f_1$ and $f_3$ are not morphological fragments because their ending activity is different (because of the same reason $f_2$ and $f_3$ are not morphological either). Also, $f_1$ and $f_4$ are not morphological fragments because set($< b, c >$)$\neq$set($< b, e, c >$).

Once such fragments are identified, we consider all of them as elements and replace each occurrence of them in the traces with corresponding element name (which is unique). For example, let us consider organization $S_1$ shown in Figure 1. For the sake of this example, we assume that trace $A B C D E F H I$ contains one fragment that is exemplified with an underline. We refer to this fragment as $F_1$. After replacing this fragment with its label, the new trace would be rewritten as $A B C F_1 H I$. This will allow us to identify the relationship between activities and fragments.
4.2 EXTRACTING CONSTRAINTS

Given the objective of our work is to find a desirable execution trace for a family of business processes from their event logs, we need to ensure that the desirable solution that we produce does not violate any of the constraints of the business processes. In various papers, according to their application, different constraints are proposed based on basic workflow patterns [34]. Most of them share several basic constraints. In [35], the well-known Alpha algorithm for process model discovery from event logs is proposed. This algorithm uses four main relation types, which we have included in Table 2 (relations 1 to 4). These four relations are direct succession, causality, parallel and choice, respectively. The Alpha algorithm redisCOVERs the process model by detecting flow patterns, e.g., AND, XOR, Sequence and Loop using these relations. For example, if after activity \( A \) there is a choice between \( B \) and \( C \) (\( A \rightarrow B, A \rightarrow C \) and \( B \leftrightarrow C \)), then it would extract an XOR relation between \( B \) and \( C \) which is preceded by \( A \). It has already been theoretically shown that the Alpha algorithm is able to redisCOVER a large class of sound SWF-nets [36]. Using these four basic relation constraints, higher-level constraints can be derived. In our work, we respect these constraints when composing desirable traces.

It should be noted that our problem is different from process discovery and we need to define constraints according to our problem conditions. So, we validate the generated desirable traces by considering AND, XOR, Sequence and Loop structural constraints. For this purpose, we directly adopt relations 1 and 2 from Table 2, and propose a revision to relation 3 by introducing relation 5. Here, if there exists a case where \( E_a \) was followed by \( E_b \) and also other cases where \( E_b \) was followed by \( E_a \), then \( E_a \) and \( E_b \) have a parallel relation (\( E_a \leftrightarrow E_b \)). Relation 5 specifies the co-occurrence relation over traces (regardless of their order). Also, we propose relation 6 instead of relation 4. If \( E_a \) and \( E_b \) have the \( E_a \uparrow E_b \) relation (relation 6), then they would definitely have the \( E_a \leftrightarrow E_b \) relation as well (relation 4), but not vice versa. Since, the Alpha algorithm extracts local patterns from traces, it is only able to identify whether \( E_a \) and \( E_b \) have been seen ‘immediately’ after each other or not. However, in our proposed approach, for each element such as \( E_a \), we are interested in extracting all elements such as \( E_b \) that are never observed with \( E_a \) regardless of immediacy.

The relations presented in Table 2 can be extracted from event logs by considering the order in which the elements occur. A relationship matrix can, therefore, be derived from the event logs, which we will use later for extracting such constraints. We use this relation matrix for extracting higher-level relations in the form of relational constraints between elements. We classify constraints between elements in two classes, namely ordinal constraints and co-occurrence constraints. The first set of constraints refers to those constraints that are related to the order of elements in the event logs (relation 1 and 2 in Table 2). For example, if \( E_b \) is always seen after \( E_a \), then this should be considered to be a constraint on the final solution. In other words, the desirable

<table>
<thead>
<tr>
<th>Row</th>
<th>Relation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( E_a \uparrow E_b )</td>
<td>If there is at least a case where ( E_b ) was seen immediately after ( E_a ) (there exist some cases that ( E_a ) is followed by ( E_b ))</td>
</tr>
<tr>
<td>2</td>
<td>( E_a \rightarrow E_b )</td>
<td>If ( E_a \rightarrow E_b ) and not ( E_b \rightarrow E_a ).</td>
</tr>
<tr>
<td>3</td>
<td>( E_a \leftrightarrow E_b )</td>
<td>If we have ( E_a \rightarrow E_b ) and ( E_b \rightarrow E_a ).</td>
</tr>
<tr>
<td>4</td>
<td>( E_a \leftrightarrow E_b )</td>
<td>If we do not have ( E_a \rightarrow E_b ) or ( E_b \rightarrow E_a ) relations.</td>
</tr>
<tr>
<td>5</td>
<td>( E_a \uparrow E_b )</td>
<td>If ( E_a ) and ( E_b ) were always seen with each other (in any order) in all domain traces.</td>
</tr>
<tr>
<td>6</td>
<td>( E_a \uparrow E_b )</td>
<td>If ( E_a ) and ( E_b ) are never seen with each other (in any order) in all domain traces.</td>
</tr>
</tbody>
</table>

Table 2. Some of the main relations between two elements \( E_a \) and \( E_b \).
solution would also need to ensure that if $E_a$ or $E_b$ are observed, that it would also guarantee that $E_b$ will always be seen after $E_a$. The second class of constraints is related to the number of occurrences of elements (relations 5 and 6 in Table 2). For instance, if $E_a$ and $E_b$ are always seen together in the event logs (not necessarily immediately or in any specific order), such consistent co-occurrence should be taken into consideration. In other words, $E_a$ should never appear without $E_b$ in a desirable solution. The reason that we handle these two classes of constraints separately is related to the way we will propose to identify a desirable solution.

It should be noted that in composing a desirable trace, we should consider the constraints that exist amongst the executed event logs of all organizations. So, for extracting behavioral and co-occurrence constraints, we consider all event logs of all organizations together. In the following two subsections, we specify the details of the algorithms that extract each of the two constraint types.

### 4.2.1 Extracting Ordinal Constraints

For extracting ordinal constraints, we create a relationship matrix over all traces using relations 1 and 2 of Table 2 as shown in Figure 4.a. In this matrix, $M_O^{t_{\times n}}$, $n$ is the number of elements (including both the number of activities and fragments). Each cell of the relationship matrix shows only one relation. Since a causal relation overrides direct succession, in the case that there is causal relation between two elements, only the causal relation will be shown (similar to the Alpha algorithm). For each element $E_i$, we extract a set of ordinal constraints denoted by $E_i^{OC}$ from $M_O$. First, we specify the elements that are always seen as the first and last activities in the traces. From an ordinal occurrence point of view, these elements should always be the first and last elements of the desirable solution. An element such as $E_s$ is considered to be a start element if there does not exist any other element such as $E_j$ that has the relation $E_j \rightarrow E_s$. Also, $E_e$ would be considered to be an end element if there are no other elements such as $E_j$ which have the $E_e \rightarrow E_j$ relation with $E_e$. For these elements, we add a rule $\{Start\}$ (specifying the start element) or $\{End\}$ (specifying the end element) to their ordinal constraint set. Furthermore, for every $E_i$ in $M_O$, if $E_i$ has the relations $\rightarrow$ or $\leftarrow$ with $E_j$, we add $E_i$ to the ordinal constraint set of $E_i^{OC}$.

Based on these constraints, we propose Algorithm 1 for extracting ordinal constraints for each element $E_i$ from the ordinal relation matrix. This proposed algorithm works with a relationship matrix $M_O$ and elements list $E$ as input and generates a set of ordinal constraints for each element. In this algorithm, an empty set $E^{OC}$ is first created for each $E_i$. Then, for each $E_j$, based on the mapping rules in Table 2, if there is an ordinal relation between $E_i$ and $E_j$, the corresponding ordinal rule will be added to $E^{OC}$ (Lines 9-11 in Algorithm 1). In Line 5 of Algorithm 1, if $E_i$ is detected as a starting element, the rule $\{Start\}$ will be added to $E^{OC}$. Also, if $E_i$ is an end element, $\{End\}$ will
be added to \( \mathcal{E} \) (Line 7). In Figure 5.c, the ordinal relation matrix of trace set of Figure 5.a is shown. Using this matrix, it can be understood that element \( D \) is a starting element and so \{Start\} is added to \( D \). The reason is that no other elements such as \( \mathcal{E}_i \) can be found such that the \( \mathcal{E}_i \rightarrow D \) relation can be observed. Also, it can be inferred that element \( I \) is an end element, because there are no other elements such as \( \mathcal{E}_i \) that have the \( I \rightarrow \mathcal{E}_i \) relation. In Figure 5.j, the extracted constraints for each element is shown.

For instance, for element \( D \), the constraint set consists of elements \( E \), \( F \) and \( G \).

### 4.2.2 EXTRACTING CO-OCCURRENCE CONSTRAINTS

In light of the fact that process model constraints are not directly exposed at the event log level, we also identify possible co-occurrence constraints for each element using its relationship matrix. For this purpose, after rewriting the traces in the event logs in the preprocessing step, for each element \( \mathcal{E}_i \), at first we filter the traces that do not include \( \mathcal{E}_i \). Then, we extract the relationship between \( \mathcal{E}_i \) and other elements that are observed in the remaining traces. In order to represent the relationships, we employ a matrix representation format \((\mathcal{M}^C[i])\) for element \( \mathcal{E}_i \) shown in Figure 4.b. The rows and columns of the matrix are the observed elements and the cells represent the relationship between the corresponding row and column elements. The number of created matrices is equal to the number of elements. However, each element matrix has a different size. The reason is that the remaining traces for each element \( \mathcal{E}_i \) might include a different number of elements depending on the remaining traces.

In order to identify and detect co-occurrence constraints, we use relations 5 and 6 in Table 2. If two elements \( \mathcal{E}_a \) and \( \mathcal{E}_b \) have the \( \mathcal{E}_a \leftrightarrow \mathcal{E}_b \) relation, it means that they have always been seen together in all traces (in some order) and therefore would also need to appear together in a desirable solution. Also, the constraint \( \mathcal{E}_a \uparrow \mathcal{E}_b \) prevents the co-occurrence of two elements that have never been seen together in any of the traces (for each element \( \mathcal{E}_a \), there is the \( \uparrow \) relation with \( \mathcal{E}_b \) if \( \mathcal{E}_b \) does not exist in co-occurrence matrix of \( \mathcal{E}_a \)). So, these elements should not appear together in a desirable solution. However, there could be subtler co-occurrence relations that cannot be identified so directly. For instance, consider an element that co-occurs with an XOR-combination. In such a case, the element is not occurring with the same element but rather with one from a set of elements. Figure 5 (Figures 5.a and 5.b) clearly shows such a case. For instance, activity \( E \) has always been observed with activities \( D, H \) and \( I \). So, these activities would have a co-occurrence constraint with \( E \). However, while Activity \( E \) does not always co-occur with \( F \) and \( G \), it is clear that it does always co-occur with their XOR-combination. So, the correct co-occurrence constraint set for \( E \)
would need to be represented as $\mathcal{E}_{cc} = \{(D), (H), (I), (F, G)\}$. This representation means that if there is activity $E$ in the composed trace, activities $D, H, I$ and one of the activities $F$ or $G$ should also be included.

In order to not only be able to identify the obvious co-occurrence constraints but also cases such as the one seen in Figure 5, we propose an algorithm for extracting co-occurrence constraints shown in Algorithm 2. For each element $\mathcal{E}_i$, the algorithm receives as input a matrix $\mathcal{M} \subset \mathcal{E}$ and generates a set of co-occurrence constraints. In Figure 5 (Figures 5.d, 5.e, 5.f, 5.g, 5.h and 5.i), we show the co-occurrence matrix of activities $D, E, F, G, H$ and $I$. For all $\mathcal{E}_i$ that do not exist in $\mathcal{M} \subset \mathcal{E}$, we add $\langle \forall \mathcal{E}_j \rangle$ to $\mathcal{E}_{cc}$ of element $\mathcal{E}_i$ (Line 6 in Algorithm 2). For example, in Figure 5.f, the co-occurrence matrix of $F$ does not include element $G$. So, we add $\langle \forall G \rangle$ to co-occurrence constraints set of $F$ (Figure 5.k). This means that if the composed trace contains element $F$, it cannot simultaneously include $G$. Next, for extracting co-occurrence constraints of $\mathcal{E}_i$, we add all elements such as $\mathcal{E}_j$ that have the $\mathcal{E}_i \leftrightarrow \mathcal{E}_j$ relation to $\mathcal{E}_{cc}$ (Line 9 in Algorithm 2). In Figure 5, looking at the co-occurrence matrix of $D$, it can be understood that elements $E, H$ and $I$ ($D \leftrightarrow E, D \leftrightarrow H$ and $D \leftrightarrow I$) co-occur with $D$ and would be added to its constraint set (Figure 5.k).

For each element $\mathcal{E}_i$, we can extract all of its co-occurrence constraints as follows: for each element $\mathcal{E}_j$ in rows of $\mathcal{M} \subset \mathcal{E}$, we create a list $\mathcal{L}$ and add $\mathcal{E}_j$ to $\mathcal{L}$ (Line 9 in Algorithm 2). Then, if there is an element such as $\mathcal{E}_k$ where $\mathcal{E}_j \uparrow \mathcal{E}_k$, we will add $\mathcal{E}_k$ to $\mathcal{L}$ (Lines 13 and 14 in Algorithm 2). In Lines 15 and 16, we will add each $\mathcal{L}$ to $\mathcal{E}_{cc}$ and remove duplicate sets. Also, if there are sets such as $\mathcal{L}_a$ and $\mathcal{L}_b$, where $\mathcal{L}_a \subseteq \mathcal{L}_b$, then we will remove $\mathcal{L}_a$ (Line 16 in Algorithm 2). For example, for element $D$ in Figure 5.d, elements $F$ and $G$ have $F \uparrow G$ based on the co-occurrence matrix of $D$. So, we add the constraints $\{F, G\}$ and $\{G, F\}$ to $\mathcal{E}_{cc}$ of $D$. Since, $\{F, G\}$ and $\{G, F\}$ are equivalent, one of
the redundant sets will be removed. The final co-occurrence set of $D$ and other elements are shown in Figure 5.k. Finally, in Lines 17 and 18, we add $E^{cc}$ to the output set $C_c$ and return it.

In the following, we theoretically prove that the proposed approach correctly extracts constraints (soundness). In this regards, we use the proof by contradiction method. First, we introduce two lemmas.

**Lemma 1 (Soundness of XOR Relations).** The proposed approach for extracting constraints is able to correctly identify XOR relations between any set of elements.

**Proof.** Suppose that the lemma is not correct, meaning that there are at least two elements such as $E_i$ and $E_j$ for which the proposed method has incorrectly detected the XOR relation, i.e. $(E_i \uparrow E_j)$. If $E_i$ and $E_j$ do not have an XOR relation, then they either have a sequence or AND relation or have never been seen together. If these elements were never seen together in any trace, then they have exhibited the behavior of an XOR relation. If $E_i$ and $E_j$ have the sequence or AND relations, then one of the following relations must have been observed in the event logs:

- $E_i \leftrightarrow E_j$
- $E_i \Rightarrow E_j$
- $E_j \Rightarrow E_i$
- $E_i \Leftrightarrow E_j$

but not the $\uparrow$ relation. Given the proposed method only detects an XOR relation based on the $\uparrow$ relation, the assumption of the proof will be in direct contradiction with $E_i \uparrow E_j$ behaviour that has been seen in the logs. $\square$

**Lemma 2 (Soundness of AND Relations).** The proposed approach for extracting constraints is able to correctly identify AND relations between any set of elements.

**Proof.** In a similar way, suppose that the lemma is not true, meaning that there is at least two elements such as $E_i$ and $E_j$ for which the proposed approach has incorrectly detected the AND relations. The proposed approach detects an AND relation only if these relations were seen in the event logs:

- $E_i \leftrightarrow E_j$
- $E_i \Leftrightarrow E_j$

If $E_i$ and $E_j$ do not have an AND relation, then they have sequence or XOR relations or have never been seen together. If the two elements have a sequence relation, then there should be a $\rightarrow$ relation and is inconsistent with the $\leftrightarrow$ relation. If they have an XOR relation or were never seen with each other in any other trace, they should have a $E_i \uparrow E_j$ relation which is in contradiction with $\leftrightarrow$ or $\leftrightarrow$ behaviour seen in the event log. $\square$

**Theory 1.** The proposed approach for extracting constraints is sound.

**Proof.** For proving this theory, we should demonstrate the following three cases:

- The approach detects XOR relations correctly (proven in Lemma 1).
- The approach detects AND relations correctly (proven in Lemma 2).
- The approach detects any combination of XOR and AND relations correctly.

The first and second statements were proven in Lemmas 1 and 2, respectively. Also, situation 3 is a nested combination of XOR or AND relations over cases 1 and 2 and is therefore being shown to hold.
In the next section, we will use the two classes of constraints to formalize a GA algorithm for composing a desirable execution trace for a family of business processes based on their event logs.

4.3 COMPOSING A DESIRABLE EXECUTION PATH

Now that we have been able to extract the required constraints from the event logs, we propose a Genetic algorithm whose solution would represent a desirable execution trace for a family of business processes extracted from their event logs optimized based on a given utility function; the solution of which would also respect the identified constraints. The desirable solution would need to begin from a given start activity and terminate with a specified end activity. The desirable path(s) should comply with the two sets of constraints that were identified earlier in the previous section. The co-occurrence constraints do not consider the order of elements and only contain information about the co-occurrence. However, the reason that our generated trace is valid is that we do not consider only co-occurrence constraints. But, at same time we consider the ordinal constraints in generating desirable traces too.

The problem of generating a desirable execution trace is naturally an NP-hard optimization problem and therefore we employ a Genetic Algorithm (GA) model to address it. A GA is a search-based process for finding desirable solutions belonging to the larger class of evolutionary algorithms [39][40]. Similar to any GA process, there are four main steps: i) initialization, ii) selection, iii) genetic operator, and iv) termination. In the following, we discuss how the GA method is customized for our problem. The pseudocode of proposed algorithm is shown in Algorithm 3. The algorithm gives as input pre-processed event log $\mathcal{L}$, the iteration number $\iota$, ordinal constraints $\mathcal{C}_o$, and co-occurrence constraints $\mathcal{C}_c$. In Line 3, the INITIALIZE function generates the initial population $\varphi$ as described in Section 4.3.1. Then in Lines 4 and 5, the initial population are evaluated and best individual are selected (Section 4.3.2). Next, for each iteration from 1 to $\iota$, following commands are executed: for each parents $p_1$ and $p_2$, the crossover function is executed(Section 4.3.3). Also, with the probability of 1%, the mutation function will perform in lines 10-12 (Section 4.3.3). In Lines 14 and 15 the new generated population are evaluated and best of them are selected. Finally, the old population $\varphi$ will be replaced by new generated population set $\varphi^\ast$. At each iteration, the termination condition will be checked using the TERMINATION($\varphi$) function (Section 4.3.4). In line 18, the result $\varphi^\ast$ will be returned.
4.3.1 INITIALIZATION

The first step towards the formalization of our work is the representation of the solutions in the GA model as a set of chromosomes. Here, we formally define the chromosomes as follows:

**Definition 8.** A chromosome $\mathcal{C} = \langle \mathcal{E}_0, \mathcal{E}_1, \ldots, \mathcal{E}_n \rangle$ is a sequence of elements that begin with a start element $\mathcal{E}_0$ and terminate with an end element $\mathcal{E}_n$ and has at least two elements.

Based on the above definition, we can define a population set in the following form:

**Definition 9.** A population set $\mathfrak{P} = \{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_m\}$ is a set of chromosomes, each of which complies with a set of ordinal and co-occurrence constraints.

We construct half of our initial population set (Definition 9) randomly. For the second half, we select the best execution traces of each set of logs (from each peer-organization) according to the utility function (Definition 12). The literature has already reported that selecting the initial population this way can increase the speed of generating better generations and at the same time avoid falling into the trap of local optimum solutions [37][38].

4.3.2 SELECTION

This selection is done by defining a fitness function where fitter solutions are determined based on the value of the fitness function. The simplest and most useful selection function is to select the solutions that have the highest fitness rate. However, there are several other selection methods such as Elitism, Rank selection, Tournament selection, Boltzmann selection and Sigma scaling [39]. The work in [41] and [43] provide a comprehensive comparison of various selection methods. In [43], the authors show that the tournament selection method outperforms rank-based roulette wheel proportional selections and achieves the best solution quality. Later in our experiments, we execute our proposed approach using two selection functions: Tournament and Elitism.

For the definition of the fitness function, we consider two issues:

- One of the important factors influencing the fitness function is the utility function. The utility function can be defined based on functional or non-functional characteristics of the event logs or a combination thereof. Given the fact that time-related

---

```plaintext
ALGORITHM 3. The Pseudocode of proposed genetic algorithm

INPUT: event log $\mathcal{L}$, the iteration number $\iota$, ordinal constraints $\mathcal{C}_0$, and co-occurrence constraints $\mathcal{C}_C$.
1 GENERATE-DESIRABLE-TRACES ($\mathcal{L}, \iota, \mathcal{C}_0, \mathcal{C}_C$)
2 let $\mathfrak{G} = \emptyset$, $i = 1$
3 $\mathfrak{G} = \text{INITIALIZATION} (\mathcal{L})$
4 $\mathfrak{G} = \text{CALC-FITNESS}(\mathfrak{G})$
5 $\mathfrak{G} = \text{SELECT-BEST}(\mathfrak{G})$
6 while $i \leq \iota$ and $\text{TERMINATION}(\mathfrak{G})$
7 let $\mathfrak{G} = \emptyset$
8 foreach parents $p_1$ and $p_2$ in $\mathfrak{G}$
9 $\mathcal{C} = \text{CROSSOVER}(p_1, p_2)$
10 if $\text{random}(1, 100) == 1$
11 $\mathcal{C}[1] = \text{MUTATION}(\mathcal{C}[1])$
12 $\mathcal{C}[2] = \text{MUTATION}(\mathcal{C}[2])$
13 $\mathfrak{G}^* = \mathfrak{G} \cup \mathcal{C}$
14 $\mathfrak{G}^* = \text{CALC-FITNESS}(\mathfrak{G}^*)$
15 $\mathfrak{G}^* = \text{SELECTION}(\mathfrak{G}^*)$
16 $\mathfrak{G} = \mathfrak{G}^*$
17 $i = i + 1$
18 return $\mathfrak{G}^*$
```
information is usually available in most event logs, in the experiments, we consider the execution time of each trace to serve as the utility function.

The second factor that we consider is the number of fragments. If there are several solutions that have the same execution time, priority would be given to those solutions which have a higher number of fragments as opposed to single activities. The reason is that a solution that has a higher number of fragments is modular. Given the fact that the central factor in our work is the execution time of the traces, we need to first calculate the average execution time of each activity before we can formally define the fitness function. The average duration time of each activity can be defined based on [45] as follows:

**Definition 10.** The execution time of an activity \( A \) is the average of the difference of the completion time of activity \( A \) and its preceding activity and can be calculated as:

\[
E_T(A) = \frac{\sum_{i=1}^{n} C^i_T(A) - C^i_T(\hat{A}_i)}{n}
\] (1)

Where \( C^i_T(A) \) is the completion time of \( A \) in case \( i \), \( \hat{A}_i \) is the preceding activity of \( A \), and \( n \) is the number of cases that include activity \( A \). It is clear that for event logs that in addition to completion time, record start time too, the duration time of each activity is the difference of its completion time and its start time. We also define execution time of each fragment \( F \) in the following form:

**Definition 11.** The execution time of a fragment \( F(a_\phi, S, a_\epsilon) \) is the average of the difference of the completion time of its end and start activities and can be calculated as:

\[
E_T(F) = \frac{\sum_{i=1}^{m} C^i_T(a_\epsilon) - C^i_T(a_\phi)}{m}
\] (2)

where \( m \) is the number of cases that include fragment \( F \). The reason why we do not calculate the execution time of each fragment by adding the execution time of its constituent activities is that we look at fragments as black boxes that can have different execution times according to their internal structure. Suppose that we have two morphological fragments \( F_1 = (A, B, C, D) \) and \( F_2 = (A, C, B, D) \). The summation of execution time of their constituent activities are the same \((\sum_{\epsilon_i \in F_1} E_T(\epsilon_i)) = \sum_{\epsilon_i \in F_2} E_T(\epsilon_i))\). However, execution time of these two fragments might not be equal (we may have \( E_T(F_1) \neq E_T(F_2) \)). This can be due to the difference in execution order of activities \( B \) and \( C \). So, we consider the difference of completion time of the last and first activities of each fragment as its execution time. Also, by considering the common fragments of other peer-organizations, we take the best execution time of the common fragments as their representative. The execution time of a trace can be defined as follows:

**Definition 12.** The execution time of a trace \( t = (\epsilon_1, \epsilon_2, ..., \epsilon_n) \) is equal to:

\[
E_T(t) = \sum_{i=1}^{n} E_T(\epsilon_i)
\] (3)
The execution time of element (activity and fragment) is calculated in Definition 10 and 11, respectively. Finally, the fitness function of a chromosome $ℭ$ has inverse relation to its execution time:

$$\text{Fit}(ℭ) = \frac{1}{E_{T}(ℭ)}$$  

(4)

In cases when several candidate chromosomes have the same execution time, the one with the highest number of fragments will be selected. The primary reason for this is that such chromosomes will be much more modular and share the most number of similarities with a higher number of traces that frequently co-occur across different organizations.

### 4.3.3 GENETIC OPERATORS (CROSSOVER AND MUTATION)

One of the most important aspects of a GA is the crossover operator. There are several methods for crossover such as single point crossover, two-point crossover, cut and splice, three-parent crossover and uniform crossover[42]. The pros and cons of each function is discussed in [44]. Here, in this paper, we use both cut and splice and two points crossover methods. Single point crossover is the simplest operator that selects a single position on both parents. Then, the parts of the two parents are swapped to form two new offsprings. Cut and splice is similar to single point crossover and the only difference is that the crossover position can be different in the two parents. Here, we define the cut and splice crossover as follows:

**Definition 13 (Cut and splice crossover).** A cut and splice crossover function $\text{CSC}(p, q, ℭ, ℭ)$ takes two fraction points $p$ and $q$, and two chromosomes $ℭ$ and $ℭ$ as inputs and generates two new child chromosomes $ℭ_1$ and $ℭ_2$ by composing the first part of $ℭ$ with the second part of $ℭ_2$ and the first part of $ℭ_2$ with the second part of $ℭ$. During these compositions, the following constraints should be respected:

- $\mathcal{E}_{q+1} \in \mathcal{E}_p^\text{OC}$
- $\mathcal{E}_{p+1} \in \mathcal{E}_q^\text{OC}$
- $\forall_{i \in [q+1,...,n]} \mathcal{E}_i \in \forall_{j \in [1,...,p]} \mathcal{E}_{j}^\text{CC}$
- $\forall_{i \in [p+1,...,n]} \mathcal{E}_i \in \forall_{j \in [1,...,q]} \mathcal{E}_{j}^\text{CC}$

Where $\mathcal{E}_p$ and $\mathcal{E}_q$ are elements located at position $p$ and $q$ of chromosomes $ℭ$ and $ℭ$, respectively. The first and second parts of each new child have ordinal constraints in the fusion point ($\mathcal{E}_{q+1} \in \mathcal{E}_p^\text{OC}$ and $\mathcal{E}_{p+1} \in \mathcal{E}_q^\text{OC}$). Also, in both new children, all elements in the first part should respect the co-occurrence constraints with all elements of the second part ($\forall_{i \in [q+1,...,n]} \mathcal{E}_i \in \forall_{j \in [1,...,p]} \mathcal{E}_{j}^\text{CC}$ and $\forall_{i \in [p+1,...,n]} \mathcal{E}_i \in \forall_{j \in [1,...,q]} \mathcal{E}_{j}^\text{CC}$). These constraints guarantee that new composed chromosomes are valid solutions (traces).

In many problems, the crossover point(s) is selected randomly. However, we cannot perform this in a completely random way because we should consider the ordinal and co-occurrence constraints in producing new offsprings. So, first we detect possible crossover points for each parent and then randomly select one or two points (according to cut and splice or two-point crossover methods) for each of them. In Figure 6.a, the cut and splice crossover is shown. In each parent, a point is selected randomly. However, the points should be selected in such a way that both ordinal and co-occurrence constraints are respected in the newly produced offsprings.

Along the same lines, we define two-point crossover as follows:

**Definition 14 (Two-point crossover).** The two-point crossover function $\text{TWC}(p, q, \hat{p}, \hat{q}, ℭ, ℭ)$ takes two chromosomes $ℭ$ and $ℭ$ and four fusion points $p, q, \hat{p}$ and $\hat{q}$ as inputs. Then, $\text{TWC}$ generates two new child chromosomes $ℭ_1$ and $ℭ_2$, by exchanging
Definition 15 (Mutation). The mutation function \( \mathcal{M}(\mathcal{C}) \) takes a chromosome as input and selects a random number \( p \) between 1 and \( |\mathcal{C}| \) (|\( \mathcal{C} \)| is the number of elements of \( \mathcal{C} \)). It then selects a new element \( \hat{\mathcal{E}} \) and replaces \( \mathcal{E}_p \) with \( \hat{\mathcal{E}} \), by considering the following constraints:

\[
\begin{align*}
\mathcal{E}_p &\in \mathcal{E}_{p-1}^C \\
\mathcal{E}_{p+1} &\in \mathcal{E}_p^C \\
\hat{\mathcal{E}} &\in \forall_{i \in [1,...,p-1]} \mathcal{E}_i^C
\end{align*}
\]
The two first terms check the ordinal constraints and the last two check the co-occurrence constraints of the mutant chromosome. For example, the first constraint $\mathcal{E} \in \mathcal{E}^p_{p-1}$ means that element $\mathcal{E}$ can appear immediately after the element $\mathcal{E}_{p-1}$. The constraint $\mathcal{E} \in \forall_{i \in [1, \ldots, p-1]} \mathcal{E}^c_{i}$ means that element $\mathcal{E}$ has co-occurrence relation with all elements $\mathcal{E}_i$, $i \in [1, \ldots, p - 1]$.

4.3.4 TERMINATION

GA is an iterative algorithm; therefore, termination conditions need to be clearly specified. We consider a combination of several conditions for terminating the algorithm:

- A fixed number of iterations.
- A solution is found that has a better utility value than the best trace in the event logs.

The solution to our GA process with the constraints described earlier and the defined utility function will produce a solution that has the fastest execution time derived from multiple event logs of peer organizations.

4.3.5 USER CONSTRAINTS

One of the strengths of the proposed approach is that users can impose their own additional constraints on the final solution beyond the ordinal and co-occurrence constraints that are automatically detected from the event logs. We classify these constraints that a user can impose into four types:

- **Existence Constraints**: users can decide to impose new activities on the final composed trace. This can be implemented by adding a new row to the relation matrix $\mathcal{M}^O_{hn}$.

- **Omitting Constraints**: users can also decide what activities should not be present in the final trace. This can be done by deleting the related row and column of the considered activity from the relation matrix $\mathcal{M}^O_{hn}$ and updating the other rows and columns of $\mathcal{M}^O_{hn}$. Also, for each element $\mathcal{E}_i$, the matrix $\mathcal{M}^E_{ij}$ should be updated.

- **Co-occurrence Constraints**: another type of constraint that the users can impose is co-occurrence constraints, defined in Section 4.2.2. Users can change the co-occurrence activities of a specific activity. However, it should be checked that the new co-occurrence constraints do not violate the existing constraints. For example, if $\mathcal{E}_i$ and
ℰᵢ are co-occurrent activities, if we define ℰⱼ as co-occurrent with ℰᵢ, then ℰᵢ and ℰⱼ should also have a co-occurrence constraint with each other.

- **Ordinal Constraints**: Similar to co-occurrence constraints, users can define or change ordinal constraints. Again, it should be checked that new constraints do not have conflict with ordinal constraints. For example, if there is a relation such as ℰᵢ → ℰⱼ, then the user cannot add ℰⱼ → ℰᵢ.

By using experts’ constraints, our proposed approach can compose a trace that not only respects the existing constraints observed in the event log but also respects any additional constraints as requested by the experts.

In the next section, the evaluation of the proposed approach is described.

5. **EVALUATION**

In this paper, we conduct two evaluations: experimental evaluation and statistical analysis. First, we describe the dataset used and its specifications. Next, we show the results of experimental evaluation. Finally, the results of the statistical analysis are presented. We compare the results of our work with a state of the art related work.

5.1 **DATASET DESCRIPTION**

We evaluate our proposed method on the BPIC’15 dataset [47]. The Business Process Intelligence Challenge (BPIC) is an annual competition focused on applying process mining-related techniques on a real-life dataset. In this competition, different tools and methods can be used to extract better insight from process data [29, 48, 49]. The BPIC’15 dataset includes the event logs of building permit applications for 5 Dutch municipalities. The data was provided in the standard XES format for each municipality. Each record in the dataset describes an activity performed for a building permit application, and contains information fields such as case ID, employee ID, Resource, and a timestamp, among others. An overview of these fields can be found in [50]. In Table 3, we have shown the overall specification of the BPIC’15 dataset. The dataset has 262,628 records and includes several process types. The top 10 most used process types are as follows:

- P1: Bouw (a new building permit)
- P2: Kapvergunning (cutting a tree permit)
- P3: Milieu vergunning (environment permit)
- P4: Sloop (demolishing permit)
- P5: Brandveilig gebruik vergunning (fire safety in building)
- P6: Milieu neutraal wijziging (environment neutral change)
- P7: Monument
- P8: Handelen in strijd met regels RO (acting contrary to regulations RO)
- P9: Aanleg: uitvoeren werk of werkzaamheid (construction: execute work or activity)
- P10: Inrit/Uitweg (entrance/way)
There are also five process types that are not used in every and each of the municipalities. These types include:
- Brandveilig gebruik melding
- Flora en Fauna
- Gebiedsbescherming
- Milieu melding
- Roerende zaken

For example, the “Flora en Fauna” process has just been used in municipality C. In Figure 7, the average completion time (execution time) of different process types (P1 to P10) over 5 municipalities is shown. As seen, municipalities act very differently in executing these process types. For example, municipality A (abbreviated as MunA) has a shorter execution time for process P10 (Inrit/Uitweg), while at the same time it has one of the worst execution times for process P3 (Milieu vergunning). This can be due to several reasons, one of which is related to the way municipalities execute the same process. Figure 7 shows that each organization (municipality) can execute a similar process with different efficiencies. Therefore, it would be ideal to identify parts of a larger process that are executed most efficiently in certain organizations and compose them in order to find a desirable trace.

Among all of the process types, the new building permit (Bouw) process is the most widely used in all municipalities and so, this process type was analyzed. There are several papers that analyzed this process from different perspectives [28, 48, 51]. Likewise, in this paper, we will analyze the new building permit process throughout our experiments.

5.2 EXPERIMENTAL EVALUATION

5.2.1 PRE-PROCESSING DATASET

In this section, we experimentally evaluate the proposed approach. As an initial step, we performed some preliminary preprocessing on the input dataset. As noted in [50], the instances of the same process might have many different starting and ending
points. In our experiments, in total, we found 25 starting points and 158 ending points for the P1 process type across all municipalities. In [50], the authors have discussed that one of the reasons for such a high number of different starting and ending points can be due to the incompleteness of the log files. It is also possible that these differences could be due to different process variations. Since, the reason for the difference in the starting and ending point is not conclusive, we added a dummy starting and ending point for all cases so that they all start and finish with similar activities. We did this using the Disco process mining toolset [52].

Furthermore, the BPIC’15 challenge noted that the building permit process was changed in 2012 and therefore one might observe a concept drift within the business process logs due to the change in the actual process. In [50], the authors analyzed the process using the approach proposed in [53] and [54] for detecting concept drift and found that the changes do not occur at the same time for all five municipalities. The reason is that the municipalities had the freedom to decide on the start date for rolling out the changes in their processes. Therefore, we needed to make sure that we chose a subset of the event logs that were implementing the same process and with the same time alignment. For this reason, we picked out event data that occurred before 1 April 2012. In Table 4, a brief description of the building process related data is shown.

5.2.2 GENETIC ALGORITHM SETTINGS

We consider the population size to be 10% of the total pre-processed event logs. We select this value based on experiments in reference [38]. However, the analysis of the impact of initial population size on the quality of extracted results would be in the interest of future work. We construct half of them randomly and for the second half, we select the best execution traces of each set of logs (from each peer-organization) according to the utility function. We ran the GA process for 50 iterations. For the selection step, we chose the combination of the 4-tournament (in each step four individuals are selected) and the Elitism methods. We set the $t$ parameter in 4-tournament to 0.9 and consider Elitism of rate 2% (Top fittest 2% of each candidates are transferred to the next generation). Here, we consider crossover to be 50%. For crossover, we used both cut and splice crossover (with a probability of 70%) and two-point crossover (with
a probability of 30%). Mutation is also performed with a probability of 1%. We repeated the process 35 times and in all cases the algorithm converged in less than 50 epochs.
5.2.3 EVALUATION RESULTS

The first step after preprocessing the log files is extracting fragments from the event logs. As previously mentioned, we extract common fragments using the method proposed in [8]. We extracted 3,036 common fragments, with a minimum size of three activities and an average size of 8 activities. In Figure 8, we have shown the distribution of the ratio of common fragments across municipalities. As seen, more than 50 percent of the extracted fragments are common between two municipalities, 28 percent are common between three municipalities, 12 percent between four municipalities and about five percent of the fragments (119 fragments) are common between all municipalities.

In Figure 9, the number of extracted fragments per municipality is shown. Municipality E has the most number of fragments and municipality B has the least number of fragments. One factor that could have impacted the number of extracted fragments is the number of initial variants. Municipality E, which has the most number of fragments, also has the most number of variants (Table 4). Similarly, municipality B, which has the least number of variants (126), has the least number of fragments.

Figure 10 shows the distribution of the common fragments across different municipalities. For instance, it shows that municipality A has 667 common fragments with municipality B, 834 common fragments with municipality C, 721 common fragments
with municipality D and 847 common fragments with municipality E. Based on Table 4, municipality E has the highest number of variants (278). This might be the reason why all other municipalities have the highest number of commonalities with this municipality. The interesting point about municipalities A and C is that they have the least average execution time per case (Table 4). This might be related to the way that they execute the process. Also, municipality B has the highest number of common fragments with municipalities E and D. Municipalities B and D have the worst execution time as seen in Table 4. So, again it can be inferred that municipalities B and D use some common way of executing the process that leads to an increased execution time.

In Figures 11-15, we compare the average execution time of the extracted common fragments among all municipalities with the average execution time of the same fragments in other municipalities. The execution time is reported in minutes. The goal of these experiments is to analyze and show the difference in the execution time of the same fragment in different municipalities. The results of the experiments show that the same fragments can have very different execution times depending on which organization is executing them.

In Figure 11, the comparison of the average execution time (called AVG) of the extracted common fragments (among all municipalities) with the execution time of the same fragments in municipality A is shown. Based on this figure, it is interesting to see that there are cases where municipality A is executing some of the process fragments much better than the average, which is highlighted with green circles. There are also four cases where municipality A has a worse performance compared to the average highlighted with a red circle. We zoom into a part of Figure 11 to better show the differences for the sake of our discussion. In the zoomed pan, there are two fragments that have a much different execution time in municipality A compared to the average. From the left, there is a fragment that has an execution time of 38,995 minutes (about 27 days) in municipality A and an average execution time of 66,249 minutes (about 46 days) across all municipalities. Here, municipality A executes this fragment about 40% faster than the average of other municipalities. Also, there is another fragment that has an average execution time of 35,279 minutes (about 24 days) in municipality A but has an average execution time of 73,307.8 minutes (about 50 days) across all municipalities. This means that this fragment is executed in municipality A with less than half of the execution time of the other municipalities. There are also several other fragments that are executed in municipality A faster than the average of other municipalities. Therefore for such fragments are sorted by their execution time in descending order).

1 - Please note that the x-axis shows different fragments, which are not related to each other. The line chart was chosen to avoid clutter. Note should be taken that fragments are not necessarily related to each other. Adjacency and connection of points do not carry semantics (the fragments are sorted by their execution time in descending order).
cases, a good idea would be to use the execution and implementation model of municipality A in other organizations when the fragment has a lower execution time in municipality A and likewise to use the more desirable solutions from other municipalities within municipality A when they have a lower execution time compared to municipality A.

In Figure 12, municipality B is analyzed. In this diagram, there are at least three fragments whose execution time is much higher than the average across other municipalities. However, there are also fragments that are better than the average of the other municipalities. Figure 13 compares the average execution time of common fragments in municipality C. The diagram of municipality C has the highest conformity with the average execution times. In Figure 14, we can see the most difference with the average execution time. In the zoomed-in pan, we can see two fragments that have a significant difference in their execution time in municipality D compared to the average execution time in other organizations. The first one from the left side has an execution time of 206,600 minutes (about 143 days) in municipality D and an average of 66,249 minutes (about 46 days) in other municipalities and the second one has an execution time of 206,643 minutes (about 143 days) in municipality D and an average of 73,307 minutes (about 51 days) in other municipalities. Obviously, municipality D executes some fragments much slower than the other municipalities. Figure 15 also represents the difference between the execution time of fragments in municipality E and the average of other organizations. To sum up, these graphs confirm our hypothesis that each organization executes some parts of the process (fragments) in a more suitable way compared to the others. So, by detecting and extracting the best sub-processes from different organizations, it is possible to compose them to achieve a more efficient process execution model.

5.3 STATISTICAL ANALYSIS

In this section, we evaluate the execution time of the generated traces. In this regard, we calculate the minimum and average execution time of traces in different municipalities. In addition, we perform a statistical paired t-test over the generated desirable traces and existing traces in the dataset.

5.3.1 FINDINGS

In Figures 16-18, we have analyzed the execution time of the composed desirable trace (abbreviated as C.O.T) with the average and best execution time of each municipality. The time is calculated based on minutes. In Figure 16, the average execution time for each municipality is shown. In this analysis, we calculated the average time of each activity based on the model proposed in [45], and then calculated the execution time of each trace by summing the average execution time of its constituent activities using Equation 3 (global sum-of-average execution time). As seen in Figure 16, although C.O.T is higher than best execution time of each municipality, there is a significant improvement in the execution time when C.O.T is compared to the average execution time of the comparable process in the municipalities. As mentioned in the related works section, the authors in [28] use a mapping table to select the minimum execution time among several event logs and return it as best practice. The minimum average execution time belongs to MunC, which is about three times more than C.O.T. So, the generated traces have much better execution time in comparison to [28]. The reason is that in [28], the authors do not consider common sub-processes and only investigate the whole process.

However, the reason why C.O.T is higher than the best observed execution is that there are various traces in the event logs, which have different number of activities. For example, there exists a trace with 4 activities and a trace with 123 activities. The average number of activities per trace in MunA, MunB, MunC, MunD and MunE is
38, 49, 37, 43 and 47 activities, respectively. For this reason, we compare C.O.T with the average execution time of the traces.

In Figure 17, similar to Figure 16, we calculate the time of each trace based on the average time of each constituent activity. The difference is that here we calculate the average time of each activity locally within the event log of each municipality (local sum-of-average execution time) and not based on [45]. The difference with global sum-of-average execution time is that in this case, we calculate the average time of each activity locally within the event log of each municipality, not in event logs of all municipalities. Here, again we see that a considerable improvement can be achieved by using our proposed approach compared to the average execution time of each municipality. As seen, C.O.T is better than [28] (minimum average execution time belongs to MunC).
In Figure 18, the average execution time of each municipality is shown. For each municipality, we calculate the average time by dividing the sum of the duration time of the traces in the event logs of that municipality by the total occurrence frequency of the traces. We calculate the duration time by subtracting the completion time of the first event in the trace from the last event in the trace. We call this value the average of the real execution time. Again the execution time of the trace generated by our proposed method is superior to the average execution time of the traces that exist in each of the municipalities. So, here too, the result is better than [28]. In Figure 18, in addition to the average and best execution times, we have also shown the standard deviation. The high value of standard deviations shows that there is a very high variance in how traces are executed within the organizations.

These results show that our proposed approach is able to compose traces that have a lower execution time compared to the average execution time of traces in the event log while considering the existing constraints of the process models, extracted from the event logs.

In order to complete our statistical analysis, we applied the paired t-test analysis using IBM SPSS. In Figure 19, we show the paired t-test between real execution time of traces and the AVG Global Sum-of execution time in five municipalities. As seen, the p-value (Sig 2-tail) is greater than 0.05. Therefore, it can be concluded that there is no statistically significant difference between the two variables. In Figure 21, the paired t-test between real execution time of traces and the AVG Local Sum-of execution time is calculated. Here, the p-value is 0.092 and is greater than 0.05, too. So, in a similar vein, it is concluded that there is no statistically significant difference between real execution time of traces and the AVG Local Sum-of execution time. These results show the validity of the employed methods for estimating the execution time by showing that the time estimation methods that we used based on global sum-of-average and estimated local sum-of-average coincide with the real execution time of the traces.

6. TOOL SUPPORT

For implementing our proposed approach, we have extended the toolset, which we had previously developed for generating process variants, called PVG [55]. The PVG toolset can create a collection of process variants randomly according to a probabilistic distribution and based on a user-defined variation rate parameter. This tool has been im-
implemented within the PLG (Process Log Generator) toolset [56]. PLG generates random process models using context-free grammars by employing basic workflow patterns. Moreover, the user can select from three probability distribution functions: Uniform, Gaussian and Beta, which will be used for generating the number of branches for AND/XOR split-join patterns. After generating a random process model, PLG is capable of generating its execution logs by traversing the generated process graph.

A screenshot of the process variant generator tool is shown in Figure 21. Figure 21-a shows the loaded event data. The loaded datasets are shown in the left side of Figure 21-a, and the user can get more information about each dataset by clicking on the dataset name. Here, the details such as the number of events, number of activities, starting and ending activities, starting and ending date of the selected dataset is shown. This view helps the user get more insight about the data. By clicking on the “Extract Fragments” menu, the user can extract and export the extracted fragments based on the method in [8]. Figure 21-b shows the initial setting for running the proposed algorithms. This setting can be set by clicking on the “Extract Optimal Trace” menu. Users can set the parameters such as the amount of initial populations, the tournament parameters, type of crossover function and number of iterations.

In Figure 22, a view of the output is shown. Each output trace can contain activities or fragments. The activities are specified with blue color (larger boxes) and the fragments with purple color (smaller boxes). Also, the starting and ending elements are shown with a circle (green circle for start and red circle for end). By right-clicking on each element box, users can select the “More details” menu and see more details about that element. Several information about each element can be shown. Here, we represent the execution time of elements in each organization, separately. In this way, users can easily assess the performance of each element in each organization. For fragment element type, users can also see the contained activities. In Figure 22, there is a fragment “F70” which contains three activities: “14_VRIJ_010\complete”, ”3_GBH_005\complete” and “05_EIND_010\complete”.

Moreover, for the better detection of bottlenecks, the tool highlights the border of the elements that have the highest execution time with red. The intensity of the red color of the detected bottleneck element depends on the amount of the difference of its execution time with the execution time of the other elements. By right clicking and selecting “More details”, the details of the bottlenecks can be seen. Users can easily see
how the other peer-organizations perform this element. If there is an organization that executes the element much faster, then it would be wise to benefit from their experience to optimize the bottleneck element. This implemented tool fully supports the desirable trace generation technique proposed in this paper.

7. DISCUSSION

In this section, we have discussed the proposed approach from different points of view. In Section 7.1, we highlight the behavior of the proposed approach in dealing with noise. In Section 7.2, we show that the important factors for execution time optimization and discuss the factors that are used in this paper. In Section 7.3, we highlight the challenges of calculating activity duration time. Finally, in Section 7.4, we investigate several extracted fragments and generated traces for the BPIC’15 dataset.

7.1 EXPOSURE TO NOISE

In the real world, while analyzing business processes using event logs, one might encounter noise in the event logs. Having suitable solutions for handling noise can increase the applicability of a proposed approach. For instance, there might be some bugs in an information system that can lead to incorrect event logs, which can in turn lead to causal relationships that are not allowed. At the same time some causal relations might not be derived from the data, while in reality they are feasible/possible to be executed. We would like to highlight that this matter can be seen as a limitation of our proposed method. In this paper, since we extract the co-occurrence and ordinal constraints based on order and occurrence of observed events, we cannot detect noise. For instance, if two activities are always observed together in all traces, the co-occurrence relation will be extracted. But if among 100 traces, this was observed in 99 of the traces, then we do not extract co-occurrence constraints for it where in fact that one instance may have been noise. However, on a positive note, our proposed approach does not generate invalid traces and is only restricted by the fact that it might reduce the possible space for generating optimal traces. Also, we would like to lightly mention that some other work in the literature such as the Alpha algorithm and some of its variations also make a similar assumption to ours and assume that the log files do not contain noise. As future work, we are interested to extend our work to consider noise.

Also, there might be some relations that are not observed in the event logs. For example, there might be some unwritten rules in the organizations that are not implemented in the information system, while the employees enforce them. This often happens in organizations because of changes or incorrect implementation/usage of the information system. The proposed approach in this paper is limited to existing event logs so unwritten rules cannot be detected automatically. But it should be noted that the
The proposed approach has the capability for imposing external constraints. In other words, a business analyst can impose any optional constraint (which is in the format of ordinal/co-occurrence constraints). For example, if there is a priority relation between two activities, which have not been observed in the event logs, it can be easily imposed in the format of ordinal constraints or a business analyst may specify that two activities that have never appeared with each other can exist together in a trace. Likewise, she can specify that two activities that always appeared with each other can appear alone in some traces. By defining such extra constraints, the business analyst can perform additional exploratory analysis on the process logs and gain more insight for improving the business process.

7.2 IMPORTANT FACTORS FOR EXECUTION TIME OPTIMIZATION

In each organization, there are several factors that can have impact on the execution time of business processes such as demographic parameters, organization size, available resource and others. For example, an organization that has a higher number of employees might execute the process faster. Alternatively, the employees of some organization might work much more efficiently than other organizations (the qualities of employees are different) and hence produce different execution times. In a more complicated case, one employee or organization might exhibit different performances in different days. In this case, modeling the efficiency of employees is not trivial. As another example, organizations’ size can impact the execution time of business processes. Small organizations are often more agile and have less official hierarchy. In automated process improvement, considering all factors together can be important to increase the degree of automation.

However, in this paper, we only analyze the differences in execution order and activity selection for the same tasks in various peer-organizations. We propose that the differences in execution order and activity selection can have noticeable impact on execution time. We also mention in the paper that the existence of a business analyst alongside the proposed approach can lead to much more realistic results. As a part of our future work, we are interested to identify other effective parameters that can influence execution time and incorporate them into our proposed framework. To this end, we will require a comprehensive dataset which would include the complete execution information of a family of a business processes in several peer-organizations.

7.3 DURATION TIME OF THE FIRST/LAST ACTIVITY

The more accurate information is logged, the more value is added. Specifically, in performance analysis, having information such as starting time, ending time, costs, resources, case types and others can help more precise recommendations. In the case of the “time” attribute, most of the existing datasets only include starting or ending time of the activities. In such cases, we have to consider the timestamp as starting time or ending time. On the one hand, if the timestamp is stored as ending time of activities (similar as BPIC’15 dataset), then the duration time of each activity can be obtained by the difference of the timestamp of that activity with timestamp of the previous one. In this case, the duration of the first activity would be equal to zero. On the other hand, if we consider the timestamp as the starting time then the duration of each activity is equal to the difference of the timestamp of the next activity with the timestamp of that activity. In this case, the duration of the last activity will be zero.

However, in cases when two separate timestamps exist for starting and ending time of each activity, we can exactly calculate the duration of all activities and in this case, the duration time of each activity is calculated precisely. In this paper, since we just had one timestamp (completion time) in the dataset, the duration time of the first activity is ignored and subsequently is not considered in the calculations.
7.4 DISCUSSIONS ON RESULTS

In this section, we have two main objectives:

1. We intend to show and investigate some of the extracted fragments of BPIC’15. We show various fragments with different behavior, for each of which, we compare their execution patterns in all municipalities.

2. We show some of the composed traces and discuss them. For each trace, its constituent elements (activities or fragments) are shown. For each activity, a brief description is given in the appendix. Also, we compare the execution of each fragment across different municipalities.

7.4.1 THE EXTRACTED FRAGMENTS

In Table 5, we have shown some of the extracted fragments from the BPIC’15 dataset. For each fragment, there are five columns, which explain the details of executions of that fragment in each municipality. For the sake of simplicity, we have shown each activity with an English letter. The mapping and details of each activity (including the short description of the activity which is extracted from the BPIC’15 dataset) is shown in Table 6 in the Appendix. For each fragment in Table 5, the sequence of activities (the Fragment column) and the average execution time (the Time column) is shown. For example, the F94 fragment exists in municipalities MunA, MunB, MunC, and MunD. In municipality A, the F94 fragment is executed in one way and includes the sequence “A, E, D, B” with execution time of 6,988 minutes (4 days, 20 hours, 28 minutes). However, this fragment is executed in municipality D in parallel (activities...
E and D are executed in parallel) with average execution time of 3,277 minutes (2 days, 6 hours, 37 minutes).

Let us now investigate each fragment separately. We first describe some fragments that have a zero execution time. The F148 fragment consists of W, C, F, G and H. This fragments is executed in MunA, MunB and MunE in different ways. However, the average execution time of this fragment in all municipalities is equal to zero. Activities W, C and F refer to “treat subcases completeness”, “register date of publishing received request” and “subcases completeness completed”, respectively. So, it seems that this fragment is about completeness of process. Here, activity C is executed in parallel with G in MunB and also executed in parallel with F and G in MunE. However, it seems that the sequential or parallel execution of activities does not have an effect on the total execution of this fragment. Similarly, the execution time of the F192 fragment is almost near zero in all municipalities where it appears (MunA, MunB, MunC and MunE). According to Table 6, this fragment is about refusing a request and similar to F148, different executions do not have effect on the total execution time of the fragment. In the F539 fragment, a different scenario can be seen. Fragment F539 has an execution time of 3,915 minutes (equal to 2d, 17h, 15m) in MunB, while its execution time is zero in both MunD and MunE. In this case, the execution pattern of F539 in MunB is different from MunD and MunE. Looking at “requestComplete” field of the related cases, we found that the value for this field was FALSE in MunB, while it has a value of TRUE for both MunD and MunE. Based on existing fields in the database, it is not clear why the “requestComplete” is FALSE for MunB.

In the next case, we review Fragment F94, which contains four activities A, E, D and B. MunA and MunC have the highest execution times and considerable difference with MunB and MunD. The interesting point is that MunA and MunC have the same execution pattern (A, E, D, B) whereas MunB and MunD have similar patterns to each other. In this case, MunB and MunD execute activities E and D in parallel. So, as a recommendation to MunA and MunC, it can inferred that E and D can be parallelized and this parallelization can lead to considerable decrease in execution time.

Fragment F45 contains 6 activities E, J, K, X, M and S. F45 is executed in MunA, MunB and MunD in parallel and have almost similar average execution times in these municipalities. However, in MunC there is a different execution pattern and has the highest average execution time. Looking at the description of the intermediate activities of F45 in Table 6, it seems that these activities can be run in parallel. In this case, the municipalities that execute this fragment in parallel had the best average execution time.

Fragment F211 contains 6 activities M, N, O, P, Q, and R. Based on the observed events, all municipalities except MunA execute this fragment in parallel. In MunA, F211 executes it as “M,N,O,P,Q,R” with an execution time of 9,979 minutes (6d, 22h, 19m). In MunB and MunC, activities N and O are executed in parallel. In MunE, a higher number of activities is executed in parallel. In this example, the average execution time of F211 in MunA, is much more than other municipalities, which execute F211 in parallel.

Fragment F236 has the exact execution pattern of “S,C,T,U,V,W” in MunA and MunD and also the highest execution time. MunB also executes the activities sequentially as “S,C,V,U,T,W”. However, its execution time is much less than MunA and MunD. Looking at the difference of execution patterns in MunB compared to MunA and MunD, the reason can be explained. In MunB, after activities S and C, activity V (terminate on request) is executed, while in MunA and MunD, after activities S and C, activity T (WAW permit aspect) is executed. In the other words, in MunB, after activities S and C, the request terminates, while in MunA and MunD, the termination occurs only after running activities S, C, T and U. Although the semantics of this termination
cannot be extracted from this event log, however it can be understood that in sequential execution, the execution pattern of MunB is better than that of the patterns from MunA and MunD. Within these municipalities, it seem that MunC executes this fragment in parallel and has the least execution time. Looking at different patterns, which have been executed in the municipalities, it seems that four activities C, U, T and V can be executed in parallel and business analyst can decide to parallelize these activities to improve the total execution time.

Fragment F364 contains activities $A^+$, $B^+$, $M$, $N$, $O$, $Q$, $Y$ and $Z$. In MunA and MunB, this fragments is executed in parallel (activities $M$ and $O$ are executed in parallel) and have similar average execution times. In MunE, activities $O$ and $N$ are executed in parallel and have average execution times close to MunA and MunB. However, in MunD, there is a sequential execution pattern of “$Y,Z,A^+,M,O,B^+,N,Q$”. So, in this case, it seems that activities O, M and N can be executed in parallel and the parallelism can considerably decrease execution time.

In the last example, fragment F136 contains activities $E$, $I^+$, $S$ and X and is common between three municipalities. In MunA and MunE, two different sequential patterns of “$X,E,S,I^+$” and “$X,S,E,I^+$” are executed, respectively. In MunD, this fragment is executed in parallel (activities S and E are executed in parallel). Also, the description of activities S (send procedure confirmation) and E (enter send date procedure confirmation) confirm that these activities can be executed in parallel. However, unlike previous cases, in this case, the parallel execution of activities leads to an increased time. One reason might be the lack of sufficient resources in MunD for running these activities, which leads to an increase in the total execution time.

In Figure 23, using the ProM tool [54], we extract a BPMN of a morphological fragment (F129). As seen, the average execution time of F129 in each municipality is calculated. Municipalities B, C and E execute this fragment in a similar pattern. In these
three municipalities, the fragment starts with activity O, then, activities N and Q are execute in parallel and finally, the fragment ends with activities RQ. Municipality A also executes the fragment in a similar way with one difference that activity N is parallel with activities Q and R. As seen, the average execution time of F129 in these four municipalities are close to each other (MunA has the least average execution time). However, municipality D executes all activities sequentially and has the most average execution time. Fragment F129 is a good example of a best practice which municipality D can learn from other municipalities. There are several other fragments that have similar behaviour to the described fragments of Table 5 that have not been included here due to space constraints. Analyzing these fragments by a business analyst, in itself, can lead to a lot of added value for peer-organizations.

The main purpose of this discussion is to show that organizations can improve their process execution time using the common sub-process patterns in similar peer-organizations. Also, the analysis of execution time of morphological fragments in BPIC’15, showed that there are noisy fragments that have an execution time of zero or near zero in some municipalities. It seems that these fragments are noise or at least their execution time was not recorded correctly.

### 7.4.2 SAMPLE COMPOSED TRACES

Now, in this section, we provide some samples of final traces generated using the proposed method. As previously mentioned, there are some extracted fragments that seem to be noise according to their execution time (which is zero or close to zero). Since, we did not have access to the municipalities’ analysts, we consider them as noise. However, if we consider them, then the GA will generate the traces, which contains these fragments such as traces $T_1$ and $T_2$.


The trace $T_1$ contains fragment F192 as described in Table 5. Trace $T_2$ contains fragments F247 and F16. Fragment F247 exists in MunB, MunC, MunD and MunE. However, its execution time in MunB, MunC and MunE is 1,636, 10,256 and 38,859 minutes, respectively, while its execution time in MunD is zero. We did not consider these fragments in generating the desirable trace. Without considering the noisy fragments, the best desirable trace, which contains morphological fragments, is trace $T_3$.


Trace $T_3$ includes three fragments F45, F381 and F16 and has an execution time of 17,852 minutes, which is much less than the minimum average execution time among 5 municipalities. Fragment F45 has been previously described in Table 5. This fragment has the best execution time in the parallel execution pattern (in MunD). The Fragment F381 contains activities A, C, E, M, O, P, Q, Y, Z and executed in MunA, MunC, MunD and MunE. MunA and MunE have executed a similar pattern “C, E, Q, Y, Z, A, O, M, P” and their execution time is 4,391 and 4,856 minutes, respectively. MunC also executed the activities sequentially in the form of
execution time, it is possible to extend our work to consider additional optimization logs.

while respecting the extracted ordinal and co-occurrence constraints from the event ties/fragments beginning from the start node and terminating at a specific end node ing a Genetic algorithm. Using GA, we find the best feasible composition of activi-

ties/fragments and calculate their utility. Afterward, we generate desirable traces us-

rithms extract their common fragments. Next, we extract the relations among activi-
tion. We take a collection of event logs as input and using existing fragmentation algo-

logs of variants of the same configurable business process according to a utility func-

In this paper, we have proposed an approach for composing efficient traces from event logs of variants of the same configurable business process according to a utility function. We take a collection of event logs as input and using existing fragmentation algo-

To the descriptions in Table 6, activities M (generating decision environmental permit) and O (register date environmental permit decision) can be executed in parallel. MunD executed the pattern “C,E,Q,Y,O,Z,A+,M,P” in 8,873 minutes. So, the minimum execution times belong to MunA and MunE. Fragment F16 contains four activities N, R, O+, P+ and is executed in MunA, MunC and MunD. The activities R and O+ is executed in parallel in MunA and MunC and these two municipalities have much less average execution time for F16 compared to MunD (10,659 minutes), which executes F16 sequentially. Based on the descriptions of activities in Table 6, activities R (generate publication document decision environmental permit) and O+ (enter date publication decision environmental permit) can be executed in parallel.

Now, we show a simple composed trace and investigate it from a composition point of view. Trace \( T_4 \) contains three fragments F77, F113 and F9.

\[
\]

This trace has two main parts, which are highlighted with red (starts from F77 to F113) and blue (starts from V+ to P+) colors. Actually, this trace is a result of combination of two different traces. We show the parent traces below:

\[
\]

\[
\]

Trace \( T_4 \) is the result of the combination of \( T_5 \) and \( T_6 \). The first part of \( T_4 \) comes from \( T_5 \) (highlighted with red color) and the second part comes from \( T_6 \) (highlighted with blue color). In combination (crossover) of \( T_5 \) and \( T_6 \) several constraints have been checked. These rules have been extracted from existing relations among constituent elements of \( T_5 \) and \( T_6 \) across all traces in the dataset. For example, since in \( T_6 \) we see that activity V+ has appeared after element F113, so in the ordinal relation of F113, element V+ will exist. Therefore, the first part of \( T_5 \) can be composed with the second part of \( T_6 \) from the location of F113. Of course, it should be noted that similar to F113, for all constituent elements of \( T_5 \) and \( T_6 \) both ordinal and co-occurrence constraints have been checked and it is only when all constraints are satisfied that the composition can occur.

In this section, we showed a few generated traces which use extracted fragments from the previous section. By studying the description of activities in Table 6 (Appendix) for the extracted traces, it was shown that the traces can be considered to be desirable.

8. CONCLUSION AND FUTURE WORKS

In this paper, we have proposed an approach for composing efficient traces from event logs of variants of the same configurable business process according to a utility function. We take a collection of event logs as input and using existing fragmentation algo-

While the focus of this paper has been on generating traces that have the lowest execution time, it is possible to extend our work to consider additional optimization
parameters. This would not require major changes to our proposed approach and would only require an update to the GA utility function. For instance, if the objective is to generate traces that have the smallest execution time and also the smallest number of elements, it would be possible to devise a utility function that considers both at the same time. As a future work, we intend to focus on this issue by looking at cases where there are more than one measure for optimization. We will be looking specifically at cases where trade-off decisions need to be made and therefore the decision lies between the concurrent optimization of multiple competing objectives. For instance, if most cost-efficient and fastest trace is desirable, the trade-off between cost and time needs to be studied. Questions such as ‘how much extra cost can be tolerated for decreasing execution time to a certain percentage’ need to be studied and embodied within the utility function. Also, another solution for making trade-off for multi-objective optimization problem is Pareto front [57]. In a Pareto front, all members are mutually non-dominating and a member dominates another one if and only if for all quality dimensions, it has equal or better value, and for one dimension has strictly better value [58]. Buijis has used the concept of Pareto front in his research for making trade-off between different quality dimensions in process discovery (precision, replay fitness, simplicity and generalization) [58]. Pareto front can help in solving multi-optimization problems in our future work.

An additional area for future work that we are interested in exploring is the adoption of process time patterns. As originally introduced by Lanz et al. [59, 60], process time patterns provide support for the specification and characterization of time constraints in business process and workflows. For instance, time lags between activities or sub-process duration are some of such temporal patterns. We are interested in understanding whether it would be possible to mine such time patterns from event logs and then enforce them when desirable traces are being composed. The importance of this would be that it will ensure that time critical aspects of a process will be respected in the composed process.

REFERENCES


### APPENDIX.

Table 6. Description of activities used in Table 5.

<table>
<thead>
<tr>
<th>Activity</th>
<th>Label</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td>A</td>
<td>automatic added starting node</td>
</tr>
<tr>
<td>01_HOOFD_010\complete</td>
<td>B</td>
<td>register submission date request</td>
</tr>
<tr>
<td>01_HOOFD_100\complete</td>
<td>C</td>
<td>register date of publishing received request</td>
</tr>
<tr>
<td>01_HOOFD_030_2\complete</td>
<td>D</td>
<td>enter send date acknowledgement</td>
</tr>
<tr>
<td>01_HOOFD_065_2\complete</td>
<td>E</td>
<td>enter send date procedure confirmation</td>
</tr>
<tr>
<td>01_HOOFD_120\complete</td>
<td>F</td>
<td>subcases completeness completed</td>
</tr>
<tr>
<td>01_HOOFD_180\complete</td>
<td>G</td>
<td>procedure change / activities regular procedure</td>
</tr>
<tr>
<td>01_HOOFD_101\complete</td>
<td>H</td>
<td>registration date publication</td>
</tr>
<tr>
<td>09_AWB45_005\complete</td>
<td>I</td>
<td>request complete</td>
</tr>
<tr>
<td>09_AH_I_010\complete</td>
<td>J</td>
<td>article 34 WABO applies</td>
</tr>
<tr>
<td>01_HOOFD_015\complete</td>
<td>K</td>
<td>phase application received</td>
</tr>
<tr>
<td>01_HOOFD_020\complete</td>
<td>L</td>
<td>send confirmation receipt</td>
</tr>
<tr>
<td>01_HOOFD_490_2\complete</td>
<td>M</td>
<td>generating decision environmental permit / decision date prior to decision</td>
</tr>
<tr>
<td>01_HOOFD_510_2\complete</td>
<td>N</td>
<td>enter send date decision environmental permit</td>
</tr>
<tr>
<td>01_HOOFD_490_3\complete</td>
<td>O</td>
<td>register date environmental permit decision</td>
</tr>
<tr>
<td>01_HOOFD_500\complete</td>
<td>P</td>
<td>register objection and appeal periods</td>
</tr>
<tr>
<td>01_HOOFD_510_1\complete</td>
<td>Q</td>
<td>transcript decision environmental permit to stakeholders</td>
</tr>
<tr>
<td>01_HOOFD_520\complete</td>
<td>R</td>
<td>generate publication document decision environmental permit</td>
</tr>
<tr>
<td>01_HOOFD_065_1\complete</td>
<td>S</td>
<td>send procedure confirmation</td>
</tr>
<tr>
<td>01_HOOFD_130\complete</td>
<td>T</td>
<td>WAW permit aspect</td>
</tr>
<tr>
<td>01_HOOFD_050\complete</td>
<td>U</td>
<td>inform BAG administrator / applicant is stakeholder</td>
</tr>
<tr>
<td>05_END_010\complete</td>
<td>V</td>
<td>terminate on request</td>
</tr>
<tr>
<td>01_HOOFD_110\complete</td>
<td>W</td>
<td>treat subcases completeness</td>
</tr>
<tr>
<td>01_HOOFD_060\complete</td>
<td>X</td>
<td>regular procedure without MER</td>
</tr>
<tr>
<td>01_HOOFD_470\complete</td>
<td>Y</td>
<td>suspension ground applicable</td>
</tr>
<tr>
<td>01_HOOFD_480\complete</td>
<td>Z</td>
<td>by law</td>
</tr>
<tr>
<td>01_HOOFD_490_1\complete</td>
<td>A</td>
<td>creating environmental permit decision</td>
</tr>
<tr>
<td>01_HOOFD_495\complete</td>
<td>B</td>
<td>phase decision taken</td>
</tr>
<tr>
<td>01_HOOFD_370\complete</td>
<td>C</td>
<td>assessment of content completed</td>
</tr>
<tr>
<td>01_HOOFD_375\complete</td>
<td>D</td>
<td>phase advice known</td>
</tr>
<tr>
<td>01_HOOFD_380\complete</td>
<td>E</td>
<td>grounds for refusal</td>
</tr>
<tr>
<td>01_HOOFD_390\complete</td>
<td>F</td>
<td>updated plan after review</td>
</tr>
<tr>
<td>01_HOOFD_400\complete</td>
<td>G</td>
<td>contacting applicant</td>
</tr>
<tr>
<td>01_HOOFD_190_1\complete</td>
<td>H</td>
<td>resume completeness subcases / regular procedure applies</td>
</tr>
<tr>
<td>14_VRIU_010\complete</td>
<td>I</td>
<td>no permit needed or only notification needed</td>
</tr>
<tr>
<td>01_HOOFD_030_1\complete</td>
<td>J</td>
<td>send confirmation receipt</td>
</tr>
<tr>
<td>01_HOOFD_040\complete</td>
<td>K</td>
<td>forward to the competent authority</td>
</tr>
<tr>
<td>01_HOOFD_190\complete</td>
<td>L</td>
<td>request complete</td>
</tr>
<tr>
<td>01_HOOFD_190_2\complete</td>
<td>M</td>
<td>enter send date procedure confirmation\subcases completeness completed</td>
</tr>
<tr>
<td>01_HOOFD_195\complete</td>
<td>N</td>
<td>phase application receptive</td>
</tr>
<tr>
<td>01_HOOFD_530\complete</td>
<td>O</td>
<td>enter date publication decision environmental permit</td>
</tr>
<tr>
<td>End</td>
<td>P</td>
<td>automatic added ending node</td>
</tr>
<tr>
<td>01_HOOFD_430\complete</td>
<td>Q</td>
<td>ask stakeholders views</td>
</tr>
<tr>
<td>03_GBH_005\complete</td>
<td>S</td>
<td>applicant is stakeholder</td>
</tr>
<tr>
<td>07_OPS_010\complete</td>
<td>T</td>
<td>suspend term</td>
</tr>
<tr>
<td>06_VD_010\complete</td>
<td>U</td>
<td>extend procedure term</td>
</tr>
<tr>
<td>01_HOOFD_200\complete</td>
<td>V</td>
<td>send letter in progress / request complete</td>
</tr>
<tr>
<td>15_NGV_010\complete</td>
<td>W</td>
<td>request further information</td>
</tr>
<tr>
<td>01_HOOFD_250\complete</td>
<td>X</td>
<td>treat subcases content</td>
</tr>
<tr>
<td>01_HOOFD_260\complete</td>
<td>Y</td>
<td>completed subcases content</td>
</tr>
<tr>
<td>01_HOOFD_330\complete</td>
<td>Z</td>
<td>procedure change / activities regular procedure</td>
</tr>
<tr>
<td>01_HOOFD_205\complete</td>
<td>A</td>
<td>phase application receptive / send letter in progress</td>
</tr>
<tr>
<td>01_HOOFD_420\complete</td>
<td>B</td>
<td>partly permit</td>
</tr>
<tr>
<td>01_BB_540\complete</td>
<td>C</td>
<td>objection lodged against decision</td>
</tr>
<tr>
<td>01_BB_770\complete</td>
<td>D</td>
<td>set phase: phase permitting irrevocable</td>
</tr>
<tr>
<td>01_HOOFD_810\complete</td>
<td>E</td>
<td>register deadline</td>
</tr>
<tr>
<td>01_HOOFD_270\complete</td>
<td>F</td>
<td>WAW permit aspect</td>
</tr>
<tr>
<td>01_HOOFD_540\complete</td>
<td>G</td>
<td>suspend term</td>
</tr>
<tr>
<td>01_HOOFD_350_1\complete</td>
<td>H</td>
<td>send procedure confirmation</td>
</tr>
<tr>
<td>01_HOOFD_350_2\complete</td>
<td>I</td>
<td>enter send date procedure confirmation</td>
</tr>
<tr>
<td>01_HOOFD_790\complete</td>
<td>J</td>
<td>register deadline / close case</td>
</tr>
<tr>
<td>01_HOOFD_600\complete</td>
<td>K</td>
<td>close case / TRUE</td>
</tr>
</tbody>
</table>