

Pattern-based Action Engine: Generating Process Management Actions Using Temporal Patterns of Process-centric Problems

Gyunam Park^{a,*}, Daniel Schuster^b, Wil M. P. van der Aalst^a

^a*Process and Data Science Group (PADS), RWTH Aachen University, Ahornstr. 55, Aachen, 52074, Nordrhein-Westfalen, Germany*

^b*Fraunhofer FIT, Schloss Birlinghoven, Konrad-Adenauer-Str., Sankt Augustin, 53757, Nordrhein-Westfalen, Germany*

Abstract

As business environments become more competitive, organizations strive to improve their business processes to reduce costs and increase quality and productivity. As process improvement traditionally embraces manual creative tasks that are time-consuming and labor-intensive, the need for automating it arises. Action-Oriented Process Mining (AOPM) aims to support automated process improvement by leveraging various process mining techniques. To that end, AOPM first monitors the presence of operational constraints, i.e., operational problems, in business processes, e.g., a high waiting time for patients to register. Next, it produces interim management actions designed to address these transient problems by analyzing the monitoring results. For instance, if an excessive waiting time persists for more than a week, the system might recommend dispatching additional resources for the upcoming week. Contrary to the mature process mining support for monitoring operational constraints, the action part is typically missing in today's process mining tools. In this work, we propose an action engine to support the automatic generation of actions. It analyzes temporal patterns of monitoring results and produces action plans that describe the execution of management actions. We have demonstrated a use case using the data of a Dutch finan-

*Corresponding author

Email addresses: gnpark@pads.rwth-aachen.de (Gyunam Park),
daniel.schuster@fit.fraunhofer.de (Daniel Schuster),
wvdaalst@pads.rwth-aachen.de (Wil M. P. van der Aalst)

cial institute to evaluate the feasibility of the proposed action engine and conducted experiments to evaluate its effectiveness.

Keywords: Process Mining, Action-Oriented Process Mining, Process Improvement, Temporal Pattern Mining, Action Planning

1. Introduction

Organizations have endeavored to improve their business processes using various approaches. Business process reengineering aims to systematically manage process improvements by making changes in people, processes, and technology. Hammer [1] provided a collection of principles for improving business processes derived from case studies of successful business processes. Business process redesign focuses on more neutral changes in terms of size and pace of changes. Reijers et al. [2] propose a framework for business process redesign and 29 best practices based on the framework. Six Sigma/Lean Management aims to focus on identifying process problems and then eliminating them. Whereas Six Sigma identifies process problems by monitoring deviations in a number of process performance measures [3], Lean Management identifies process problems by analyzing various “wastes” [4].

Such efforts to improve business processes commonly require manual activities that are time-consuming, expensive, and labor-intensive [5]. Beer-repoot et al. [6] enlist automating business process improvements as one of “the biggest business process management problems to solve before we die”. The goal is to incorporate process improvement as a part of daily business and incrementally improve processes with increasing degrees of autonomy.

Action-Oriented Process Mining (AOPM) aims to achieve automated and continuous process improvement by leveraging process mining techniques [7]. Process mining provides a wide range of techniques both for backward-looking (e.g., finding a bottleneck in a process) and forward-looking insights (e.g., predicting a bottleneck). Backward-looking process mining techniques include process discovery, conformance checking, and performance analysis, whereas forward-looking process mining techniques include deviation detection, prediction, and recommendation [8].

The framework for AOPM embodies two major functional components to automatically generate process management actions [9]. First, *constraint monitoring* analyzes event data to evaluate the presence of various *operational constraints* in business processes. An operational constraint in business

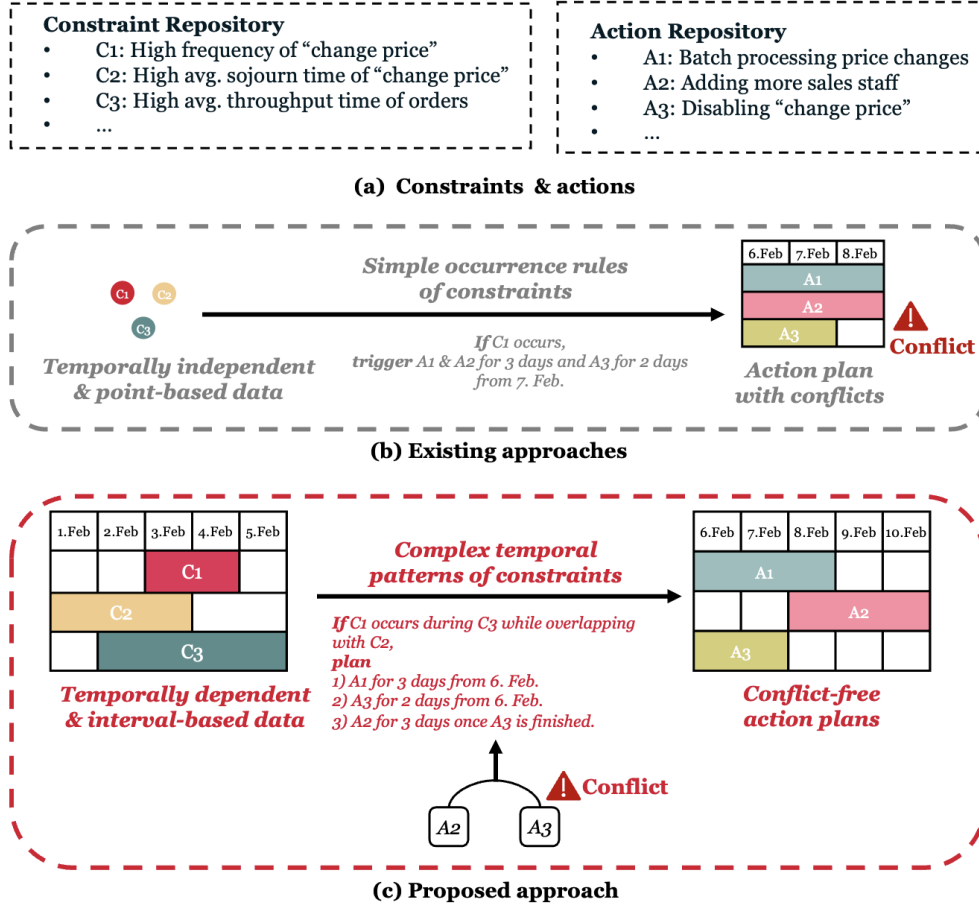


Figure 1: (a) Examples of operational constraints (i.e., operational problems) and actions in an O2C process, (b) Existing action engines analyzing operational constraints as temporally independent and point-based data and producing conflicting action plans, (c) Our proposed action engine analyzing operational constraints as temporally dependent and interval-based data and producing conflict-free action plans

processes represents an operational problem. For instance, an Order-To-Cash (O2C) process may have the operational constraints shown in Figure 1(a): $C1$, $C2$, and $C3$. The insights from backward-looking process mining techniques can be used to elicit such operational constraints to monitor, and various backward-looking and forward-looking monitoring techniques can be

used to track the (potential) presence of such operational constraints. Second, an *action engine* analyzes the monitoring results and generates necessary management actions. For instance, consider actions *A1*, *A2*, and *A3* depicted in Figure 1(a) to mitigate the risk caused by the presence of operational constraints.

In contrast to extensive support for constraint monitoring, an action engine is missing in most of today’s process mining tools. *Celonis Action Engine* is one of the few examples implementing action-oriented process mining [10]. Based on a trigger-action programming model [11], the action engine generates triggering signals by analyzing the event data and executes the actions corresponding to these signals to the source system. For instance, as shown in Figure 1(b), it analyzes the presence of operational constraint *C2* and, if so, triggers action *A1* and *A2* for 3 days. In [9], the cube-based action engine is proposed to use multi-dimensional queries to analyze the aggregated value, e.g., if *C1* occurs more than once a week, then trigger actions.

The existing approaches have two limitations. First, they consider operational constraints as temporally independent and point-based data, analyzing relatively simple patterns, e.g., the mere presence of operational constraints. However, in reality, operational constraints are 1) temporally dependent, e.g., *C1* is followed by *C2*, and 2) interval-based, e.g., *C1* is present from 1.Feb. to 2.Feb. By analyzing operational constraints as temporally dependent and interval-based data, we enhance an action engine to analyze the temporal pattern of operational constraints, as shown in Figure 1(c). Second, the existing approaches consider actions as being conflict-free. However, in reality, actions may have conflicts, i.e., some actions cannot be executed simultaneously. For instance, *A2* and *A3* may have conflicts, i.e., adding more staff to support price changes cannot be simultaneously executed with disabling the activity.

In this work, we propose a *pattern-based action engine* consisting of three phases. First, we design *action graphs* that are graphical notations to specify how to analyze temporal patterns of operational constraints and how to generate the corresponding actions, using, e.g., domain knowledge. Next, based on the action graphs, we generate actions by analyzing *constraint instances*, i.e., the record of constraint monitoring. Finally, we plan the generated actions by considering possible conflicts between the actions. We evaluate the feasibility of the proposed action engine using a real-life loan application process of a Dutch financial institute. Moreover, we conduct experiments to evaluate the scalability and performance of the action engine.

75 The remainder is organized as follows. We present related work in Sec-
76 tion 2. Next, we explain preliminaries in Section 3. Afterward, we present
77 the proposed action engine in Section 4. Section 5 presents the evaluation
78 of the proposed approach. Finally, Section 6 discusses the implication and
79 limitations of the proposed approach, and Section 7 concludes this paper.

80 2. Related Work

81 This section presents the related work in business process reengineering
82 & redesign, and action-oriented process mining. Furthermore, we introduce
83 several approaches to analyzing temporal patterns of interval-based data.
84 Finally, we present literature on process improvement actions to introduce
85 various available management actions that can incrementally improve busi-
86 ness processes.

87 2.1. Business Process Reengineering and Redesign

88 The drive to improve business processes is an ongoing pursuit [12]. *Busi-*
89 *ness process engineering*, an early systematic approach to this endeavor, aims
90 for transformational changes across people, processes, and technology [1]. It
91 aspires to radically restructure an organization’s existing processes, aiming
92 for significant improvements in performance, efficiency, and effectiveness.

93 In contrast, *business process redesign* places emphasis on more moderate
94 adjustments, with a focus on refining existing processes rather than complete
95 transformation [13]. Reijers et al. [2] offer a comprehensive framework for
96 business process redesign, highlighting six key elements to consider during
97 redesign efforts. Based on these elements, the authors share 29 best practices
98 for process redesign, derived from case studies and expert opinions. Fehrer
99 et al. [14] suggest a tool, Assisted Business Process Redesign (ABPR), to
100 streamline the redesign process. As a data-driven, iterative approach, ABPR
101 targets specific performance objectives. It delivers four tiers of recommen-
102 dations, with each tier increasing in domain and use case specificity. The
103 proposed reference architecture serves as a template for new instantiations,
104 addressing the current void in available tools.

105 2.2. Action-Oriented Process Mining

106 Recent developments suggest various methods for producing prescriptive
107 actions based on predicted performance and risk of process instances. Kubrak

et al. [15] suggest a comprehensive framework to categorize prescriptive process monitoring techniques. One of the core dimensions of this framework is the *objective*, which classifies techniques into two main categories: those focused on reducing negative outcomes, such as defect rates, and those aiming to optimize a specific performance metric, like cycle time. For example, the work of Conforti et al.[16] fits into the first category with its goal of minimizing risks in ongoing process instances. Conversely, the method by Bozorgi et al. [17] aligns with the second category, aiming to optimize the cycle time of processes. Another key dimension in the framework is *type of actions* prescribed to meet the objectives. This dimension identifies three primary types of actions: resource allocation, control flow adjustments, and alarm generation. For example, Conforti et al.[16] deploy techniques that generate actions concerning resource allocation. Their approach predicts the risk levels of running process instances and then optimizes resource allocations based on these predictions. Similarly, Weinzierl et al.[18] focus on control flow adjustments by recommending the next best activities based on key performance indicators. Lastly, Fahrenkrog-Petersen et al. [19] belong to the alarm generation category, as their approach aims to produce alarms for predicted problematic instances. Their method further employs a cost model to optimize these alarms, thereby capturing the trade-offs between different types of alarms.

Celonis Action Engine [10], a part of *Celonis Execution Management System (EMS)*, is a representative effort to achieve the goal of action-oriented process mining, i.e., turning diagnostics into actions. It analyzes event data to produce signals and execute necessary actions to source systems based on the signals. In [9], a systematic framework for action-oriented process mining is suggested to transform event data into process-centric diagnostics and the diagnostics into needed management actions instead of generating actions in an ad-hoc manner. Moreover, the authors propose a cube-based action engine to enable multi-dimensional queries to analyze the aggregated value and trigger the corresponding action.

Existing approaches in action-oriented process mining, however, consider operational constraints as temporally independent and point-based data, generating actions based on relatively simple occurrence rules (cf. Figure 1). Moreover, they do not consider possible conflicts between actions that are prevalent in reality when triggering the generated actions. This paper tackles this research gap by proposing a pattern-based action engine to analyze complex temporal patterns of operational constraints and produce conflict-free

actions by considering possible conflicts between the actions. Additionally, in contrast to existing prescriptive process monitoring techniques that focus on individual process instances, our proposed action engine operates at the process level, tackling operational constraints across multiple process instances.

2.3. Temporal Pattern Mining

Operational constraints within business processes often manifest as temporal patterns that indicate problematic situations requiring actions. Domain experts, knowledgeable in their specific processes, are typically responsible for defining these patterns. However, this task may be challenging given the complexity of operational constraints in business processes. One potential solution is the automation of temporal pattern discovery from the historical record of constraint monitoring, thereby facilitating the identification and characterization of these problematic situations. This is where temporal pattern mining, a relatively young research field, plays a role.

Temporal pattern mining aims to analyze temporal patterns in symbolic time intervals, also known as interval-based data. Villafane et al. [20] propose an initial approach to analyze containments of time intervals in multi-symbolic time interval series. Kam and Fu [21] use Allen’s temporal relations [22] to compose frequent temporal patterns, called *A1 patterns*. An *A1 pattern* represents the temporal relation between an existing temporal pattern and the next symbolic time interval. However, the temporal relations among the time intervals in A1 patterns are ambiguous since the temporal relations are defined only among the pairs of successive intervals [21]. Höppner [23] presents a nonambiguous representation of temporal patterns by a k^2 matrix representing all of the pair-wise relations within a k -sized Time Intervals Related Pattern (TIRP). Later, the representation was simplified to the sequence of symbolic time intervals and the conjunction of the pair-wise temporal relations among time intervals by removing redundant temporal relations, i.e., the inverse of Allen’s temporal relations [22].

Papapetrou et al. [24] propose two approaches for generating TIRP trees: Breath First Search (BFS)-based and Depth First Search (DFS)-based approaches. The former enumerates the node at each level before proceeding to the next level, whereas the latter enumerates each path up to the leaves as a greedy approach. Moreover, they present a hybrid approach that combines both approaches based on the Sequential Pattern Mining Algorithm (SPAM) [25] mining method. Moskovitch et al. [26] propose the KarmaLego

algorithm that efficiently generates candidate TIRPs by exploiting the transitivity property of temporal relations. Harel and Moskovitch [27] suggest TIRPClo to efficiently discover the complete set of frequent closed TIRPs, i.e., a compact subset of all the frequent TIRPs, from which their complete information can be revealed.

Wu and Chen [28] propose TPrefixSpan for mining nonambiguous temporal patterns from interval-based events by adopting a sequential mining algorithm PrefixSpan [29]. The proposed approach represents each time interval as a start-time event and an end-time event and discovers frequent sequences of the start-time and end-time events.

In [30], the authors introduce concurrency-aware process execution variants that account for time interval data of activities. As these variants represent a tree structure, frequent patterns can be mined using frequent subtree mining approaches, such as [31, 32]. The process mining tool Cortado [33] features frequent pattern mining from concurrency-aware process execution variants.

In this work, we use Cortado [33] in the evaluation of our proposed action engine. By exploiting the tailored approach to analyze business processes, we discover frequent temporal patterns of operational constraints and identify relevant problems to tackle with actions generated by our proposed action engine. In particular, the tool supports the discovery of temporal patterns in a binary tree structure that facilitates the interpretation by domain experts. Moreover, the tool is publicly available, ensuring the reproducibility of our evaluation results.

2.4. Process Management Actions

In this work, we conceptualize an action as a change in Process-Aware Information Systems (PAISs) that include ERP (Enterprise Resource Planning), CRM (Customer Relationship Management), WFM (Workflow Management Systems), and BPMS (Business Process Management Systems) [34]. In contrast to data- or function-centered information systems, PAISs are characterized by a strict separation of process logic and application code. The process logic dictates the order and conditions for the execution of various activities within a process, e.g., order confirmations only after order payments. The application code includes business logic, user interface, data manipulation, computations, interactions with other systems, etc. Examples include calculating the total cost of items in an order, processing user

218 authentication, communicating with the payment gateway to process the
219 transaction, and updating the database.

220 An action corresponds to a change in the process logic or the applica-
221 tion code. In the following, we explain actions in process logic and ones in
222 application code.

223 2.4.1. Actions in Process Logic

224 Several approaches have been proposed to change the process logic of
225 PAISs: *process redesign*, *process configuration*, and *process adaptation*. The
226 process logic of PAISs is often described by explicit process models, e.g.,
227 Workflow Nets [35] and Business Process Model and Notation (BPMN) [36].
228 Such a process model prespecifies activities to be executed, their control
229 flow dependencies, the organizational entities performing the activities, the
230 data objects manipulated by them, and the software applications (e.g., order
231 management systems) needed. Based on a process model, process instances
232 can be created at run-time, each representing a concrete business case (e.g.,
233 an order by customer Adams). *Process redesign* aims to improve business
234 processes by modifying process models at design time and, thus, influencing
235 the process instances created at run-time by the model. Control flow patterns
236 are used to modify process models, e.g., by enumerating different alternatives
237 in the process model or by allowing activities to be executed in parallel [37].

238 However, it would be too costly for organizations to design and imple-
239 ment standardized business processes from scratch. For these reasons, there
240 is a great interest in capturing common process knowledge only once and
241 using it in terms of reference process models (reference processes for short).
242 *Process configuration* aims to design a reference process model capturing
243 the behavior of all process variants; i.e., a reference process model merges a
244 multitude of process variants into one configurable model capturing both the
245 commonalities and the differences of the process variants [38]. In such a refer-
246 ence process model, variation points are represented in terms of configurable
247 nodes and edges [38]. By configuring these, the behavior of the reference
248 process model can be customized to the given context. Modeling languages
249 supporting this approach include Configurable Event-driven Process Chains
250 (C-EPC) [39] and C-YAWL [40].

251 A business process cannot always be executed in accordance with process
252 models at design time due to emerging exceptions or special situations [41].
253 Hence, authorized users should be allowed to situationally adapt single pro-
254 cess instances during run-time to cope with unanticipated exceptions, e.g., by

255 inserting, deleting, or moving activities. *Process adaptations* aim to support
 256 such ad hoc deviations from a designed process model while ensuring PAIS ro-
 257 bustness to end-users. Reichert and Dadam [42] present an approach for the
 258 support of dynamic structural changes of running process instances. Based
 259 upon a formal model called ADEPT, they define a complete and minimal set
 260 of change operations called ADEPTflex, which support users in modifying
 261 the structure of a running process model while maintaining its structural
 262 correctness and consistency. Weber et al. [43] suggest 18 change patterns
 263 independent of concrete implementations and constitute solutions to typical
 264 changes. They enable structural changes of process models at a high level
 265 of abstraction, e.g., by adding, deleting, or moving activities and process
 266 fragments, respectively. Furthermore, adaptation patterns can be applied at
 267 both the process type and the process instance level.

268 2.4.2. *Actions in Application Code*

269 Some approaches focus on revising the application code, specifically the
 270 business rules governing the functionality of individual activities, in order to
 271 enhance business processes. In the realm of software engineering, such mod-
 272 ification initiatives fall under the category of *maintenance* [44]. A widely
 273 accepted classification of software maintenance identifies 12 types, including
 274 training, consultive, evaluative, reformative, updative, groomative, preven-
 275 tive, performance, adaptive, reductive, corrective, and enhancive [45].

276 Among these, enhancive maintenance activities stand out for their focus
 277 on changing business rules to extend or restrict the software’s functionality
 278 and accessibility. These amendments can involve the addition, elimination,
 279 or modification of business rules that dictate the functioning of specific ac-
 280 tivities within the application. For instance, Wermelinger et al. [46] propose
 281 a framework that permits users to define multiple business rules based on
 282 predetermined parameters, thereby enabling the adaptation of these rules in
 283 response to changing business needs.

284 Similarly, Charifi and Mezini [47] suggest an approach that divides a
 285 process into a core part and a business rule part. This division modularizes
 286 the business rule segment, enabling it to exist and evolve independently.
 287 They achieved this by implementing the business rules as aspects using the
 288 BPEL extension AO4BPEL, which can be dynamically (un)deployed during
 289 the process interpretation stage.

290 In summary, various approaches have been proposed in the literature to
 291 define actions as changes in PAISs. In this work, we aim to develop a pattern-

292 based action engine that plans such actions by analyzing temporal patterns
 293 of operational constraints and resolving conflicts among such actions. The
 294 definition of actions is outside the scope of this paper.

295 3. Background

296 As depicted in Figure 1, this work considers operational constraints as
 297 temporally dependent and interval-based data, called *constraint instances*.
 298 By analyzing *temporal relations* of such constraint instances, we produce
 299 *action plans* that describe which actions should be executed and how they
 300 should be scheduled to avoid conflicts. In the following, we formally introduce
 301 the constraint instances, temporal relations, and action plans.

302 3.1. Constraint Instances

303 An operational constraint in business processes represents an operational
 304 problem, e.g., the high frequency of “change price” in an O2C process (cf.
 305 *C1* depicted in Figure 1(a)). Such constraints may occur in a *time window*.
 306 A time window describes the period between a start and an end timestamp.

307 **Definition 1 (Time Window).** Let $\mathbb{U}_{time}$ be the totally ordered set of all
 308 possible timestamps. $\mathbb{U}_{tw} = \{(t_s, t_e) \in \mathbb{U}_{time} \times \mathbb{U}_{time} \mid t_s < t_e\}$ is the set
 309 of all possible time windows. For any $tw = (t_s, t_e) \in \mathbb{U}_{tw}$, $\pi_{st}(tw) = t_s$
 310 and $\pi_{et}(tw) = t_e$. For any two time windows $tw_1, tw_2 \in \mathbb{U}_{tw}$, $tw_1 \cup tw_2 =$
 311 $(\min(\{\pi_{st}(tw_1), \pi_{st}(tw_2)\}), \max(\{\pi_{et}(tw_1), \pi_{et}(tw_2)\}))$.

312 For instance, $tw_1 = (2023-01-01\ 00:00:00, 2023-01-08\ 00:00:00)$ is a time window
 313 where $\pi_{st}(tw_1) = 2023-01-01\ 00:00:00$ and $\pi_{et}(tw_1) = 2023-01-08\ 00:00:00$. For tw_1
 314 and $tw_2 = (2023-01-08\ 00:00:00, 2023-01-15\ 00:00:00)$, $tw_1 \cup tw_2 = (2023-01-01\ 00:00:00,$
 315 $2023-01-15\ 00:00:00)$.

316 A constraint instance refers to the presence of an operational constraint
 317 during a specific time window, e.g., the presence of operational constraint *C1*
 318 in Figure 1(a) from 2023-01-01 to 2023-01-08.

319 **Definition 2 (Constraint Instance).** Let $\mathbb{U}_{cn}$ be the universe of opera-
 320 tional constraint names. $\mathbb{U}_{ci} = \mathbb{U}_{cn} \times \mathbb{U}_{tw}$ is the universe of constraint in-
 321 stances. For $ci = (cn, tw) \in \mathbb{U}_{ci}$, $\pi_{cn}(ci) = cn$ and $\pi_{tw}(ci) = tw$.

For instance, $(C1, tw_1) \in \mathbb{U}_{ci}$ denotes a constraint instance representing the presence of operational constraint $C1$ during time window tw_1 .
 Figure 2 shows a set of constraint instances, $CI_1 = \{ci_1, \dots, ci_4\} \subseteq \mathbb{U}_{ci}$.
 For instance, $ci_1 = (C1, (t1, t3))$ denotes a constraint instance representing the presence of operational constraint $C1$ from $t1$ to $t3$.

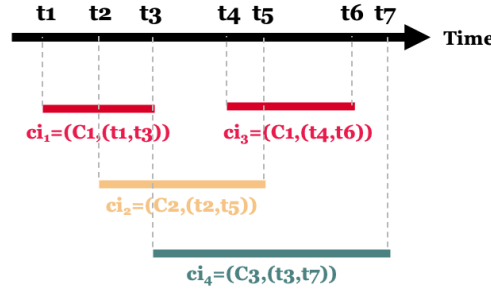


Figure 2: Examples of constraint instances, $CI_1 = \{ci_1, \dots, ci_4\} \subseteq \mathbb{U}_{ci}$.

3.2. Temporal Relations

A temporal relation indicates a relation between two time windows, e.g., a time window *overlaps with* the other time window.

Definition 3 (Temporal Relations). $Rel \subseteq \mathbb{U}_{tw} \times \mathbb{U}_{tw}$ is a temporal relation. We denote $\mathcal{R}$ to be the set of all possible temporal relations. Any two time windows belong to exactly one relation, i.e., $\forall (tw_1, tw_2) \in \mathbb{U}_{tw} \times \mathbb{U}_{tw} \exists Rel \in \mathcal{R} \forall Rel' \in \mathcal{R} \setminus \{Rel\} (tw_1, tw_2) \in Rel \wedge (tw_1, tw_2) \notin Rel'$.

Allen [22] suggests 13 temporal relations as shown in Figure 3. For instance, *before* $\in \mathcal{R}$ is a temporal relation describing that a time window is before the other time window, i.e., $before = \{(tw_1, tw_2) \in \mathbb{U}_{tw} \times \mathbb{U}_{tw} \mid \pi_{et}(tw_1) < \pi_{st}(tw_2)\}$. Moreover, *overlaps* $\in \mathcal{R}$ is a temporal relation describing that a time window overlaps with the other time window, i.e., $overlaps = \{(tw_1, tw_2) \in \mathbb{U}_{tw} \times \mathbb{U}_{tw} \mid \pi_{st}(tw_1) < \pi_{st}(tw_2) < \pi_{et}(tw_1) < \pi_{et}(tw_2)\}$.

Any two constraint instances have exactly one temporal relation. For example, $ci_1 = (C1, (t1, t3))$ is before $ci_3 = (C1, (t4, t6))$ as $((t1, t3), (t4, t6)) \in before$. Moreover, $ci_1 = (C1, (t1, t3))$ overlaps with $ci_2 = (C2, (t2, t5))$ as $((t1, t3), (t2, t5)) \in overlaps$. We analyze such temporal relations to produce *action plans*.

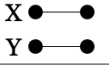
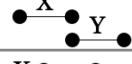
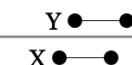
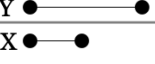
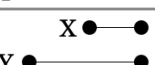
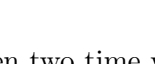
Relation	Relation for Inverse	Example
X before Y	Y after X	
X equal Y	Y equal X	
X meets Y	Y met by x	
X overlaps Y	Y overlapped by X	
X during Y	Y contains X	
X starts Y	Y started by X	
X finishes Y	Y finished by X	

Figure 3: Allen’s temporal relations between two time windows

3.3. Action Plans

An *action plan* describes the execution of actions (e.g., $A1$ is executed from 6. Feb.2023 to 8. Feb.2023 and $A3$ is executed from 8. Feb.2023 to 11. Feb.2023). We deliberately abstract from specific definitions of actions, denoting $\mathbb{U}_{action}$ as the collection of actions introduced in Subsection 2.4.

Definition 4 (Action Plan). Let $\mathbb{U}_{action}$ be the universe of actions. $AP \subseteq \mathbb{U}_{action} \times \mathbb{U}_{tw}$ is an action plan. We denote $\mathbb{U}_{AP}$ to be the set of all possible action plans.

For example, Figure 4 shows an action plan, $AP_1 = \{(A1, (t11, t12)), (A2, t11, t12)), (A3, (t8, t10))\}$, specifying that $A3$ should be executed from $t8$ to $t10$, while $A1$ and $A2$ should be performed from $t10$ to $t12$.

In the following, we explain how a *pattern-based action engine* produces such action plans by analyzing temporal relations inherent in constraint instances.

4. Pattern-based Action Engine

This section introduces a pattern-based action engine. Figure 5 provides an overview of the proposed approach, which is composed of three phases.

t8	t9	t10	t11	t12
			A1	
			A2	
A3				

Figure 4: An example of action plans, AP_1 .

First, domain experts design *action graphs*. These are graphical notations that specify how to analyze temporal patterns of operational constraints and subsequently generate appropriate actions. Each action graph is constituted by *temporal pattern trees* and *actions*. A temporal pattern tree represents the specific temporal pattern of operational constraints being addressed, while an action represents the corrective measure undertaken in response to that pattern. Subsequently, the pattern-based action engine generates *action requirements* by analyzing constraint instances using the action graphs. An action requirement describes requisite actions, along with their duration. Lastly, the pattern-based action engine formulates action plans by scheduling the requisite actions while taking into account the duration and *action conflicts*.

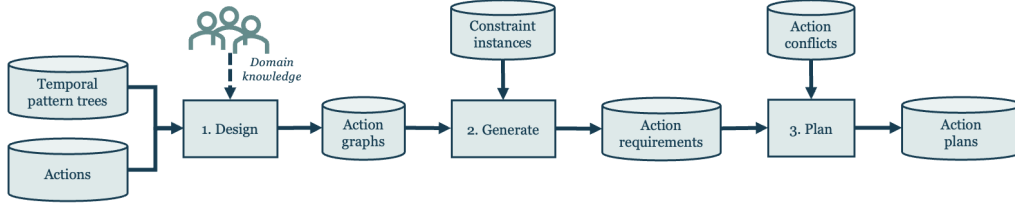


Figure 5: Overview of the proposed pattern-based action engine

4.1. Designing Action Graphs

In this work, we use temporal pattern trees to represent temporal relations among a variable number of operational constraints. Leaves represent operational constraints, and inner nodes represent temporal relations between their subtrees.

Definition 5 (Temporal Pattern Tree Syntax). A temporal pattern tree $pt = (V, E, \lambda, r)$ consists of a totally ordered set of nodes V , a set of edges $E \subseteq V \times V$, a labeling function $\lambda \in V \rightarrow \mathbb{U}_{cn} \cup \mathcal{R}$, and a root node $r \in V$.

- 382 • $(\{v\}, \emptyset, \lambda, v)$ with $\lambda(v) \in \mathbb{U}_{cn}$ is a temporal pattern tree.
- 383 • Given two different temporal trees $pt_1 = (V_1, E_1, \lambda_1, r_1)$ and $pt_2 =$
384 $(V_2, E_2, \lambda_2, r_2)$ with $r \notin V_1 \cup V_2$, $pt = (V, E, \lambda, r)$ is a temporal pat-
385 tern tree such that
 - 386 – $V = V_1 \cup V_2 \cup \{r\}$
 - 387 – $E = E_1 \cup E_2 \cup \{(r, r_1), (r, r_2)\}$
 - 388 – $\lambda(v) = \lambda_i(v)$ for any $i \in \{1, 2\}$ and $v \in V_i$
 - 389 – $\lambda(r) \in \mathcal{R}$

390 $\mathbb{U}_{pt}$ denotes the set of all possible temporal pattern trees.

391 Figure 6 shows a temporal pattern tree pt_0 . The tree describes that $C1$ “a
392 high frequency of *change price*” happens during $C3$ “a high avg. throughput
393 time of orders”. This pattern of $C1$ and $C3$ overlaps with $C2$ “a high avg.
394 sojourn time of *change price*”.

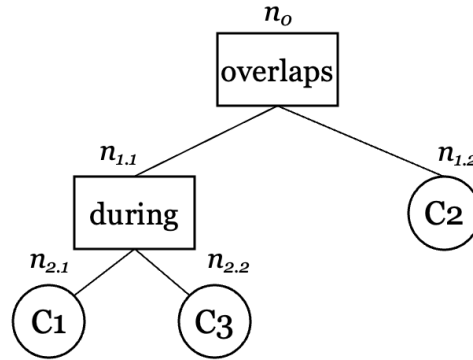


Figure 6: An example temporal pattern tree $pt_0 = (\{n_0, n_{1,1}, n_{1,2}, n_{2,1}, n_{2,2}\}, \{(n_0, n_{1,1}), \dots, (n_{1,1}, n_{2,2})\}, \lambda_0, n_0)$ with $\lambda_0(n_0) = overlaps, \dots, \lambda_0(n_{2,2}) = C3$, where $C1$ denotes a high frequency of “change price”, $C2$ denotes a high avg. sojourn time of “price change”, and $C3$ denotes a high avg. throughput time of orders (cf. Figure 1(a)).

395 Note that every temporal relation (inner node) has exactly two chil-
396 dren. Next to the graphical representation, any temporal pattern tree can
397 be textually represented because of its totally ordered node set, e.g., $pt_0 \hat{=}$
398 $overlaps(during(C1, C3), C2)$.

For a tree $pt = (V, E, \lambda, r)$, $L(pt) = \{v \in V \mid \nexists_{v' \in V} (v, v') \in E\}$ denotes its *leaf nodes*. Assuming that for each node, the left child appears first in the order, we define the left child of a node $v \in V$ as:

$$lc(pt, v) = \begin{cases} w & \text{if } (v, w) \in E \wedge \nexists_{u \in V, u < w} (v, u) \in E \\ \emptyset & \text{otherwise.} \end{cases}$$

399 Then, assuming that E^+ denotes the transitive closure E , the leaf nodes
400 of the left child of a vertex $v \in V$ are defined as follows:

$$LL(pt, v) = \begin{cases} \{w \in V \mid (lc(pt, v), w) \in E^+ \wedge \nexists_{u \in V} (w, u) \in E\} & \text{if } lc(v) \neq \emptyset \\ \emptyset & \text{otherwise.} \end{cases}$$

401 For instance, $lc(pt_0, n_0) = n_{1.1}$ and $LL(pt_0, n_0) = \{n_{2.1}, n_{2.2}\}$. We define
402 $RL(pt, v)$ to be the leaf nodes of the right child of $v \in V$ in the same manner.

403 We define the semantics of temporal pattern trees in relation to a set of
404 constraint instances with the notion of *occurrences*.

405 **Definition 6 (Temporal Pattern Tree Semantics).** Let $pt = (V, E, \lambda, r)$
406 $\in \mathbb{U}_{pt}$ be a temporal pattern tree. Let $CI \subseteq \mathbb{U}_{ci}$ be a set of constraint in-
407 stances. The function $occurs(pt, CI) = true$ if there is an injective mapping
408 $\mu \in L(pt) \rightarrow CI$ such that

- 409 • $\forall_{v \in L(pt)} \lambda(v) = \pi_{cn}(\mu(v))$ and
- 410 • $\forall_{v \in V \setminus L(pt)} \left(\bigcup_{v' \in LL(pt, v)} \pi_{tw}(\mu(v')), \bigcup_{v' \in RL(pt, v)} \pi_{tw}(\mu(v')) \right) \in \lambda(v)$.

411 Otherwise, $occurs(pt, CI) = false$.

412 Figure 7 shows an example of an injective mapping μ_1 that demonstrates
413 that $occurs(pt_0, CI_1) = true$. Formally, $\mu_1 = \{(n_{2.1}, ci_3), (n_{2.2}, ci_4), (n_{1.2}, ci_2)\}$
414 such that

- 415 • $\lambda(n_{2.1}) = \pi_{cn}(ci_3) = C1$, $\lambda(n_{2.2}) = \pi_{cn}(ci_4) = C3$, $\lambda(n_{1.2}) = \pi_{cn}(ci_2) =$
416 $C2$,
- 417 • $(\pi_{tw}(ci_3), \pi_{tw}(ci_4)) \in during$, and
- 418 • $(\pi_{tw}(ci_3) \cup \pi_{tw}(ci_4), \pi_{tw}(ci_2)) \in overlaps$.

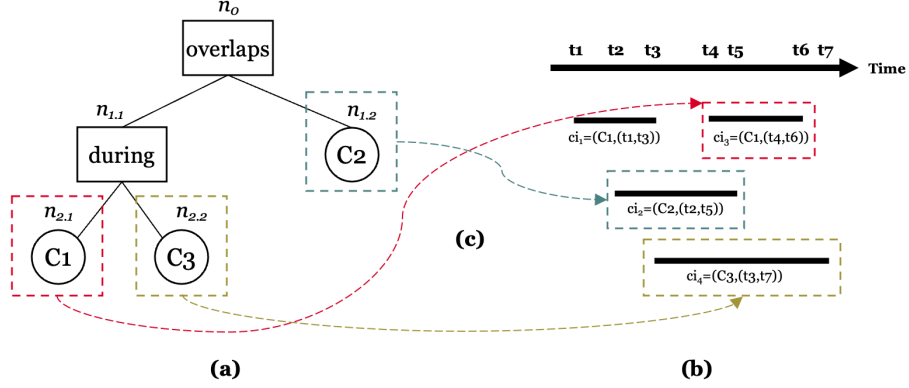


Figure 7: (a) Temporal pattern tree pt_0 (cf. Figure 6), (b) Set of constraint instances CI_1 (cf. Figure 2), and (c) An injective mapping demonstrating that the temporal pattern tree *occurs* in the set of constraint instances.

419 An action graph is a graphical notation to formally represent 1) how
 420 to analyze constraint instances using temporal pattern trees and 2) how to
 421 trigger actions. It consists of two types of nodes, i.e., temporal pattern
 422 trees and actions, and edges connecting the nodes (cf. Figure 8). Each
 423 edge represents that the associated pattern is addressed by the corresponding
 424 action and is labeled with the required time duration for the remedial action.

425 **Definition 7 (Action Graph).** *An action graph is a graph $ag = (PT, A, PA, l)$*
 426 *where*

- 427 • $PT \subseteq \mathbb{U}_{pt}$ is a set of temporal pattern trees,
- 428 • $A \subseteq \mathbb{U}_{action}$ is a set of actions,
- 429 • $PA \subseteq PT \times A$ is a set of pattern-to-action edges, and
- 430 • $l \in PA \rightarrow \mathbb{R}$ maps pattern-to-action edges to time duration.

431 $\mathbb{U}_{ag}$ denotes the set of all possible action graphs.

432 Figure 8 shows an action graph ag_1 . It consists of six problems patterns,
 433 i.e., $pt_0, \dots, pt_5 \in \mathbb{U}_{pt}$, and three actions, i.e., $a_1, a_2, a_3 \in \mathbb{U}_{action}$. Each action
 434 reflects a different conceptual action type that we introduce in Subsection 2.4.

435 Action a_1 , i.e., batch processing price changes, for one day, addresses
 436 pt_1, pt_2, pt_3 . This action aligns with the concept of *process redesign*, as it

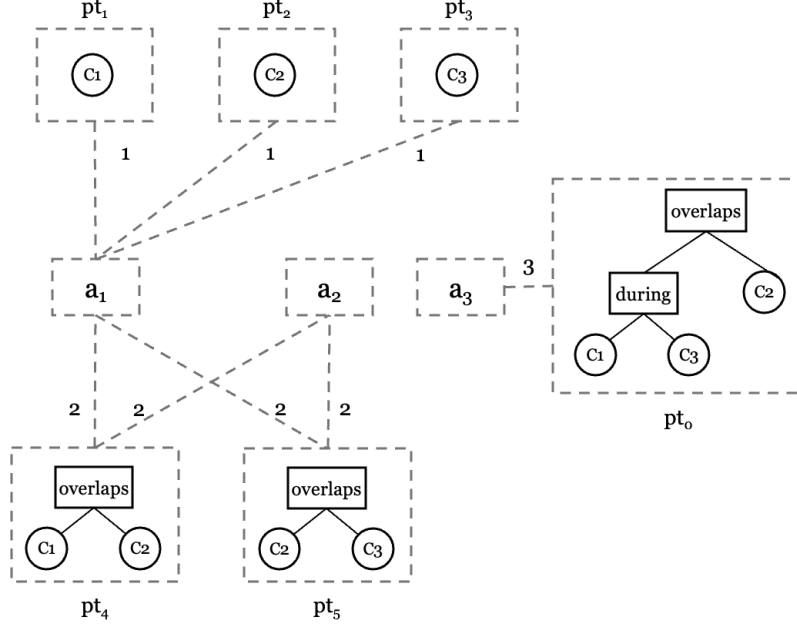


Figure 8: Action graph ag_1 consisting of six temporal patterns of operational constraints, i.e., $pt_0, \dots, pt_5 \in \mathbb{U}_{pt}$, and three actions, i.e., $a_1, a_2, a_3 \in \mathbb{U}_{action}$

437 involves modifying the process to accumulate price changes and apply them
 438 in batches rather than making individual changes. Next, patterns pt_4 and pt_5
 439 are tackled by a_2 , i.e., adding more sales staff to support the price changes
 440 for two time units, as well as a_1 for two time units. The action a_2 aligns
 441 with the concept of *enhancive maintenance*, which involves changing the
 442 application code to extend functionality, such as managing and assigning
 443 tasks to an increased number of sales staff. Finally, pt_0 is addressed with a_3 ,
 444 i.e., disabling “change” price, for three time units. This action aligns with
 445 the concept of *process configuration*, as it involves disabling or restricting the
 446 accessibility of an activity.

447 Formally, $ag_1 = (PT_1, A_1, PA_1, l_1)$ where $PT_1 = \{pt_0, \dots, pt_5\}$, $A_1 =$
 448 $\{a_1, a_2, a_3\}$, $PA_1 = \{(pt_1, a_1), \dots, (pt_0, a_3)\}$, $l_1(pt_1, a_1) = 1$, $l_1(pt_0, a_3) = 3$,
 449 etc.

450 4.2. Generating Actions

451 Based on action graphs, we analyze constraint instances to generate *ac-*
 452 *tion requirements*. An action requirement describes actions with required

453 durations. For example, a_1 is required for two time units, a_2 is required for
 454 two time units, and a_3 is required for three time units. An action require-
 455 ment refers to unique actions, i.e., it does not describe the same action with
 456 different duration (e.g., a_1 required for two time units and a_1 required for
 457 three time units).

458 **Definition 8 (Action Requirement).** *An action requirement $AR \subseteq \mathbb{U}_{action}$
 459 $\times \mathbb{R}$ is a set of action and duration pairs such that, for any $(a, r), (a', r') \in$
 460 AR , $a = a' \implies (a, r) = (a', r')$. $\mathbb{U}_{AR}$ denotes the set of all possible action
 461 requirements.*

462 For instance, $AR_1 = \{(a_1, 2), (a_2, 2), (a_3, 3)\} \in \mathbb{U}_{AR}$ refers to an action
 463 requirement that describes the execution of action a_1 for two time units, the
 464 execution of action a_2 for two time units, and the execution of action a_3 for
 465 two time units. Note that, for the sake of brevity, we define durations as real
 466 numbers.

467 Given a set of constraint instances and an action graph, an action genera-
 468 tor returns an action requirement. More in detail, it generates actions if any
 469 of their connected temporal patterns *occur* in the set of constraint instances,
 470 e.g., a_1 is generated if any of $pt_1, \dots, pt_5$ *occur* in a set of constraint instances.
 471 In this work, we associate the duration of a generated action with the maxi-
 472 mum duration among the ones specified by occurring temporal patterns, e.g.,
 473 a_1 's duration is 2 if both pt_1 and pt_5 *occur*.

474 **Definition 9 (Action Generator).** *An action generator $gen \in \mathcal{P}(\mathbb{U}_{ci}) \times$
 475 $\mathbb{U}_{ag} \rightarrow \mathbb{U}_{AR}$ maps a set of constraint instances and an action graph to an
 476 action requirement. For any $CI \subseteq \mathbb{U}_{ci}$ and $ag = (PT, A, PA, l) \in \mathbb{U}_{ag}$,
 477 $gen(CI, ag) = \{(a, dur) \in \mathbb{U}_{action} \times \mathbb{R} \mid (\exists_{(pt, a) \in PA} \text{occurs}(pt, CI) = \text{true} \wedge$
 478 $dur = l(pt, a)) \wedge (\nexists_{(pt', a) \in PA} \text{occurs}(pt', CI) = \text{true} \wedge dur < l(pt', a))\}$.*

479 Given $CI_1 \subseteq \mathbb{U}_{ci}$ described in Figure 7 and ag_1 depicted in Figure 8,
 480 $gen(CI_1, ag_1) = AR_1 = \{(a_1, 2), (a_2, 2), (a_3, 3)\}$. First, $(a_1, 2)$ is generated
 481 since pt_1, pt_2, pt_3, pt_4 , and pt_5 *occur* in CI_1 , and the maximum duration
 482 specified in the action graph is 2. Second, $(a_2, 2)$ is generated since pt_4 ,
 483 and pt_5 *occur* in CI_1 , and the maximum duration specified in the action
 484 graph is 2. Finally, $(a_3, 3)$ is generated since $\text{occurs}(pt_0, CI_1) = \text{true}$, and
 485 the maximum duration specified in the action graph is 3.

486 4.3. Planning Actions

487 Based on action requirements, we plan the execution of actions in the
488 action requirement, e.g., a_1 is executed from 6. Feb.2023 to 8. Feb.2023, and a_3 is
489 executed from 8. Feb.2023 to 11. Feb.2023, ensuring the resolution of any *conflicts*
490 between actions. A conflict between actions can be interpreted in multiple
491 ways, and for this work, we categorize them as follows:

- 492 • Temporal Conflicts: These conflicts stem from the timing and schedul-
493 ing of actions. They can be further classified into:
 - 494 – Concurrent Execution Conflicts: These occur when two or more
495 actions are planned to happen simultaneously. If action $a1$ in-
496 volves shutting down a machine for maintenance and action $a2$
497 involves running a diagnostic test on the same machine, schedul-
498 ing both actions at the same time would create a conflict.
 - 499 – Sequential Execution Conflicts: These occur when the order of ac-
500 tion execution is essential, such as when one action must conclude
501 before another begins. If action $a3$ requires the results of action
502 $a1$ for its execution, then $a1$ must complete before $a3$ starts.
- 503 • Effect-based Conflicts: These conflicts emerge based on the aftermaths
504 of the actions. They can be further segmented into:
 - 505 – Incompatibility Conflicts: These conflicts arise when the execution
506 of one action nullifies the effect of another. If action $a4$ is meant
507 to increase system security but action $a5$ inadvertently creates a
508 security vulnerability, the two actions are in conflict.
 - 509 – Counterproductivity Conflicts: These conflicts appear when the
510 outcome of an action interferes with or obstructs the overall objec-
511 tive. If the objective is to reduce energy consumption and action
512 $a6$ reduces energy use but action $a7$ increases it, this creates a
513 counterproductivity conflict.
- 514 • Dependency-based Conflicts: These conflicts emerge due to the depen-
515 dency relationships between actions. They can be further broken down
516 into:
 - 517 – Interdependency Conflicts: These conflicts arise when certain ac-
518 tions must be carried out together for a successful operation. If

519 action a_8 sets a system configuration that action a_9 relies on,
 520 failing to execute both actions together would result in a conflict.

521 In this work, our primary focus lies in resolving temporal conflicts among
 522 actions. To this end, an action conflict model is designed to express the
 523 temporal conflict between actions and precedence constraints. For instance,
 524 an action conflict may describe that a_3 has a conflict with a_1 , i.e., disabling
 525 *change price* cannot be executed simultaneously with alerting sales staff re-
 526 sponsible for *change price*, and the conflict between a_3 and a_1 is resolved by
 527 executing a_3 before a_1 .

528 **Definition 10 (Action Conflict).** *An action conflict $AP = (A, C)$ is a*
 529 *directed acyclic graph where $A \subseteq \mathbb{U}_{action}$ and $C \subseteq A \times A$ such that $(a, a) \notin C^+$*
 530 *for all $a \in A$, where C^+ denotes the transitive closure of C . $\mathbb{U}_{ac}$ denotes the*
 531 *set of all possible action conflicts.*

532 Figure 9(b) shows an example action conflict, $ac_1 = (A_1, C_1)$ with $A_1 =$
 533 $\{a_1, a_2, a_3\}$ and $C_1 = \{(a_3, a_1), (a_3, a_1)\}$. Action a_3 has a conflict with action
 534 a_1 , and the conflict is resolved by executing a_3 before a_1 . Moreover, action
 535 a_3 has a conflict with action a_2 , and the conflict is resolved by executing a_3
 536 before a_2 .

537 Given an action requirement and an action conflict, an action planner
 538 produces an action plan. For each action specified in the action requirement,
 539 the action planner produces an action instance that describes the execution
 540 of the corresponding action with a start timestamp and an end timestamp.
 541 While doing so, the action planner ensures that no conflicts exist between
 542 the action instances using the action conflict, i.e., no conflicting actions are
 543 simultaneously executed.

544 **Definition 11 (Action Planner).** *An action planner $plan \in \mathbb{U}_{AR} \times \mathbb{U}_{ac} \rightarrow$*
 545 *$\mathbb{U}_{AP}$ maps an action requirement and an action conflict to an action plan.*
 546 *For any $AR \in \mathbb{U}_{AR}$ and $(A, C) \in \mathbb{U}_{ac}$,*

- 547 • $\{(a, \pi_{et}(tw) - \pi_{st}(tw)) \mid (a, tw) \in plan(AR, ac)\} = AR$ and
- 548 • $\forall_{(a, a') \in C} \forall_{(a, tw), (a', tw') \in plan(AR, ac)} \pi_{et}(tw) \leq \pi_{st}(tw')$.

549 Figure 9 shows three action planners that produce different action plans,
 550 given action requirement AR_1 and action conflict ac_1 . For three actions

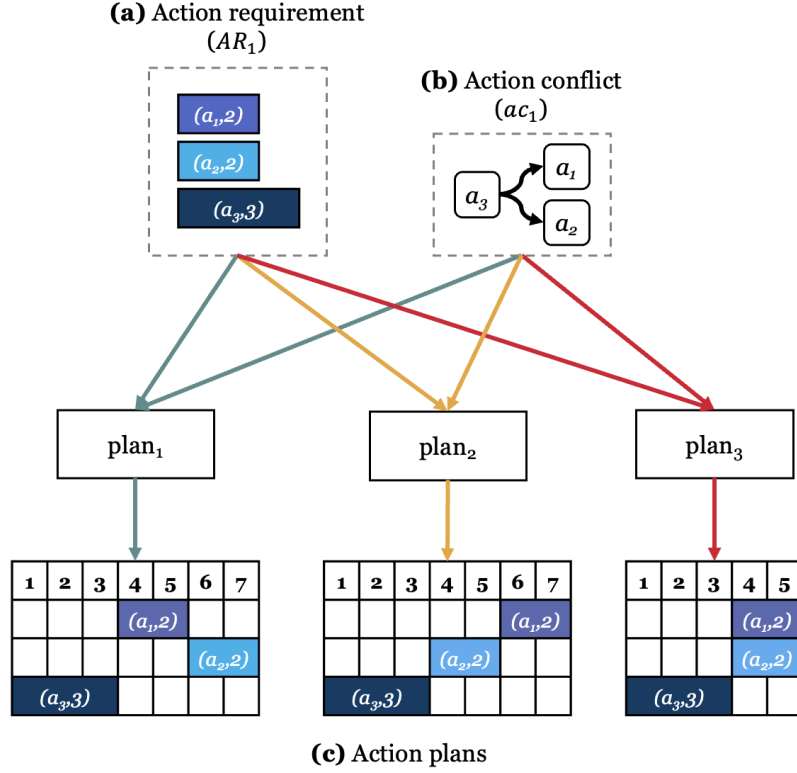


Figure 9: Examples of action planners: (a) An action requirement (AR_1), (b) An action conflict (ac_1), and (c) Different action plans computed from the action requirements and action conflict.

specified in AR_1 , action planner $plan_1$ produces an action plan with three action instances, i.e., $plan_1(AR_1, ac_1) = \{(a_1, (4, 5)), (a_2, (6, 7)), (a_3, (1, 3))\}$. Action instance $(a_1, (3, 5))$ is produced for $(a_1, 2) \in AR_1$, while $(a_2, (5, 7))$ and $(a_3, (0, 3))$ are produced for $(a_2, 2)$ and $(a_3, 3)$, respectively. Actions a_1 and a_2 are planned after a_3 to resolve the conflict specified in ac_1 .

In this work, we implement an action planner that minimizes the makespan, i.e., the total time taken to finish all action instances (e.g., the leftmost action plan in Figure 9(c) has the makespan of 7). By minimizing the makespan, we ensure that actions are executed in the shortest possible time, which in turn enables a faster response to operational issues. However, makespan minimization is not the only possible objective for an action planner. Depending on the specific requirements and constraints of the process and problem sce-

nario, other potential objectives could include minimizing the total cost of actions, maximizing the quality or effectiveness of actions, etc. For instance, in scenarios where action costs are a concern, an action planner might seek to minimize the total cost of actions. On the other hand, in cases where the quality of the actions is paramount, the planner might aim to maximize the overall effectiveness of the actions, potentially at the expense of longer makespan or higher costs.

We formulate the problem as a parallel machine scheduling problem, i.e., scheduling n jobs (i.e., n actions in the given action requirement) on m parallel machines (i.e., m entities which execute actions) under precedent constraints (i.e., the action conflict), while minimizing the makespan (denoted as $P_m|prec|C_{max}$) [48].

$P_m|prec|C_{max}$, is an NP-hard problem [49]. However, a polynomial time algorithm to solve the problem exists if there are an unlimited number of machines (i.e., $m \geq n$) [48]. We implement an action planner under the assumption that there are an unlimited number of entities that can execute the actions. Algorithm 1 describes the action planner algorithm. Given an action requirement and an action conflict, we schedule the actions in the action requirement that have no conflicts one at a time, starting at time 0. Whenever an action has been completed, we schedule the actions of which all predecessors have been completed.

The proposed algorithm produces the third action plan described in Figure 9. The action plan has the minimum makespan of 5 time units, given the action requirement and action conflict.

5. Evaluation

This section evaluates the action engine proposed in Section 4. First, we evaluate the feasibility of the proposed action engine by demonstrating a use case using the data of a real-life loan application process. Next, we conduct experiments to evaluate the scalability and performance of the proposed action engine.

5.1. Use Case: Loan Application Process

We demonstrate a use case using the data of a real-life loan application process from a Dutch financial institute [50]. The execution of the loan application process involves an application and a variable number of offers. Figure 10 shows a process model as an object-centric Petri net [51]. First,

Algorithm 1 Action planner algorithm

Input: An action requirement $AR \in \mathbb{U}_{AR}$ and an action conflict $(A, C) \in \mathbb{U}_{ac}$

Output: An action plan $AP \in \mathbb{U}_{AP}$

```
1:  $AP \leftarrow \emptyset$ 
2:  $S \leftarrow \emptyset$      $\triangleright$  A set of pairs of a scheduled action and its completion time
3:  $t \leftarrow 0$ 
4: while  $AR \neq \emptyset$  do
5:   for  $(a, r) \in AR$  do
6:     if  $\{a' \mid (a', a) \in C\} \subseteq \text{complete}(S, t)$  then       $\triangleright$  complete is a
       function to return the set of completed actions at  $t$ 
7:        $S \leftarrow S \cup (a, t + r)$ 
8:        $AP \leftarrow AP \cup (a, (t, t + r))$ 
9:        $AR \leftarrow AR \setminus \{(a, r)\}$ 
10:    end if
11:  end for
12:   $t \leftarrow t + 1$ 
13: end while
14: return  $AP$ 
```

598 a customer creates an application by visiting the bank or using an online
 599 system. In the former case, *submit* activity is skipped. After the completion
 600 and acceptance of the application, the bank offers loans to the customer
 601 by sending the offer to the customer and making a call. An offer is either
 602 accepted or canceled.

603 In this use case, we focus on the offers canceled due to various reasons. In
 604 order to introduce operational constraints in the context of canceled applica-
 605 tions, we explain a process execution with a canceled application as depicted
 606 in Figure 11. The process execution starts with *Application1*. The applica-
 607 tion has its own lifecycle that consists of multiple activities such as *create*
 608 *application*, *submit*, *call incomplete files*, etc. The process execution involves
 609 two offers: *Offer1* and *Offer2*. The lifecycle of each offer includes activities
 610 such as *create offer*, *send offer*, *call applicants*, etc. The process execution
 611 finishes by canceling *Application1* with *Offer1* and *Offer2*.

612 During the process, we monitor the presence of the following operational
 613 constraints focusing on *cancel application (CA)* activity in the process:

- 614 1. *High ratio of CA's executions (CE)* which represents the high ratio of
 615 applications that are canceled.
- 616 2. *High ratio of CA's executions with multiple offers (MO)* which repre-
 617 sents the high ratio of *CA* events that involve multiple offers.
- 618 3. *High avg. flow time of CA (FT)* which represents the high avg. *flow*
 619 *time* of *CA* events; In Figure 11, *flow time* of *CA* is the time difference
 620 between the completion of *CA* and the completion of *Application1*'s
 621 lifecycle before *CA*.
- 622 4. *High avg. sojourn time of CA (SoT)* represents the high avg. sojourn
 623 time of *CA* events; In Figure 11, *sojourn time* of *CA* is the time dif-
 624 ference between the completion of *CA* and the completion of *Offer2*'s
 625 lifecycle before *CA*.
- 626 5. *High avg. synchronization time of CA (SyT)* represents the high avg.
 627 synchronization time of *CA* events; In Figure 11, *synchronization time*
 628 of *CA* is the time difference between the completion of *Offer2*'s lifecycle
 629 before *CA* and the completion of *Application1*'s lifecycle before *CA*.
- 630 6. *High avg. pooling time of CA w.r.t. offers (PT)* represents the high
 631 avg. pooling time of *cancel application* events w.r.t. offers; In Fig-

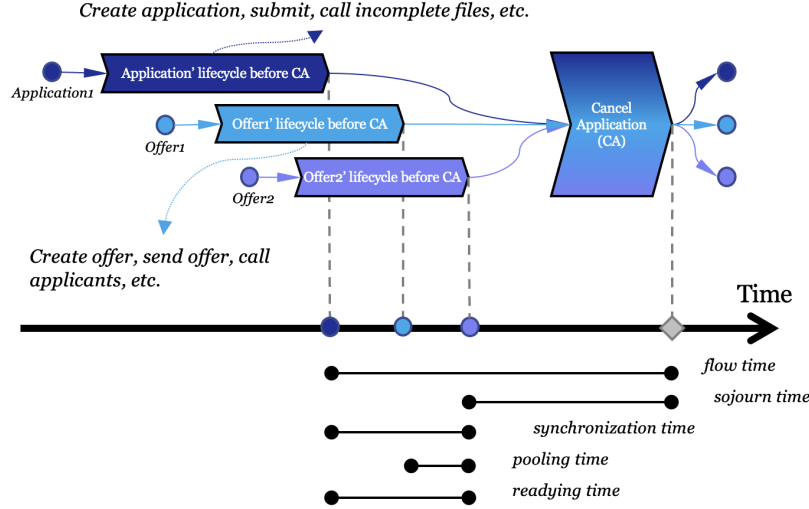


Figure 11: A process execution of the loan application process

Figure 11, *pooling time* of CA w.r.t. *offers* is the time difference between the completion of *Offer2*'s lifecycle before CA and the completion of *Offer1*'s lifecycle before CA.

7. *High avg. readiness time of CA w.r.t. applications (RT)* represents the high avg. readiness time of *cancel application* events w.r.t. applications; In Figure 11, *readiness time* of CA w.r.t. *applications* is the time difference between the completion of *Offer2*'s lifecycle before CA and the completion of *Application1*'s lifecycle before CA.

5.1.1. Designing Action Graphs

First, we define relevant temporal patterns of the operational constraints in the loan application process. Typically, these temporal patterns are derived from the expertise of domain experts. The application of frequent temporal pattern mining approaches aids these experts in specifying such patterns, as well as facilitates the validation of their occurrences in the historical process executions.

To discover these temporal patterns, we utilize the weekly monitoring results, spanning from January 2016 to September 2016. For this, we employ the frequent pattern mining approach implemented in a process mining tool [33]. Each weekly monitoring result comprises a week's constraint in-

stances, for example, the week starting from 01.01.2016 00:00:00 and ending at 01.07.2016 23:59:59.

From the 91 discovered frequent temporal patterns, we select those we aim to address. Our selection criteria are based on: (a) relevance to the business process, and (b) simplicity and interpretability of the patterns. In terms of (a), we exclude patterns such as $before(FT, CE)$, which refers to a high average flow time of CA *before* a high ratio of CA's executions, as they do not significantly contribute to understanding the problem areas of the loan application process.

In terms of (b), we disregard overly complex patterns such as $before(overlaps(CE, overlaps(MO, FT)), before(overlaps(SoT, SyT), overlaps(SoT, SyT)))$, despite their potential interest. These intricate conditions are too complex to elicit meaningful actions for tackling the problem.

However, it's important to note that these temporal patterns should ideally be confirmed by domain experts for maximum utility. Given our reliance on publicly available data and the unavailability of access to such experts in this evaluation, we have selected the most plausible temporal patterns based on the aforementioned criteria.

Figure 12 shows the selected six frequent temporal patterns: $pt_1^{loan}, \dots, pt_6^{loan}$.

- pt_1^{loan} portrays a situation where numerous applications are canceled (CE) amid a high average flow time for cancellation (FT). This pattern indicates the demands for cancellation are increasing, while the cancellation process is delayed, which can lead to customer dissatisfaction.
- pt_2^{loan} illustrates a scenario where multiple offers are presented (MO) in many canceled applications (CE), concurrent with a high average flow time for the cancellation (FT). This could mean unnecessary work is being done for applications destined for cancellation.
- pt_3^{loan} signifies a situation where a high synchronization time of CA (SyT) overlaps with a high sojourn time (SoT). This could indicate a bottleneck in the offering system, which eventually causes a bottleneck in the application management system.
- pt_4^{loan} represents a repeating occurrence of a high synchronization time (SoT) overlapping with a high sojourn time (SyT). The repetition

- could point to a systemic issue in the process synchronization mechanism, leading to unnecessary delays.
- pt_5^{loan} presents a situation where the synchronization of application cancellations is delayed (SyT) due to a high pooling time concerning offers (PT). This pattern may imply that cancellations are being processed too slowly due to resources being tied up in offer pooling.
 - pt_6^{loan} denotes a scenario where the synchronization of application cancellations (SyT) is hampered by a high readiness time regarding applications (RT). This pattern suggests that the readiness of applications is causing a slowdown in the cancellation synchronization process.

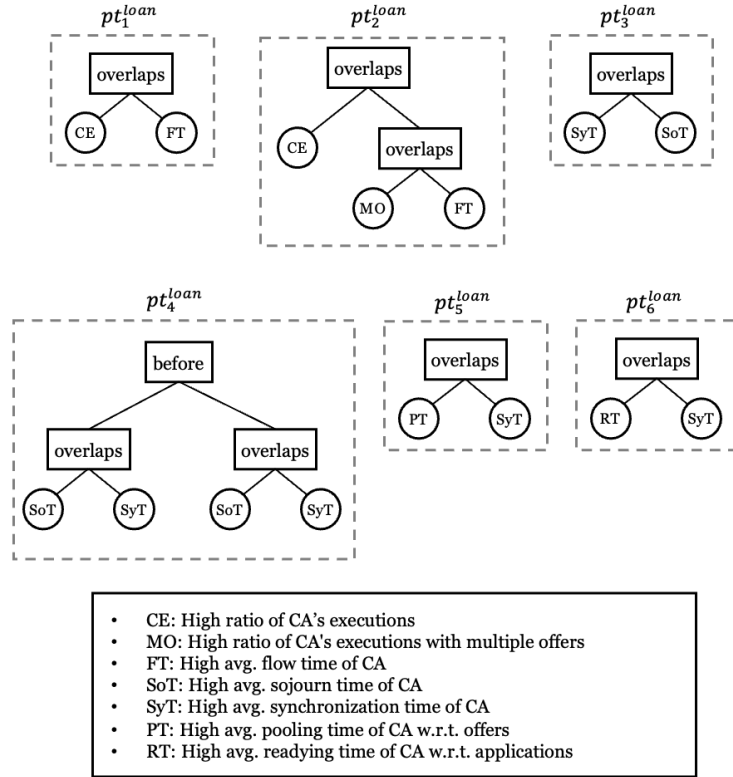


Figure 12: Temporal pattern trees defined over operational constraints including CE , MO , FT , SoT , SyT , PT , and RT

696 Next, we derive several actions to address the temporal patterns of oper-
 697 ational constraints, i.e., $a_1^{loan}, \dots, a_7^{loan} \in \mathbb{U}_{action}$. In order to determine the
 698 most appropriate actions, we consider the following factors: the selected tem-
 699 poral patterns and their business implications and the potential effectiveness
 700 and feasibility of the action within the loan application process. The elicited
 701 actions are as follows:

- 702 • a_1^{loan} is to send a warning message to the staff member responsible for
 703 the *cancel application* task,
- 704 • a_2^{loan} is to alter the business rule associated with the *create offer* activ-
 705 ity, so that the new rule encourages the acceptance of additional offers
 706 by applicants, e.g., by reducing the interest on loans for additional
 707 offers,
- 708 • a_3^{loan} is to modify the business rule associated with the *call applicants*
 709 activity to discourage repeated calls to applicants,
- 710 • a_4^{loan} is to revise the business rule associated with the *cancel application*
 711 activity to allow for additional resources to carry out the cancellation
 712 of applications,
- 713 • a_5^{loan} is to adjust the business rule associated with the *cancel application*
 714 activity to prevent the cancellation of applications with a single offer,
- 715 • a_6^{loan} is to send reminders to applicants about their pending offers,
 716 thereby reducing the likelihood of cancellations, and
- 717 • a_7^{loan} is to alert loan application managers.

718 Finally, we design an action graph using the temporal pattern trees and
 719 the actions. Figure 13 shows the action graph.

720 5.1.2. Generating and Planning Actions

721 Using the action graphs, we apply the action engine to the loan applica-
 722 tion process for 12 weeks from October 2016 to December 2016. In the first
 723 week, we analyze the constraint instances from 30.Sep.2016 to 06.Oct.2016
 724 and produce action instances starting from 07.Oct.2016, and, in the second
 725 week, we analyze the constraint instances from 07.Oct.2016 to 13.Oct.2016
 726 and produce action instances starting from 14.Oct.2016, etc. We use the
 727 action conflict depicted in Figure 14 to resolve conflicts between actions.

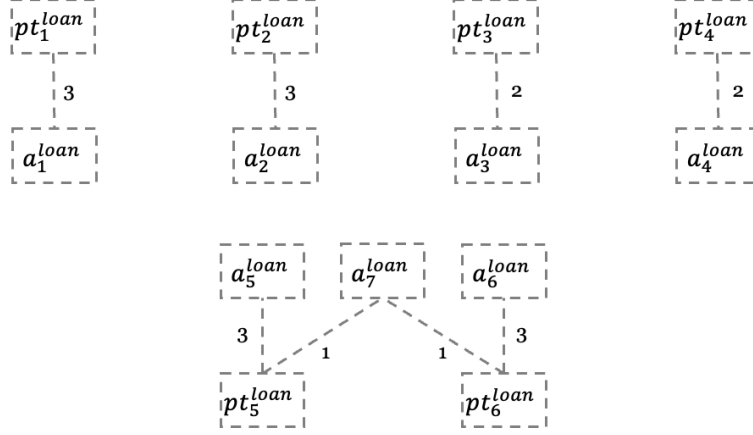


Figure 13: An action graph that describes the handling of temporal patterns $pt_1^{loan}, \dots, pt_6^{loan} \in \mathbb{U}_{pt}$ with actions $a_1^{loan}, \dots, a_7^{loan} \in \mathbb{U}_{action}$

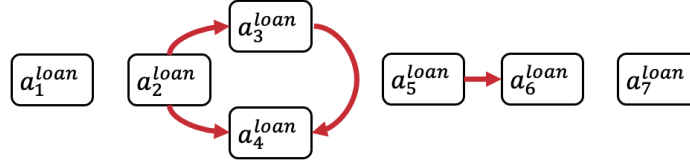


Figure 14: A conflict graph used to resolve conflicts between actions in the loan application process

728 Figure 15 shows the action plans computed by the action engine. For
729 instance, *action plan 2* in Figure 15 shows the action plan starting from
730 14.Oct.2016 and ending at 18.Oct.2016. From 07.Oct.2016 to 13.Oct.2016,
731 pt_1^{loan} , pt_3^{loan} , pt_4^{loan} , and pt_6^{loan} occur, and thus, the action generator gener-
732 ates an action requirement with $(a_1^{loan}, 4)$, $(a_3^{loan}, 2)$, $(a_4^{loan}, 2)$, $(a_6^{loan}, 3)$, and
733 $(a_7^{loan}, 2)$. Action a_1^{loan} , a_2^{loan} , a_5^{loan} , and a_6^{loan} are planned on 14.Oct.2016
734 and completed on 18.Oct.2016, 16.Oct.2016, 17.Oct.2016, and 15.Oct.2016,
735 respectively. Action a_4^{loan} is planned after a_3^{loan} is completed since a_3^{loan} is
736 conflicting with a_4^{loan} , and a_3^{loan} needs to be executed before a_4^{loan} .

737 Each action plan involves a different number of actions since the presence
738 of temporal patterns varies in different periods. For instance, *action plan 9*,
739 the action plan starting from 02.Dec.2016, comprises all the actions since all
740 the selected temporal patterns occur in the week starting from 25.Nov.2016

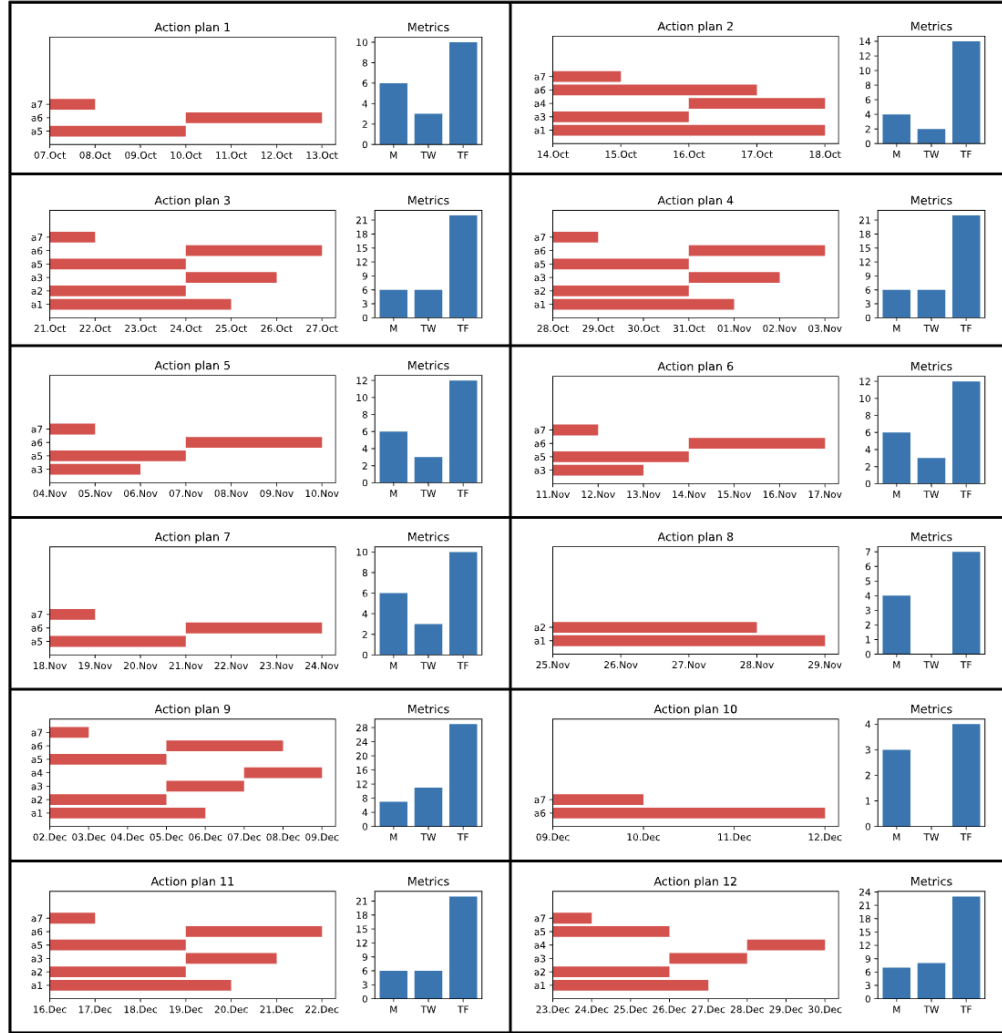


Figure 15: Action plans computed by the action engine for 12 weeks from October 2016 to December 2016. M, TW, and TF indicate makespan, Total Waiting time, and Total Flow time, respectively.

741 to 01.Dec.2016, which demonstrates the severity of problems in the week. In
 742 contrast, *action plan 10*, the action plan starting from 09.Dec.2016, includes
 743 two actions, i.e., a_6^{loan} and a_7^{loan} , since only one temporal pattern, i.e., pt_6 , is
 744 present in the week starting from 02.Dec.2016 to 08.Dec.2016.

745 We measure several performance metrics for the generated action plans,

including makespan, Total Waiting time (TW), and Total Flow time (TF). The waiting time of actions indicates the time taken to start the action, e.g., the waiting time of a_4^{loan} in *action plan 2* is 2 days, whereas the flow time of actions denotes the time taken to complete the action, e.g., the flow time of a_4^{loan} in *action plan 2* is 4 days. For instance, the makespan of *action plan 2* is 4, while the TW and TF are 2 and 14, respectively.

Different action plans show varying total waiting times. For instance, *action plan 9* has a total waiting time of 11, whereas *action plan 10* shows no waiting time. Depending on the type of actions, the waiting time could be essential, e.g., the earlier the action is executed, the higher its effect is. The waiting time for an action depends on the existence of its conflicting actions that need to precede the action. For instance, a_6^{loan} often has waiting times, i.e., 8 out of 9 weeks, since it is mostly planned together with a_5^{loan} that is conflicting, precedent action of a_6^{loan} . Moreover, a_4^{loan} always shows waiting times since its preceding action, a_3^{loan} , is always planned together, i.e., 3 out of 3 weeks.

5.2. Experiments

The presented use case demonstrates that different action plans generated by the pattern-based action engine show varying scheduling performance. The scheduling performance may depend on 1) the number of actions in a given action requirement, 2) the number of conflicts between the actions in a given action conflict, and 3) the type of conflicts. To evaluate the scheduling performance of the proposed action engine in a comprehensive manner, we design the following research questions:

- **RQ1:** How do the number of actions in the action requirement and the number of conflicts in the action conflict, affect the computation time and scheduling performance?
- **RQ2:** How do different action conflicts of the same size affect the computation time and scheduling performance of the action planner?

In the following, we explain experiments to answer the aforementioned research questions. The source code and manual for the experiments is publicly available at <https://github.com/aopm/pattern-based-action-engine.git>.

779 5.2.1. Experiments Regarding RQ1

780 To address the first research question, we establish a pool of 1,000 ac-
 781 tions and generate nine distinct action conflict scenarios, each representing
 782 a different level of conflict among action pairs. For example, the scenario
 783 denoted as “action conflict (10%)” reflects conflicts occurring in 10% of all
 784 possible action pairs. With these conflict scenarios, we then generate action
 785 plans involving various numbers of actions from different action requirements,
 786 ranging from 100 to 1,000.

787 In this experimental setup, we assume each action has a uniform duration
 788 of 1. We maintain this constant duration to control for potential variability
 789 it may introduce, enabling us to better isolate and understand the effects
 790 of our primary variables of interest: the number of actions within an action
 791 requirement and the degree of conflict between actions in an action conflict.

792 In assessing scheduling performance, we utilize makespan as our evalu-
 793 ative measure, primarily due to its direct alignment with the objectives of
 794 our proposed action planner. Moreover, makespan offers a valuable means to
 795 determine whether necessary actions are being planned and executed quickly
 796 for the enhancement of business processes.

797 Figure 16(a)-(b) shows the runtime of the action planner when planning a
 798 variable number of actions from 100 to 1,000 with the nine action conflicts.
 799 Each line in Figure 16(a) represents the runtime of planning the variable
 800 number of actions with an action conflict. For instance, the pink-colored
 801 line in Figure 16(a) describes the runtime of planning the variable number
 802 of actions with *action conflict 90%*. The experimental results show that the
 803 runtime increases quadratically to the number of actions in an action re-
 804 quirement for each action conflict. As the number of actions in an action
 805 requirement increases, the runtime difference between different action con-
 806 flicts increases. For instance, the runtime difference between *action conflict*
 807 *(90%)* and *action conflict (80%)* is around 5 seconds with 500 actions in an
 808 action requirement and around 35 seconds with 1,000 actions in an action
 809 requirement.

810 Each line in Figure 16(b) describes the runtime of planning the actions of
 811 action requirements with different action conflicts. For instance, the yellow-
 812 colored line in Figure 16(b) describes the runtime of planning 1,000 actions
 813 of an action requirement with different action conflicts. It shows that the
 814 runtime increases proportionally to the number of conflicts between actions.
 815 When the number of actions in the action requirement is less than or equal

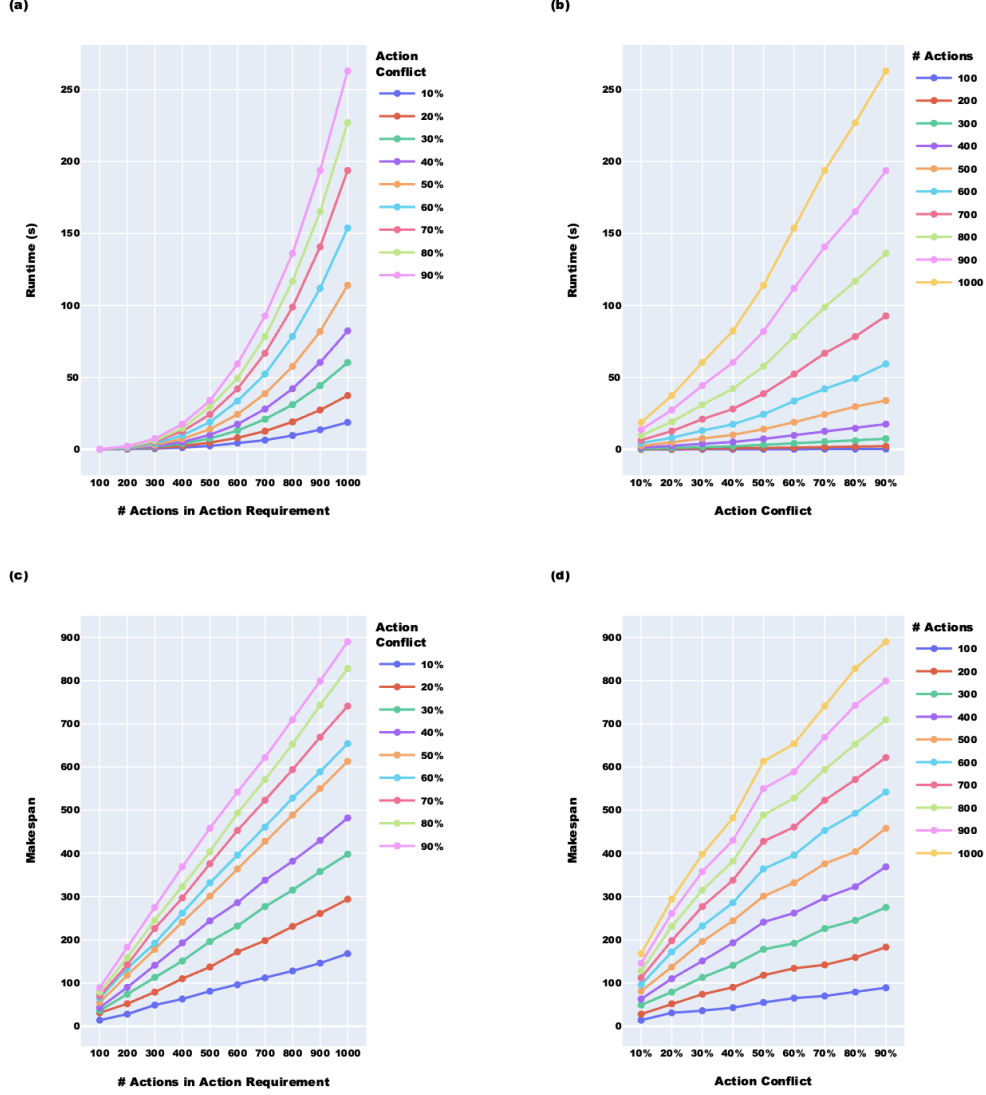


Figure 16: Experimental results to show the scalability and performance of the proposed action engine

816 to 200, almost no differences exist in the runtime.

817 Next, Figure 16(c)-(d) shows the makespan of the action planner when
818 planning a variable number of actions of different action requirements from
819 100 to 1,000 with the nine different action conflicts. Each line in Figure 16(c)

820 represents the makespan of action plans generated with the variable number
821 of actions and an action conflict of a fixed number of conflicts. For instance,
822 the pink-colored line describes the makespan of action plans generated for
823 the variable number of actions with *action conflict (90%)*. The experimental
824 result shows that the makespan increases linearly to the number of candidates
825 for each action conflict.

826 As shown in Figure 16(d), the higher the number of conflicts is, the higher
827 the makespan is. However, the makespan shows more significant increases for
828 specific changes in the number of conflicts. For instance, when action conflict
829 changes from *action conflict (40%)* to *action conflict (50%)*, the makespan
830 increases more than other changes for each number of actions in the action
831 requirement.

832 5.2.2. Experiments Regarding RQ2

833 The previous experiment considers ten action requirements and nine ac-
834 tion conflicts to evaluate the impact of the number of actions and the number
835 of conflicts on the scheduling performance and computation time. In this ex-
836 periment, we shift our focus to understanding the effects of varying conflict
837 relationships. More specifically, we aim to investigate how different conflict
838 relationships impact the planning process, even when the total number of
839 conflicts remains constant.

840 To conduct this experiment, we utilize an action requirement involving
841 500 actions. We subsequently define 99 categories of action conflicts. Each
842 category represents a specific percentage of conflicts among the actions. For
843 example, “action conflict (1%)” symbolizes conflicts in 1% of all possible
844 action pairs, while “action conflict (99%)” represents conflicts in 99% of all
845 action pairs. Within each category, we alter the conflict relationships while
846 keeping the overall number of conflicts constant.

847 Figure 17 shows the makespan of the action planner when planning the
848 500 actions under different categories of action conflicts. Each box plot show-
849 cases the makespan when planning the 500 actions with ten unique conflict
850 relationships in the respective category. For instance, the first box plot of
851 *action conflict (1%)* describes the makespans of planning 500 actions with
852 ten different action conflicts with the same total number of conflicts (i.e., 1%
853 of the action pairs have conflicts), but varying conflict relationships. The
854 experimental results show that the variance of makespans continuously in-
855 creases until a certain point around *action conflict (50%)* and incrementally
856 decreases afterward on a small scale. The variance is significantly small when

857 the number of conflicts is minimal, e.g., 1 and 2 percent of all the action pairs,
858 and the number of conflicts is substantial, e.g., 99 percent of all the action
859 pairs.

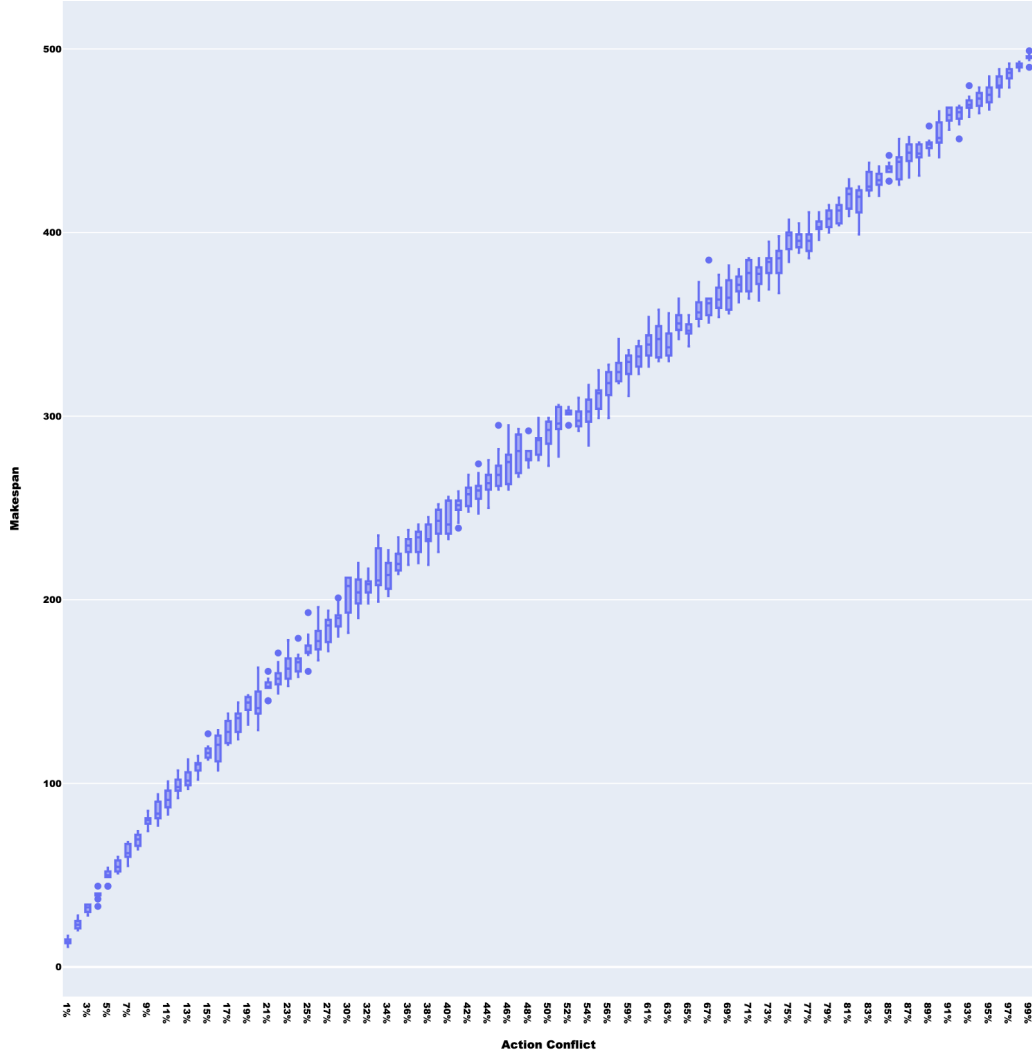


Figure 17: Experimental results showing the effect of different action conflicts of the same size on the scheduling performance of the proposed action engine

860 Next, Figure 18 shows the runtime of the same experiment. Each box plot
861 signifies the runtime while planning 500 actions with ten different conflict
862 relationships in the corresponding category. For instance, the first box plot

863 of *action conflict (1%)* describes the runtime of planning 500 actions with
864 ten different action conflicts with the same total number of conflicts (i.e.,
865 1% of the action pairs have conflicts), but varying conflict relationships.
866 Unlike the makespan, the runtime variance incrementally increases as the
867 total number of conflicts increases. However, the runtime does not show
868 significant differences when almost all the action pairs have conflicts, i.e.,
869 with *action conflict (99%)*. In some cases, the runtime is hugely higher than
870 the others. For instance, one of the action conflicts in *action conflict (75%)*
871 shows an exceptionally high runtime, even higher than the larger size of
872 action conflicts does, e.g., *action conflict (76%)* and *action conflict (78%)*.

873 6. Discussion

874 This section presents a discussion of the evaluation results, limitations,
875 and implications of this work.

876 6.1. Evaluation Results

877 The use case on the loan application process (cf. Subsection 5.1) demon-
878 strates that real-life business processes entail temporal patterns of opera-
879 tional constraints, and the proposed pattern-based action engine effectively
880 analyzes such temporal patterns and generates necessary process manage-
881 ment actions to deal with the resulting problems. First, a process analyst
882 can flexibly design action graphs using graphical notation. In the use case,
883 we designed an action graph that tackles six temporal patterns of operational
884 constraints with seven actions. Second, constraint instances can be contin-
885 uously monitored, and actions can be automatically generated based on the
886 designed action graph. In the use case, we monitored the loan application
887 process weekly and automatically generated actions for the coming days. Fi-
888 nally, the proposed action engine’s action planner optimizes the generated
889 actions’ schedule. In the use case, a conflict-free action plan of the optimal
890 makespan was successfully produced for each week.

891 As we analyzed the use case, we made noteworthy observations. First,
892 due to uncertain demands for loans and flexible terms for communications,
893 the process showed dynamic behavior each week, resulting in the occurrence
894 of various temporal patterns. Therefore, in some weeks, most of the seven
895 actions were planned, whereas only a few were planned for the other weeks.
896 This, in turn, resulted in varying makespans of the action plans in other
897 weeks. Second, although the action planner is not implemented to optimize

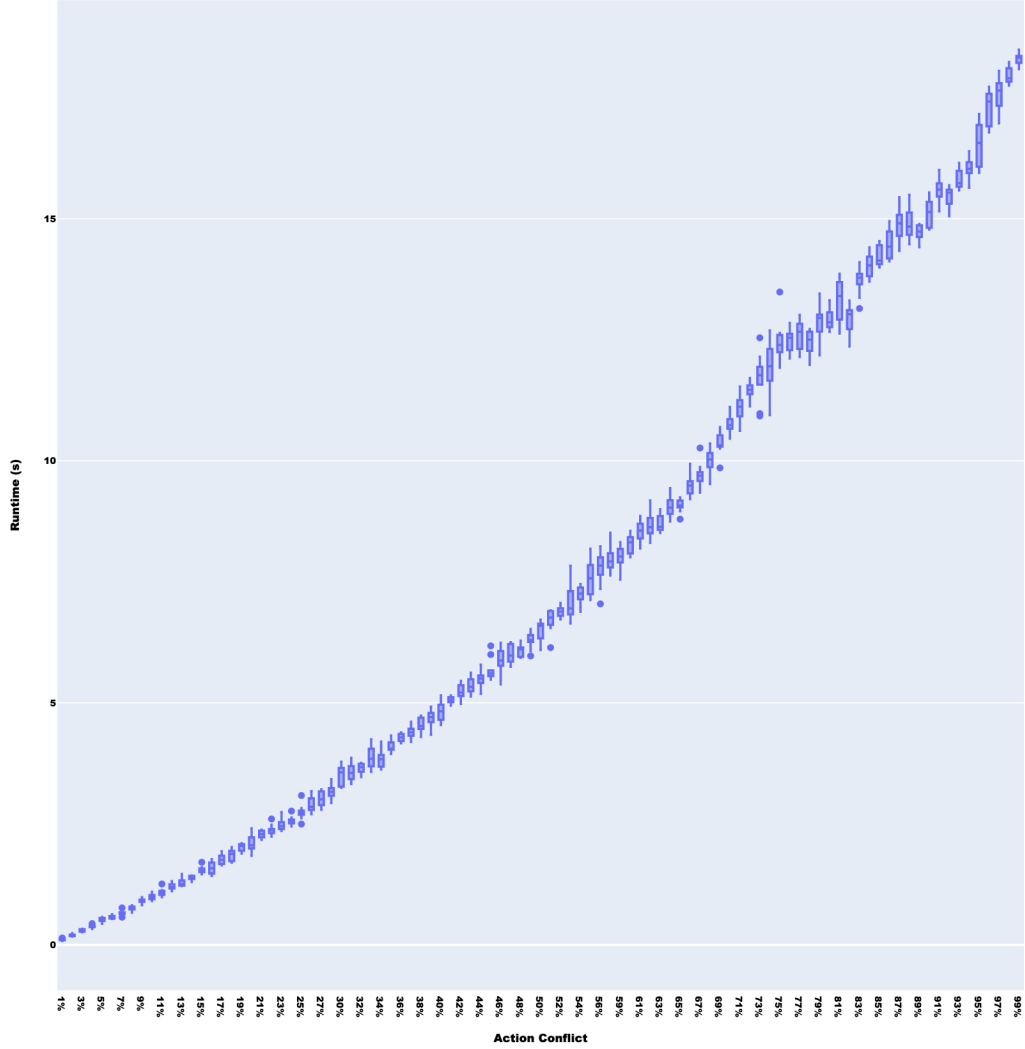


Figure 18: Experimental results showing the effect of different action conflicts of the same size on the runtime of the proposed action engine

the total waiting time and total flow time, all the action plans for the 12 weeks showed the minimum total waiting time and flow time. Third, the waiting time of an action plan results from a specific action in the action requirement. This is because the action is often planned together with its conflicting actions. For instance, a_6^{loan} is often planned together with a_5^{loan} , while a_4^{loan} is always planned together with a_3^{loan} , leading to the delay of a_6^{loan}

904 and a_4^{loan} until a_5^{loan} and a_3^{loan} are completed, respectively.

905 The experimental results (cf. Subsection 5.2) demonstrated that the run-
906 time of the proposed action planner is quadratic to the number of actions
907 in a given action requirement and proportional to the number of conflicts
908 in a given action conflict. For instance, when the number of actions in the
909 given action requirement is 1,000, and 90% of the actions have conflicts, the
910 runtime is 250 seconds. Since the number of process management actions
911 in real-life business processes does not scale to thousands and the automatic
912 generation of such actions does not occur every minute, the proposed ap-
913 proach is scalable to be deployed in real-life business processes.

914 6.2. Limitations

915 Although the use case shows the feasibility of the proposed action engine
916 in a real-life business process, it was not conducted in the organization and,
917 thus, lacks the actual application of the action plans to the process. Indeed,
918 it is essential to conduct the case study in the organization to evaluate the
919 effects of actions due to various factors [52]. For instance, diverse influencing
920 factors may arise while applying the actions, e.g., business processes are
921 affected by social factors such as organizational frictions [53]. Adding a
922 new resource to a task, in principle, increases the productivity of the task.
923 However, considering the possible organizational frictions of the action, it
924 can also negatively impact productivity.

925 Second, we abstract from defining actions generated by the action engine,
926 leaving it up to domain-specific experts. A recent development in workflow
927 automation systems is a promising solution to translating abstracted ac-
928 tions into formats the source systems can execute. For instance, Make¹ is
929 a web-based automation platform that provides a vast amount of workflow
930 automation scenarios in various information systems, e.g., SAP S/4HANA²,
931 ServiceNow³, HubSpot⁴, with visualizations for the scenarios. For instance,
932 one can connect SAP S/4HANA to create automated visual workflows, e.g.,
933 canceling a supplier invoice, deleting a credit memo request, etc.

934 Third, the action planner of the proposed action engine resolves the con-
935 flicts between the actions of an action requirement. In other words, it does

¹<https://www.make.com/>

²<https://www.sap.com>

³<https://www.servicenow.com>

⁴<https://www.hubspot.com>

not consider the action instances already being executed in the system. For instance, when applied weekly, it analyzes constraint instances in a week, generates actions to tackle the issues in the week, and produces action instances for the coming days. However, if the action instances range longer than a week, the newly generated action instances may conflict with the previous week’s action instances. The action planner needs to be extended to consider such ongoing action instances when planning the actions of action requirements.

Lastly, our proposed model produces an action plan for a fixed action requirement. As demonstrated in the evaluation, this approach may result in the start of action execution in the week following the detection of problems. While this might appear delayed, it serves a specific purpose in scenarios that do not mandate immediate intervention and are better addressed through advanced planning and subsequent execution. For instance, situations that necessitate the reassignment of resources or complex logistical coordination stand to benefit from this approach. However, there are situations that demand prompt response, and any delay could result in the issue dissipating before action is taken. To deal with such situations, it is necessary to extend the proposed approach to incorporate planning for “streaming” action requirements, enabling to plan and execute actions as soon as problems arise.

6.3. *Implications*

Our work has important implications from both academic and practical viewpoints. From an academic research viewpoint, this work provides the foundation for developing novel techniques for analyzing temporal patterns of operational constraints, i.e., operational problems. For instance, one can extend action graphs to restrict the time of evaluating the presence of temporal pattern trees, e.g., if the temporal pattern tree occurs in the last three days of the given constraint instances. Moreover, the generation of action requirements can be enhanced by extending action graphs. For instance, one can add the due date constraint to action graphs, e.g., the corresponding action should be executed in three days.

Furthermore, this work provides the formal architecture upon which novel techniques for generating action requirements and planning action instances can be developed. In this work, we use a brute-force approach to implement the generation of action requirements. Instead, other approaches with different analytical advantages or computational efficiencies can be deployed to implement the action generation. Moreover, our implementation of action

973 planners optimizes the makespan of action plans. Depending on various pur-
974 poses, one can define different objectives, e.g., optimizing the total waiting
975 time of action plans, and develop different algorithms to achieve the objec-
976 tives. In addition, the action planner can be extended to resolve different
977 types of conflicts, e.g., effect-based conflicts and dependency-based conflicts.

978 Besides, in real-life business processes, it is common to observe variations
979 and changes in process behavior, i.e., concept drifts [54]. In order to imple-
980 ment effective techniques for action engines, we need to consider such concept
981 drifts. In particular, the design of action graphs should be adapted accord-
982 ingly to the concept drift so that they remain relevant in varying situations.
983 To this end, opportunities exist for fellow scholars to connect the detection,
984 characterization, and explanation of concept drifts in business processes to
985 the dynamic and adaptive design of action graphs.

986 Furthermore, the proposed action engine can be extended by distinguish-
987 ing between observed and predicted constraint instances. The input for the
988 action engine is a collection of constraint instances that may be observed us-
989 ing backward-looking techniques and predicted using forward-looking tech-
990 niques. For example, an approach suggested in [55] predicts the performance
991 of business processes in a given time window and can be used to produce
992 such predicted constraint instances. The significance of these forecasted op-
993 erational constraints may vary, potentially influenced by factors like the like-
994 lihood or precision of the prediction. To optimize its effectiveness, an action
995 engine can be designed to assign varying weights to constraint instances,
996 thereby prioritizing those constraint patterns bearing higher weights.

997 From a practical viewpoint, this work allows organizations to continuously
998 analyze temporal patterns of operational constraints detected by various pro-
999 cess monitoring techniques and automatically connect the resulting insights
1000 into concrete forms of actions. The automated actions tackle the temporal
1001 patterns of operational constraints existing in organizations' business pro-
1002 cesses and contribute to improving the process. This allows organizations
1003 to maintain competitive advantages in fast-evolving and dynamic business
1004 environments. Moreover, the proposed action engine involves practitioners
1005 in designing action graphs. This allows them to leverage domain knowledge
1006 to define temporal patterns of operational constraints and elicit actions to
1007 deal with the temporal patterns, facilitating the adoption of the technique.

7. Conclusion

Existing approaches in action-oriented process mining consider operational constraints as temporally independent and point-based data, generating actions based on relatively simple occurrence rules (cf. Figure 1). Moreover, they do not consider possible conflicts between actions that are prevalent in reality when triggering the generated actions. This paper tackles this research gap by proposing a pattern-based action engine to analyze complex temporal patterns of operational constraints and produce conflict-free actions by considering possible conflicts between the actions.

Existing approaches in action-oriented process mining view operational constraints as temporally independent and point-based data and do not consider the potential conflicts between actions. This paper aims to address this research gap by introducing a pattern-based action engine. This action engine is designed to analyze the intricate temporal patterns of operational constraints and produce conflict-free actions, while considering the potential conflicts that might arise between actions.

The proposed action engine consists of three phases. First, domain experts design *action graphs* that are graphical notations consisting of two types of nodes, i.e., temporal pattern trees and actions, and edges connecting the nodes. Using the graphical notations, one can visually define action graphs. Second, the action generator evaluates if the temporal patterns specified in the action graphs occur in the constraint instances of the time window and generates action requirements that describe which actions need to be executed for how long. Finally, the action planner produces an action plan such that no conflicts occur during the execution of the actions.

We demonstrated a use case using the data of a real-life loan application process of a Dutch financial institute to evaluate the feasibility of the proposed action engine in real-life business processes. In the use case, we designed an action graph to tackle the redundant operational costs caused by the cancellation of applications. Afterward, we applied the action engine to the loan application process with the action graphs for 12 weeks. The use case demonstrates that the proposed pattern-based action engine effectively analyzes temporal patterns of operational constraints existing in the real-life business process and automatically generates actions to tackle the risks caused by the patterns. Furthermore, we conducted experiments to evaluate the scalability and performance of the proposed action engine.

In future work, we plan to apply the proposed action engine in real-life

business processes. This application will enable us to collaborate with domain experts to derive actual issues present in the process and to exercise full control in addressing these problems with the suggested actions. Subsequently, this application will also allow us to assess the real-world efficacy of the actions generated by our proposed action engine in resolving operational issues. For example, we could measure the impact of these actions by comparing Key Performance Indicators (KPIs) before and after the implementation of the proposed actions. Moreover, we plan to extend the proposed action engine's implementation to generate executable actions by incorporating workflow automation systems. Another interesting direction for future work is to extend the action planner such that the plans are optimized to maximize the impact of actions. To that end, we may predict the impact of generated actions and extend the objective of action planners to maximize the overall impact of the actions.

References

- [1] M. Hammer, Reengineering work: Don't automate, obliterate., Harvard business review 68 (4) (1990) 104–112.
- [2] H. Reijers, S. Liman Mansar, Best practices in business process redesign: an overview and qualitative evaluation of successful redesign heuristics, Omega 33 (4) (2005) 283–306. doi:<https://doi.org/10.1016/j.omega.2004.04.012>. URL <https://www.sciencedirect.com/science/article/pii/S0305048304000854>
- [3] T. Pyzdek, P. Keller, Six sigma handbook, McGraw-Hill Education, 2014.
- [4] P. J. Martínez-Jurado, J. Moyano-Fuentes, Lean management, supply chain management and sustainability: A literature review, Journal of Cleaner Production 85 (2014) 134–150, special Volume: Making Progress Towards More Sustainable Societies through Lean and Green Initiatives. doi:<https://doi.org/10.1016/j.jclepro.2013.09.042>. URL <https://www.sciencedirect.com/science/article/pii/S0959652613006550>
- [5] R. J. B. Vanwersch, I. T. P. Vanderfeesten, E. Rietzschel, H. A. Reijers, Improving business processes: Does anybody have an idea?, in:

- 1079 H. R. Motahari-Nezhad, J. Recker, M. Weidlich (Eds.), Business Pro-
 1080 cess Management - 13th International Conference, BPM 2015, Inns-
 1081 bruck, Austria, August 31 - September 3, 2015, Proceedings, Vol.
 1082 9253 of Lecture Notes in Computer Science, Springer, 2015, pp. 3–18.
 1083 doi:10.1007/978-3-319-23063-4_1.
 1084 URL https://doi.org/10.1007/978-3-319-23063-4_1
- 1085 [6] I. Beerepoot, C. Di Ciccio, H. A. Reijers, S. Rinderle-Ma, W. Bandara,
 1086 A. Burattin, D. Calvanese, T. Chen, I. Cohen, B. Depaire, G. Di Fed-
 1087 erico, M. Dumas, C. van Dun, T. Fehrer, D. A. Fischer, A. Gal, M. Indul-
 1088 ska, V. Isahagian, C. Klinkmüller, W. Kratsch, H. Leopold, A. Van Looy,
 1089 H. Lopez, S. Lukumbuzya, J. Mendling, L. Meyers, L. Moder, M. Mon-
 1090 tali, V. Muthusamy, M. Reichert, Y. Rizk, M. Rosemann, M. Röglinger,
 1091 S. Sadiq, R. Seiger, T. Slaats, M. Simkus, I. A. Someh, B. Weber, I. We-
 1092 ber, M. Weske, F. Zerbato, The biggest business process management
 1093 problems to solve before we die, *Computers in Industry* 146 (2023)
 1094 103837. doi:<https://doi.org/10.1016/j.compind.2022.103837>.
 1095 URL [https://www.sciencedirect.com/science/article/pii/](https://www.sciencedirect.com/science/article/pii/S0166361522002330)
 1096 [S0166361522002330](https://www.sciencedirect.com/science/article/pii/S0166361522002330)
- 1097 [7] W. M. P. van der Aalst, J. Carmona (Eds.), *Process Mining Handbook*,
 1098 Vol. 448 of *Lecture Notes in Business Information Processing*, Springer,
 1099 2022. doi:10.1007/978-3-031-08848-3.
 1100 URL <https://doi.org/10.1007/978-3-031-08848-3>
- 1101 [8] W. M. P. van der Aalst, *Process Mining - Data Science in Action*, Second
 1102 Edition, Springer, 2016. doi:10.1007/978-3-662-49851-4.
 1103 URL <https://doi.org/10.1007/978-3-662-49851-4>
- 1104 [9] G. Park, W. M. P. van der Aalst, Action-oriented process mining: bridg-
 1105 ing the gap between insights and actions, *Progress in Artificial Intelli-*
 1106 *gence* (2022) 1–22.
- 1107 [10] P. Badakhshan, G. Bernhart, J. Geyer-Klingenberg, J. Nakladal,
 1108 S. Schenk, T. Vogelgesang, The Action Engine – Turning Process In-
 1109 sights into Action, in: 2019 ICPM Demo Track, *ceur-ws*, Aachen, Ger-
 1110 many, 2019, pp. 28–31.
 1111 URL <http://ceur-ws.org/Vol-2374/paper8.pdf>

- 1112 [11] B. Ur, M. P. Y. Ho, S. Brawner, J. Lee, S. Mennicken, N. Picard,
1113 D. Schulze, M. L. Littman, Trigger-action programming in the wild: An
1114 analysis of 200, 000 IFTTT recipes, in: J. Kaye, A. Druin, C. Lampe,
1115 D. Morris, J. P. Hourcade (Eds.), Proceedings of the 2016 CHI Con-
1116 ference on Human Factors in Computing Systems, San Jose, CA, USA,
1117 May 7-12, 2016, ACM, 2016, pp. 3227–3231. doi:10.1145/2858036.
1118 2858556.
1119 URL <https://doi.org/10.1145/2858036.2858556>
- 1120 [12] M. Dumas, M. L. Rosa, J. Mendling, H. A. Reijers, Fundamen-
1121 tals of Business Process Management, Springer, 2013. doi:10.1007/
1122 978-3-642-33143-5.
1123 URL <https://doi.org/10.1007/978-3-642-33143-5>
- 1124 [13] S. L. Mansar, H. A. Reijers, Best practices in business process redesign:
1125 use and impact, Bus. Process. Manag. J. 13 (2007) 193–213.
- 1126 [14] T. Fehrer, D. A. Fischer, S. J. J. Leemans, M. Röglinger, M. T. Wynn,
1127 An assisted approach to business process redesign, Decis. Support Syst.
1128 156 (2022) 113749. doi:10.1016/j.dss.2022.113749.
1129 URL <https://doi.org/10.1016/j.dss.2022.113749>
- 1130 [15] K. Kubrak, F. Milani, A. Nolte, M. Dumas, Prescriptive process moni-
1131 toring: *Quo vadis?*, PeerJ Comput. Sci. 8 (2022) e1097. doi:10.7717/
1132 peerj-cs.1097.
1133 URL <https://doi.org/10.7717/peerj-cs.1097>
- 1134 [16] R. Conforti, M. de Leoni, M. L. Rosa, W. M. P. van der Aalst, A. H. M.
1135 ter Hofstede, A recommendation system for predicting risks across mul-
1136 tiple business process instances, Decis. Support Syst. 69 (2015) 1–19.
1137 doi:10.1016/j.dss.2014.10.006.
1138 URL <https://doi.org/10.1016/j.dss.2014.10.006>
- 1139 [17] Z. D. Bozorgi, I. Teinemaa, M. Dumas, M. L. Rosa, A. Polyvyanyy,
1140 Prescriptive process monitoring for cost-aware cycle time reduction,
1141 in: C. D. Ciccio, C. D. Francescomarino, P. Soffer (Eds.), 3rd In-
1142 ternational Conference on Process Mining, ICPM 2021, Eindhoven,
1143 The Netherlands, October 31 - Nov. 4, 2021, IEEE, 2021, pp. 96–103.
1144 doi:10.1109/ICPM53251.2021.9576853.
1145 URL <https://doi.org/10.1109/ICPM53251.2021.9576853>

- 1146 [18] S. Weinzierl, S. Dunzer, S. Zilker, M. Matzner, Prescriptive business
1147 process monitoring for recommending next best actions, in: D. Fahland,
1148 C. Ghidini, J. Becker, M. Dumas (Eds.), Business Process Management
1149 Forum - BPM Forum 2020, Seville, Spain, September 13-18, 2020, Pro-
1150 ceedings, Vol. 392 of Lecture Notes in Business Information Processing,
1151 Springer, 2020, pp. 193–209. doi:10.1007/978-3-030-58638-6_12.
1152 URL https://doi.org/10.1007/978-3-030-58638-6_12
- 1153 [19] S. A. Fahrenkrog-Petersen, N. Tax, I. Teinemaa, M. Dumas, M. de Leoni,
1154 F. M. Maggi, M. Weidlich, Fire now, fire later: alarm-based systems for
1155 prescriptive process monitoring, *Knowl. Inf. Syst.* 64 (2) (2022) 559–587.
1156 doi:10.1007/s10115-021-01633-w.
1157 URL <https://doi.org/10.1007/s10115-021-01633-w>
- 1158 [20] R. Villafane, K. A. Hua, D. Tran, B. Maulik, Knowledge discovery from
1159 series of interval events, *J. Intell. Inf. Syst.* 15 (1) (2000) 71–89. doi:
1160 10.1023/A:1008781812242.
1161 URL <https://doi.org/10.1023/A:1008781812242>
- 1162 [21] P. Kam, A. W. Fu, Discovering temporal patterns for interval-based
1163 events, in: Y. Kambayashi, M. K. Mohania, A. M. Tjoa (Eds.), Data
1164 Warehousing and Knowledge Discovery, Second International Confer-
1165 ence, DaWaK 2000, London, UK, September 4-6, 2000, Proceedings,
1166 Vol. 1874 of Lecture Notes in Computer Science, Springer, 2000, pp.
1167 317–326. doi:10.1007/3-540-44466-1_32.
1168 URL https://doi.org/10.1007/3-540-44466-1_32
- 1169 [22] J. F. Allen, Maintaining knowledge about temporal intervals, *Commun.*
1170 *ACM* 26 (11) (1983) 832–843. doi:10.1145/182.358434.
1171 URL <https://doi.org/10.1145/182.358434>
- 1172 [23] F. Höppner, Learning temporal rules from state sequences, in: *IJCAI*
1173 *Workshop on Learning from Temporal and Spatial Data*, Vol. 25, Cite-
1174 seer, 2001.
- 1175 [24] P. Papapetrou, G. Kollis, S. Sclaroff, D. Gunopulos, Mining frequent
1176 arrangements of temporal intervals, *Knowl. Inf. Syst.* 21 (2) (2009) 133–
1177 171. doi:10.1007/s10115-009-0196-0.
1178 URL <https://doi.org/10.1007/s10115-009-0196-0>

- 1179 [25] J. Ayres, J. Flannick, J. Gehrke, T. Yiu, Sequential pattern mining using
1180 a bitmap representation, in: Proceedings of the Eighth ACM SIGKDD
1181 International Conference on Knowledge Discovery and Data Mining,
1182 July 23-26, 2002, Edmonton, Alberta, Canada, ACM, 2002, pp. 429–
1183 435. doi:10.1145/775047.775109.
1184 URL <https://doi.org/10.1145/775047.775109>
- 1185 [26] R. Moskovitch, Y. Shahar, Fast time intervals mining using the tran-
1186 sitivity of temporal relations, *Knowl. Inf. Syst.* 42 (1) (2015) 21–48.
1187 doi:10.1007/s10115-013-0707-x.
1188 URL <https://doi.org/10.1007/s10115-013-0707-x>
- 1189 [27] O. D. Harel, R. Moskovitch, Complete closed time intervals-related pat-
1190 terns mining, in: Thirty-Fifth AAAI Conference on Artificial Intelli-
1191 gence, AAAI 2021, Thirty-Third Conference on Innovative Applications
1192 of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Edu-
1193 cational Advances in Artificial Intelligence, EAAI 2021, Virtual Event,
1194 February 2-9, 2021, AAAI Press, 2021, pp. 4098–4105.
1195 URL <https://ojs.aaai.org/index.php/AAAI/article/view/16531>
- 1196 [28] S.-Y. Wu, Y.-L. Chen, Mining nonambiguous temporal patterns for
1197 interval-based events, *IEEE Transactions on Knowledge and Data En-*
1198 *gineering* 19 (6) (2007) 742–758. doi:10.1109/TKDE.2007.190613.
- 1199 [29] J. Pei, J. Han, B. Mortazavi-Asl, J. Wang, H. Pinto, Q. Chen, U. Dayal,
1200 M. Hsu, Mining sequential patterns by pattern-growth: The prefixspan
1201 approach, *IEEE Trans. Knowl. Data Eng.* 16 (11) (2004) 1424–1440.
1202 doi:10.1109/TKDE.2004.77.
1203 URL <https://doi.org/10.1109/TKDE.2004.77>
- 1204 [30] D. Schuster, L. Schade, S. J. van Zelst, W. M. P. van der Aalst, Vi-
1205 sualizing trace variants from partially ordered event data, in: *Process*
1206 *Mining Workshops*, Springer, 2022, pp. 34–46.
- 1207 [31] T. Asai, K. Abe, S. Kawasoe, H. Sakamoto, H. Arimura, S. Arikawa,
1208 Efficient substructure discovery from large semi-structured data, *IEICE*
1209 *TRANSACTIONS on Information and Systems* 87 (12) (2004).
- 1210 [32] Y. Chi, Y. Yang, Y. Xia, R. R. Muntz, Cmtreeminer: Mining both closed
1211 and maximal frequent subtrees, in: *Advances in Knowledge Discovery*

- 1212 and Data Mining: 8th Pacific-Asia Conference, PAKDD 2004, Sydney,
1213 Australia, May 26-28, 2004. Proceedings 8, Springer, 2004.
- 1214 [33] D. Schuster, S. J. van Zelst, W. M. P. van der Aalst, Cortado: A ded-
1215 icated process mining tool for interactive process discovery, *SoftwareX*
1216 22 (2023) 101373. doi:10.1016/j.softx.2023.101373.
1217 URL <https://doi.org/10.1016/j.softx.2023.101373>
- 1218 [34] M. Dumas, W. M. P. van der Aalst, A. H. M. ter Hofstede (Eds.),
1219 Process-Aware Information Systems: Bridging People and Software
1220 Through Process Technology, Wiley, 2005. doi:10.1002/0471741442.
1221 URL <https://doi.org/10.1002/0471741442>
- 1222 [35] W. M. P. van der Aalst, The application of petri nets to workflow
1223 management, *J. Circuits Syst. Comput.* 8 (1) (1998) 21–66. doi:
1224 10.1142/S0218126698000043.
1225 URL <https://doi.org/10.1142/S0218126698000043>
- 1226 [36] A. Grosskopf, G. Decker, M. Weske, The process: business process mod-
1227 eling using BPMN, Meghan Kiffer Press, 2009.
- 1228 [37] H. Schonenberg, R. Mans, N. Russell, N. Mulyar, W. M. P. van der Aalst,
1229 Process flexibility: A survey of contemporary approaches, in: J. L. G.
1230 Dietz, A. Albani, J. Barjis (Eds.), *Advances in Enterprise Engineering*
1231 I, 4th International Workshop CIAO! and 4th International Workshop
1232 EOMAS, held at CAiSE 2008, Montpellier, France, June 16-17, 2008.
1233 Proceedings, Vol. 10 of Lecture Notes in Business Information Process-
1234 ing, Springer, 2008, pp. 16–30. doi:10.1007/978-3-540-68644-6_2.
1235 URL https://doi.org/10.1007/978-3-540-68644-6_2
- 1236 [38] M. L. Rosa, M. Dumas, A. H. M. ter Hofstede, Modeling business pro-
1237 cess variability for design-time configuration, in: J. Cardoso, W. M. P.
1238 van der Aalst (Eds.), *Handbook of Research on Business Process Model-*
1239 *ing*, IGI Global, 2009, pp. 204–228. doi:10.4018/978-1-60566-288-6.
1240 ch009.
1241 URL <https://doi.org/10.4018/978-1-60566-288-6.ch009>
- 1242 [39] M. Rosemann, W. M. P. van der Aalst, A configurable reference mod-
1243 elling language, *Inf. Syst.* 32 (1) (2007) 1–23. doi:10.1016/j.is.2005.

- 1244 05.003.
1245 URL <https://doi.org/10.1016/j.is.2005.05.003>
- 1246 [40] F. Gottschalk, Configurable process models, Ph.D. thesis, Industrial En-
1247 gineering and Innovation Sciences, proefschrift. (2009). doi:10.6100/
1248 IR653896.
- 1249 [41] M. Peleg, J. Somekh, D. Dori, A methodology for eliciting and modeling
1250 exceptions, J. Biomed. Informatics 42 (4) (2009) 736–747. doi:10.1016/
1251 j.jbi.2009.05.003.
1252 URL <https://doi.org/10.1016/j.jbi.2009.05.003>
- 1253 [42] M. Reichert, P. Dadam, Adept_{flex}-supporting dynamic changes of work-
1254 flows without losing control, J. Intell. Inf. Syst. 10 (2) (1998) 93–129.
1255 doi:10.1023/A:1008604709862.
1256 URL <https://doi.org/10.1023/A:1008604709862>
- 1257 [43] B. Weber, M. Reichert, S. Rinderle-Ma, Change patterns and change
1258 support features - enhancing flexibility in process-aware information sys-
1259 tems, Data Knowl. Eng. 66 (3) (2008) 438–466. doi:10.1016/j.datak.
1260 2008.05.001.
1261 URL <https://doi.org/10.1016/j.datak.2008.05.001>
- 1262 [44] J. Buckley, T. Mens, M. Zenger, A. Rashid, G. Kniesel, Towards a
1263 taxonomy of software change 17 (5) 309–332. doi:10.1002/smr.319.
1264 URL <https://onlinelibrary.wiley.com/doi/10.1002/smr.319>
- 1265 [45] N. Chapin, J. E. Hale, K. M. Khan, J. F. Ramil, W.-G. Tan, Types
1266 of software evolution and software maintenance 13 (1) 3–30. doi:10.
1267 1002/smr.220.
1268 URL <https://onlinelibrary.wiley.com/doi/10.1002/smr.220>
- 1269 [46] M. Wermelinger, G. Koutsoukos, H. Lourenço, R. Avillez, J. Gouveia,
1270 L. F. Andrade, J. L. Fiadeiro, Enhancing dependability through flexible
1271 adaptation to changing requirements, in: R. de Lemos, C. Gacek, A. B.
1272 Romanovsky (Eds.), Architecting Dependable Systems II - [the book
1273 is a result of the ICSE 2003 Workshop on Software Architectures for
1274 Dependable Systems], Vol. 3069 of Lecture Notes in Computer Science,
1275 Springer, 2003, pp. 3–24. doi:10.1007/978-3-540-25939-8_1.
1276 URL https://doi.org/10.1007/978-3-540-25939-8_1

- 1277 [47] A. Charfi, M. Mezini, Hybrid web service composition: business pro-
 1278 cesses meet business rules, in: M. Aiello, M. Aoyama, F. Curbera, M. P.
 1279 Papazoglou (Eds.), Service-Oriented Computing - ICSOC 2004, Second
 1280 International Conference, New York, NY, USA, November 15-19, 2004,
 1281 Proceedings, ACM, 2004, pp. 30–38. doi:10.1145/1035167.1035173.
 1282 URL <https://doi.org/10.1145/1035167.1035173>
- 1283 [48] M. L. Pinedo, Scheduling, Vol. 29, Springer, 2012.
- 1284 [49] T. Cheng, C. Sin, A state-of-the-art review of parallel-machine schedul-
 1285 ing research, European Journal of Operational Research 47 (3) (1990)
 1286 271–292. doi:[https://doi.org/10.1016/0377-2217\(90\)90215-W](https://doi.org/10.1016/0377-2217(90)90215-W).
 1287 URL [https://www.sciencedirect.com/science/article/pii/](https://www.sciencedirect.com/science/article/pii/037722179090215W)
 1288 [037722179090215W](https://www.sciencedirect.com/science/article/pii/037722179090215W)
- 1289 [50] B. van Dongen, Bpi challenge 2017 (2017). doi:10.4121/UUID:
 1290 5F3067DF-F10B-45DA-B98B-86AE4C7A310B.
 1291 URL https://data.4tu.nl/articles/_/12696884/1
- 1292 [51] W. M. P. van der Aalst, A. Berti, Discovering object-centric petri
 1293 nets, Fundam. Informaticae 175 (1-4) (2020) 1–40. doi:10.3233/
 1294 FI-2020-1946.
 1295 URL <https://doi.org/10.3233/FI-2020-1946>
- 1296 [52] M. Dees, M. de Leoni, W. M. P. van der Aalst, H. A. Reijers, What
 1297 if process predictions are not followed by good recommendations?, in:
 1298 J. vom Brocke, J. Mendling, M. Rosemann (Eds.), Proceedings of the
 1299 Industry Forum at BPM 2019 co-located with 17th International Con-
 1300 ference on Business Process Management (BPM 2019), Vienna, Austria,
 1301 September 1-6, 2019, Vol. 2428 of CEUR Workshop Proceedings, CEUR-
 1302 WS.org, 2019, pp. 61–72.
 1303 URL <http://ceur-ws.org/Vol-2428/paper6.pdf>
- 1304 [53] M. de Leoni, M. Dees, L. Reulink, Design and evaluation of a process-
 1305 aware recommender system based on prescriptive analytics, in: B. F. van
 1306 Dongen, M. Montali, M. T. Wynn (Eds.), 2nd International Conference
 1307 on Process Mining, ICPM 2020, Padua, Italy, October 4-9, 2020, IEEE,
 1308 2020, pp. 9–16. doi:10.1109/ICPM49681.2020.00013.
 1309 URL <https://doi.org/10.1109/ICPM49681.2020.00013>

- 1310 [54] J. N. Adams, S. J. van Zelst, T. Rose, W. M. van der Aalst, Explainable
1311 concept drift in process mining, *Information Systems* 114 (2023) 102177.
1312 doi:<https://doi.org/10.1016/j.is.2023.102177>.
1313 URL [https://www.sciencedirect.com/science/article/pii/](https://www.sciencedirect.com/science/article/pii/S0306437923000133)
1314 [S0306437923000133](https://www.sciencedirect.com/science/article/pii/S0306437923000133)
- 1315 [55] G. Park, M. Song, Predicting performances in business processes using
1316 deep neural networks, *Decis. Support Syst.* 129 (2020). doi:[10.1016/](https://doi.org/10.1016/j.dss.2019.113191)
1317 [j.dss.2019.113191](https://doi.org/10.1016/j.dss.2019.113191).
1318 URL <https://doi.org/10.1016/j.dss.2019.113191>