Augmenting Structural Knowledge on Process Models to Improve Explainability

Tolga Tel
Department of Computer Science
Goethe University Frankfurt
Frankfurt am Main, Germany
0009-0005-5012-8250

Hoai Vy Nguyen
Department of Computer Science
Goethe University Frankfurt
Frankfurt am Main, Germany
0009-0006-0434-4742

Mirjam Minor
Department of Computer Science
Goethe University Frankfurt
Frankfurt am Main, Germany
0000-0002-6592-631X

***Abstract*—In many real-world examples, bad modeling practices reduce the interpretability of business process models for humans and systems. This problem is notable while using Generative AI to explain business processes. Previous work includes different approaches to describe business process models in a complete and comprehensible manner. However, the order of explanations element by element is perceived as tiring and has only achieved a low user acceptance. This paper explores the augmentation of business process models, by reconstructing a block-orientation to derive a more sophisticated explanation order. Experiments with process model experts were conducted to verify the derived order of elements and to highlight the feasibility of our novel augmentation method.**

***Index Terms*—Structural Knowledge, Explainability, Generative AI, Process Models**

I. Introduction

A knowledge graph (KG) describes real world entities from various domains and their interrelations, organized in a graph [1]. KGs are an effective way to represent knowledge in a structured, easily accessible form.

Since tasks are entities and sequence flows can be seen as a preference relation between tasks, business process models fit this definition of a knowledge graph. There Business Process Model and Notation (BPMN) has become the standard for visualizing and communicating business processes. Weske claims that business processes are "essential to understanding how companies operate" [2]. However, many application domain experts and other users lack knowledge on process modeling languages. They are struggling to understand the graphical process models.

Our method, which we term guided generation, addresses the challenge of LLM reliability in specific tasks. The generation process is guided by an intermediate tree structure that describes structural properties of the business process model. The paper is an extension of previous work [3] in which LLMs are used to generate descriptive texts explaining business processes. The important change in comparison to past work was the division of BPMN models into smaller subgraphs, to reduce hallucinations and increase recall of the generated descriptions. This approach also bypasses the limits of context windows. In the end, the generated descriptions are merged to get a description of the complete BPMN model.

This approach resulted in a new, open challenge: how to order the subgraphs so the descriptions generated by the LLM are presented in an understandable sequence. This ordering is now crucial because the LLM only provides descriptions for the individual subgraphs without deciding the order of these descriptions.

In this work, we introduce an algorithm to compute an order to explain BPMN models. The algorithm augments a process model by structural knowledge recorded as an intermediate tree structure. The order of explanation is directly derived from this tree. To verify our solution, we conducted an experiment with 30 human experts. With BPMN models of different sizes, we aim to observe the difference between human experts and our algorithm.

Generative AI models have achieved remarkable success in creating novel content, but they often struggle with consistency and adherence to complex real-world domains. To move towards reliable and controllable generation, explicitly incorporating inherent relationships, like hierarchies or graphs, is essential. Our research leverages structural knowledge to enhance Generative AI methodologies.

The remainder of this paper is organized as follows. Sec. II introduces some preliminaries that form the background of the paper. In Sec. III, related work is discussed. The novel augmentation method is described in Sec. IV and experimentally evaluated in Sec. V. In Sec. VI, a conclusion is drawn.

II. Background

A. *Workflow management*

The consortium OMG has established the standard BPMN [4] for modeling languages. BPMN offers a standardized graphical notation for visualizing business processes. Its primary goal is to bridge the gap between business and technical users, providing a notation that is both understandable to business analysts and precise enough for technical developers. This flowchart-like notation empowers stakeholders to design, manage, and realize business processes efficiently. Its independence from specific implementation environments ensures flexibility and adaptability. An example for a small BPMN model can be found in Figure 1.

The algorithms presented in this work operate on simplified BPMN models. Based on this simplification, we introduce

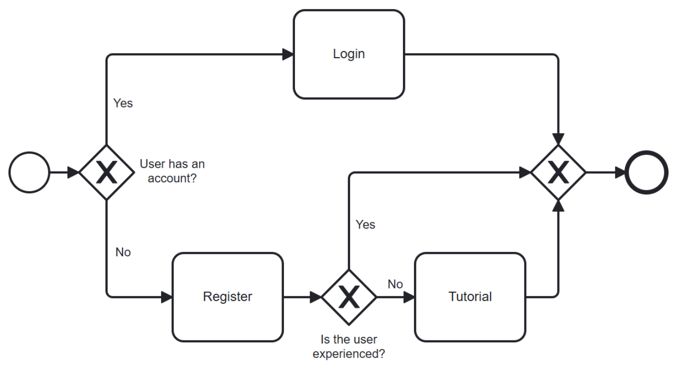

Fig. 1. Example BPMN model

the workflow graph serving as the foundation for the algorithmic processing. A workflow graph is a directed graph $G = (N, E, Start, End)$, where N is the set of nodes, $E \subseteq N \times N$ is the set of edges, and $Start, End \in N$ are distinguished nodes. Each tuple $(n_1, n_2) \in E$ is a directed edge from node n_1 to node n_2 representing a sequence flow.

The set of nodes N is defined as $N := \{Start, End\} \cup T \cup G$, where *Start* is the unique start node with no predecessors and exactly one successor, *End* is the unique end event with exactly one predecessor and no successors, T is the set of tasks, where each task $t \in T$ has exactly one predecessor and one successor, G is the set of gateways, which can either be a split gateway (multiple successors), a join gateway (multiple predecessors), or both. Every node $n \in N$ lies on at least one path from *Start* to *End*.

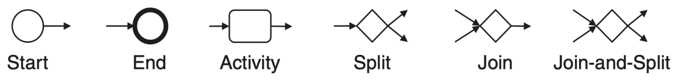

Fig. 2. Types of nodes and their representations. Reproduced from Choi et al. [5]

B. Structural properties

In a workflow graph $G = (N, E, Start, End)$ the set of dominators of a node $n \in N$ is denoted by $Dom(n)$. A node $d \in N$ is said to dominate n if every path from *Start* to n passes through d. By definition, every node dominates itself.

Similarly, the set of post-dominators of a node $n \in N$ is denoted by $Pdom(n)$. A node $p \in N$ is set to post-dominate n if every path from n to *End* passes through p.

In a workflow graph G, a single-entry single-exit (SESE) region $SESE(s, t)$ is a weakly connected subgraph of G spanned by exactly one entry $s \in N$ and exactly one exit $t \in N$, plus its interior nodes, which are dominated by s, post-dominated by t, and only connected to nodes in $SESE(s, t)$.

The relation between two SESE regions A and B is either a nesting relation, where $N_A \subset N_B$ or $N_B \subset N_A$, or a disjoint relation, where $N_A \cap N_B = \emptyset$. A SESE region is canonical if sharing any of its interior nodes with other SESE regions is equivalent to being a nesting relation. A workflow graph G is structured if for every split gateway, there is a corresponding join gateway, and vice versa, such that the two gateways bound a canonical SESE region. A rigid component contains a gateway, either a split or a join, which does not have a corresponding gateway to define a canonical SESE region. A workflow graph is structured if and only if it contains no rigid components.

Canonical SESE regions can be classified as one of following categories.

– *Singleton*: A single task node, which is bounded by itself.
– *Sequence*: A maximal, non-cyclic sequence of SESE regions, where each region is connected to exactly one other via a single edge or a shared boundary node.
– *Bond*: For a pair of distinct gateways, the entry s and the exit t, a bond is a maximal set of at least two paths starting from s and ending with t, each connected to others only at s and t. A bond contains at least one interior node $n \notin \{s, t\}$.
– *Loop*: For a pair of distinct gateways, the entry s and the exit t, a loop is a SESE region with at least one path from s to t and at least one path from t to s, each connected to others only at s and t. A loop contains at least one interior node $n \notin \{s, t\}$.
– *Rigid*: A structure that does not conform to any of the above categories. A rigid is also bounded by two gateways, and contains at least one interior node.

A *block* is a canonical SESE region of G, which is not a rigid.

A set of all canonical SESE regions of a workflow graph G is called the refined process structure tree (RPST) decomposition of G. The corresponding parse tree is called the *RPST* of G, i.e., the tree of all canonical SESE regions such that the parent of an SESE region R is the smallest SESE region that contains R [6].

III. Related work

A particularly relevant line of work addresses structural decomposition of BPMN models to enable textualization. The RPST is a widely adopted technique for decomposing process models into hierarchically nested SESE regions. RPSTs facilitate both structured visualization and textual interpretation of process models by imposing a well-defined ordering.

Notably, the method proposed by Leopold et al. [7] shares the same objective as our work – leveraging RPSTs to produce a linearized representation of BPMN models. However, their approach handles rigids by transforming them into Petri nets and generating a representative execution trace for explanation, along with a set of deviations. While this allows for behaviorally faithful representations, it introduces a different formalism and diverges from RPST-based structural consistency. This divergence becomes particularly problematic when the BPMN model is fully unstructured (i.e., consists of a single rigid), in which case no RPST is constructed at all, and the model is translated entirely into a Petri net.

Additionally, Leopold et al. [7] employ a deprecated algorithm for RPST construction based on triconnected decomposition, which may fail to detect certain SESE regions [5].

In contrast, our approach maintains a consistent RPST-based structure across both structured and unstructured parts of the model. Furthermore, we propose a novel strategy for handling BPMN models with multiple end events, which is not addressed in [7]. By incorporating explicit mechanisms to merge terminal paths while preserving process semantics, our method produces a comprehensive linearization of BPMN models.

The paper [8] uses Natural Language Processing tools and LLMs to automatically create BPMN models from text descriptions. Our work focuses on model-to-text methods instead of text-to-model.

In [3] the structure of the process model was utilized for a guided generation approach to increase accuracy and consistency. In contrast, this work offers a more sophisticated approach on ordering the explained process model elements.

The survey [9] claims, that knowledge graphs are a promising strategy to reduce hallucinations and reviews specific techniques, their performance, and future potential. While our method uses knowledge graphs and their structure to improve generation results, the methods presented in the survey explore different directions, providing extra information to the LLM or optimizing their learning.

IV. Augmentation method

Our algorithm operates under a set of well-defined assumptions regarding the structure and semantics of the input BPMN model, which are derived from the BPMN documentation:

(1) Start events are not allowed to have any predecessors.
(2) All nodes must be dominated by a start event. In cases where a start event is absent within a participant, the participant must contain a node with no predecessor that dominates all of its nodes. Then, a start event can be inserted ahead of this node.
(3) The model may contain multiple start events within a participant, but at most one none-type start event is allowed per participant.
(4) End events are not allowed to have any successors.
(5) We expect all nodes to be post-dominated by an end event or a node with no successor. For example, models that solely consist of a cycle do not satisfy this condition and are not considered valid inputs.

Apart from the constraints 1–5, any BPMN model that conforms to the syntax and structural rules of [2] is considered valid input. For instance, tasks and gateways may have multiple incoming or outgoing edges. However, no assumptions are made about the model's behavioral correctness. In particular, checks for executability, deadlocks, or asynchronous sections, as discussed in [10], are beyond the scope of this work, as they do not directly affect the explainability of the model.

A. Preprocessing

Our preprocessing routine transforms a valid BPMN model represented as a directed graph into a structurally consistent form. The preprocessing comprises the following steps:

1) **Intermediate and Boundary Event Generalization**:
In the following, intermediate and boundary events are treated as tasks. A sequence flow from a task to its attached boundary event preserves their connection.
2) **Task Normalization**:
Ensures that each task or subprocess has exactly one incoming and one outgoing edge. A joining XOR gateway is inserted before each task that has multiple predecessors, while a splitting AND gateways is inserted right after a task with multiple successors. Tasks with no incoming or outgoing edges are prepended or appended with start or end events, respectively.
3) **Gateway Normalization**:
Gateways with at most one predecessor and one successor are treated as task nodes, thereby simplifying the control flow. Gateways with no incoming or outgoing edges are prepended or appended with start or end events, respectively.
4) **Start Event Normalization**:
If no start event exists, one is added before a node with indegree zero that dominates all other nodes of the graph. If multiple start events are present, they are changed to task nodes, and a new unified start event is introduced before them, connected via a splitting XOR gateway. This step follows the normalization approach of Polyvyanyy et al. [11].
5) **End Event Normalization**:
Each end event is ensured to have exactly one predecessor by prepending it with a joining XOR gateway where necessary. After this step, the model is guaranteed to have at least one end event.
6) **Recursive Preprocessing of Subprocesses**:
Processes embedded within tasks, referred to as subprocesses, are recursively preprocessed.
7) **Removal of Other BPMN Symbols**:
Other BPMN symbols not mentioned so far, such as groups, data objects, and text annotations showing additional information, which are not directly relevant for the control-flow of the process, are ignored and removed.
8) **Gateway Generalization**:
The specific types of gateways will be disregarded, i.e., our algorithm does not differentiate between AND and XOR gateways etc.

Now, the BPMN model has been normalized to a directed graph containing a reduced subset of BPMN symbols, and every node in the graph is dominated by a unique start event and post-dominated by an end event.

Our BPMN model from Fig. 1 is now transformed into Fig. 3 and has a corresponding RPST in Fig. 4.

The final step of the preprocessing phase is motivated by the possible existence of multiple end events and involves partitioning the BPMN model into a *main subgraph* and *early-end subgraphs*:

– One main subgraph: Let End be the last node of the longest among all shortest paths from the start event

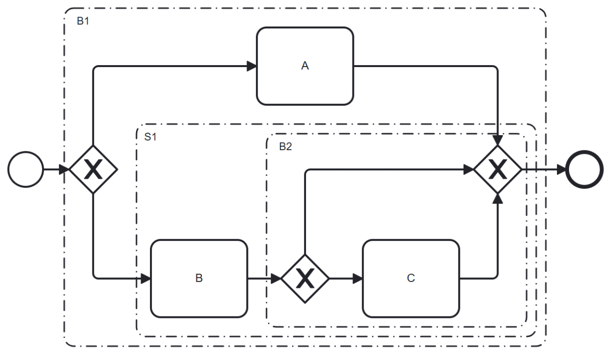

Fig. 3. Preprocessed BPMN model with two bonds (B1 and B2) and a sequence (S1)

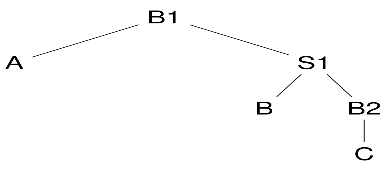

Fig. 4. The RPST of the workflow graph presented in Fig. 3

$Start$ to any end event in G. Then, the *main subgraph* is the subgraph of G where all nodes are dominated by $Start$ and post-dominated by End.

– A set of early-end subgraphs: A collection of subgraphs representing paths that deviate from the main subgraph and terminate prematurely. We add a new start event, succeeded by an exclusive gateway, and insert an edge from the gateway to each node, that was previously connected to the main subgraph.

If an early-end subgraph contains multiple end events, the partitioning procedure is applied recursively to that subgraph, resulting in a hierarchical decomposition of the process model. Ultimately, every subgraph is a *workflow graph.*

For a pair of distinct gateways, s and t, in a workflow graph G, we can detect a subgraph as a *rigid* $R(s,t)$ if its interior nodes IN satisfy following conditions:

1) Every node in IN is dominated by s and post-dominated by t;
2) Every node in IN is connected only to other nodes in $R(s,t)$;
3) s and t have at least 2 neighbors in $R(s,t)$ in the undirected version of G;
4) s and t have more neighbors in $R(s,t)$ than in any nested rigid $R(s,w)$ and $R(u,t)$ component
5) $R(s,t)$ does not correspond to a bond, a loop, or a sequence.

To structure a rigid component, we focus on edges within the rigid leading to any join gateway within the rigid, referred to as *non-structured edges*.

– *(split, join):* These edges originate from a split gateway and can be removed entirely without compromising reachability.
– *(task, join)* or *(join, join):* These edges require redirection to another gateway within the rigid rather than removal: the redirection must ensure that a path from the source node of the non-structured edge to the exit of the rigid component is maintained.

We aim to identify and act upon the edge that yields the greatest structural benefit. The guiding principle in this process is to modify the rigid component as minimally as possible, while still enabling its translation into a structured form. Specifically:

– For a non-structured edge with a split as its source node, we evaluate its usefulness based on the size reduction of the rigid upon removal.
– For a non-structured edge originating from a task or a join, we consider alternative gateways within the rigid as redirection targets, selecting the one that leads to the greatest size reduction.

Our algorithm for structuring a rigid is a simplified version of the algorithm for structuring homogeneous-XOR rigids presented by [10] which does not generate patches for the RPST, as we do not work with supplement and correction edges to maintain soundness [10].

B. RPST Derivation

For our topological ordering of process models, we transform workflow graphs into their respective RPST representations. Therefore, we employ a simplified version of the structuring method presented by Tang [10], disregarding the classification of gateway types. A structured workflow graph G can be represented as a hierarchy of blocks, i.e., as an RPST. Every time our algorithm detects a block, it replaces it with a new task node containing the block's corresponding RPST. In contrast to the approach in [10], our method distinguishes fewer patterns, as it deliberately abstracts from gateway types. Initially, each task node contains an RPST of size one, only containing the task itself. In this way, a workflow graph can be translated into an RPST in a bottom-up manner. The pattern reduction rules are as follows [10]:

Sequences are identified by first constructing the subgraph of G induced by all task nodes. Weakly connected components of this subgraph are then extracted, and components that represent linear/non-branching paths with more than one node are selected.

For each pair of a split and a join, all intermediate task nodes that are direct successors of the split and direct predecessors of the join are identified as potential bond members. Then, a bond is detected if this set contains more than one element.

A loop pattern is a simple cycle of length 3 or 4 containing two gateways, where the entry is a joining gateway and the exit is a splitting gateway. Specifically, a loop pattern contains two kind of tasks: a *loop body* and a *redo part*, which is dominated by the loop body. A loop fulfilling these conditions can be detected by using Johnson's algorithm for finding all the elementary circuits of a directed graph [12].

Figures 5, 6 and 7 visualize how the different patterns are reduced.

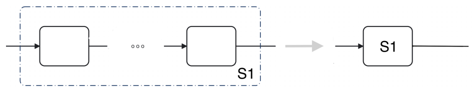

Fig. 5. Sequence pattern reduction. The new task node S1 contains an RPST with S1 as the root and the RPSTs of the members of the sequence as its children.

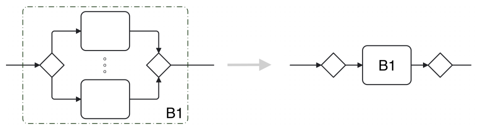

Fig. 6. Bond pattern reduction. The new task node B1 contains an RPST with B1 as the root and the RPSTs of the members of the bond as its children.

We can fully translate an unstructured workflow graph by iteratively applying the pattern reductions in figures 5, 6 and 7 and, upon encountering a point of failure, structuring the rigids responsible for the obstruction. This process is repeated until the entire workflow graph is completely translated into an RPST.

C. Topological Ordering

A topological ordering of the elements in a workflow graph G – that is, a linear sequence of its nodes where each node appears before all nodes it directs to – can be derived straightforwardly by traversing the leaves of its RPST representation in pre-order. Since loops in G are encapsulated as distinct blocks within the RPST, the RPST itself is acyclic, ensuring that a topological sort of G is always possible using this structure.

In BPMN models, subprocess tasks encapsulate lower-level processes to enhance model clarity. To preserve this structure during topological sorting, our algorithm processes each subprocess recursively. The resulting orderings from these subprocesses are appended to the end of the main workflow's topological order, rather than interleaving them within the main sequence. This design choice avoids disrupting the logical flow of the primary process.

Finally, it is crucial that the ordering of a block's children within the RPST is well-defined, as this preserves dependency constraints and may yield a more intuitive overall ordering. Algorithm 1 captures our procedure for computing a topological ordering of the elements of a workflow graph G.

In practice, not every BPMN model satisfies the structural conditions required to constitute a workflow graph – for instance, a model may contain multiple end events, thereby violating these assumptions. To address such cases, the preprocessing phase (Section IV-A) is applied that transforms the input BPMN model through a series of steps. This process results in a partitioning P of the original model into one or more workflow graphs. The topological orderings of these individual graphs are subsequently merged to form a unified ordering. We present an algorithm, which outlines the final algorithm used to compute the topological sorting of a BPMN model's elements using its RPST representation. It manages

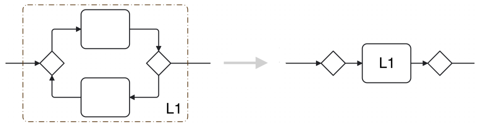

Fig. 7. Loop pattern reduction. The new task node L1 contains an RPST with L1 as the root and the RPST of the loop body as its left child and the RPST of the redo part as its right child.

```
Algorithm 1: TopologicalSort($G$)
  Input: A workflow graph $G$
  Output: A list of all the nodes of $G$, topologically
          sorted
1 $rpst \leftarrow$ the RPST of $G$;
2 $result \leftarrow [Start, root(rpst), End]$;
3 while there exists a task node $b$ in $result$
    corresponding to a block do
4     Replace $b$ in $result$ with its children in $rpst$;
5 foreach Subprocess $s \in result$ do
6     $result \leftarrow result$ + TopologicalSort($s$);
7 return $result$;
```

algorithm 1 for early ends and puts their results in order. We first use algorithm 1 for the main subgraph. Then we use it again for each early-end subgraph and insert the result at the point, where they are separated from the main subgraph to receive a total ordering.

V. Experiments

To evaluate how well the algorithm's output aligns with human reasoning in process explanation, an empirical study was conducted. While the algorithm ensures syntactic correctness by preserving control-flow dependencies, this does not guarantee that the resulting order is intuitive for human users. Since clarity and comprehensibility are essential in process communication, the evaluation compares algorithm-generated orderings with those created by users.

We conducted an empirical study with 30 participants possessing basic knowledge of graph theory and modeling, aiming to evaluate the intuitiveness of the algorithm's output for topologically sorting BPMN elements. Participants were presented with a series of BPMN models and asked to assign sequential numbers to each element, indicating the order in which they would verbally explain the process.

To examine the effect of semantic context, we designed a questionnaire, containing a mix of BPMN models.

This method was chosen to capture participants' natural explanation strategies, allowing a quantitative evaluation. To quantify the similarity to the algorithm's output as well as the agreement among the participants, we used the *normalized generalized Levenshtein distance* [13], while the *control-flow complexity* [14] was used to quantify the structural complexity of each model. In addition to evaluating alignment with the algorithm statistically, we also examined how confidence ratings relate to both model complexity and sequence similarity.

A. Hypotheses

Our experiments investigated two specific hypotheses which incorporate both, the complexity of the model and bad modeling practices (rigids).

H1 As the complexity of the model increases, the participants' orderings tend to diverge more from the algorithm's output.

H2 The greatest discrepancies between participant orderings and the algorithm's output are expected in models with rigid components, as opposed to structured models.

B. Hypothesis 1

A linear regression was conducted to examine whether control-flow complexity (CFC) predicts the degree to which participants' orderings deviate from the algorithmic reference ordering, as measured by the normalized GLD. The results showed that CFC significantly predicted normalized GLD, $\beta = 0.030$, $R^2 = 0.093$, $F(1, 178) = 18.19$, $p < 0.001$: Models with higher control-flow complexity were associated with greater divergence from the reference ordering. This finding supports the hypothesis that increased model complexity leads to less alignment between participant orderings and the algorithmic ideal. The trend is illustrated in Fig.8, where a positive relationship between CFC and normalized GLD is evident.

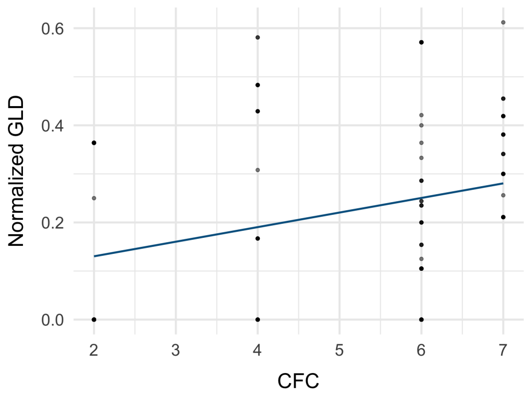

Fig. 8. Relationship between control-flow complexity (CFC) and the normalized generalized Levenshtein distance. Each point represents a participant's ordering for a specific model. The solid line indicates the linear regression fit, showing that higher CFC is associated with greater divergence from the algorithmic reference ordering.

C. Hypothesis 2

A two-sample *t*-test was conducted to assess whether participant orderings diverged more from the algorithm in unstructured models compared to structured ones. The assumption of equal variances was met, $F(1, 178) = 0.31, p = .58$. The results revealed that participants showed significantly greater divergence in unstructured models (M = 0.28, SD = 0.18) than in structured models (M = 0.17, SD = 0.19), $t(178) = 3.55, p < .001$. This supports the hypothesis that models with rigids yield higher discrepancies between algorithmic and human orderings compared to structured models.

VI. Discussion and conclusion

In this work, we presented an algorithm for topologically sorting a BPMN model's elements, building on existing approaches with several key adaptations aimed at improving intuitiveness and alignment with human reasoning. Our method augments a process model employing a block-oriented strategy to restructure the control flow inspired by Tang [10] and introduces a novel handling of multiple end events by partitioning the model into multiple workflow graphs. To address unstructured models, we modified and employed a structuring technique of [10] that serves as an alternative to Petri net-based translations, such as those proposed by Leopold et al. [7], which similarly utilize a block-oriented strategy for the linearization of BPMN models.

Additionally, we implemented the algorithm as a `Python` tool capable of processing Camunda 8 BPMN 2.0 models [2], [15], producing a topological ordering of element IDs for a reduced but representative subset of its elements.

This topological ordering can be used to support LLM-generated descriptions, by helping to order the divided graphs in approaches like [3].

References

[1] H. Paulheim, "Knowledge graph refinement: A survey of approaches and evaluation methods," *Semantic web*, vol. 8, no. 3, pp. 489–508, 2016.

[2] M. Weske, *Business Process Management: Concepts, Languages, Architectures*, ch. 4.7, pp. 206–242. Springer Berlin Heidelberg, 2012.

[3] T. Tel and M. Minor, "Utilizing the Structure of Process Models for Guided Generation of Explanatory Texts," in *Case-Based Reasoning Research and Development - 33rd International Conference, ICCBR 2025, Biarritz, France, June 30 - July 3, 2025, Proceedings* (I. Bichindaritz and B. López, eds.), vol. 15662 of *Lecture Notes in Computer Science*, (Biarritz, France), pp. 157–171, Springer, 2025.

[4] O. M. Group, "Business process model and notation," Jan. 2014.

[5] Y. Choi, N. L. Ha, P. Kongsuwon, and K. Han, "An alternative method for refined process structure trees (rpst)," *Business Process Management Journal*, 10 2019.

[6] J. Vanhatalo, H. Völzer, and J. Koehler, "The refined process structure tree," *Data and Knowledge Engineering*, vol. 68, no. 9, pp. 793–818, 2009. Sixth International Conference on Business Process Management (BPM 2008).

[7] H. Leopold, J. Mendling, and A. Polyvyanyy, "Supporting process model validation through natural language generation," *IEEE Transactions on Software Engineering*, vol. 40, no. 8, pp. 818–840, 2014.

[8] J. T. Licardo, N. Tanković, and D. Etinger, "A method for extracting bpmn models from textual descriptions using natural language processing," *Procedia computer science*, vol. 239, pp. 483–490, 2024.

[9] G. Agrawal, T. Kumarage, Z. Alghamdi, and H. Liu, "Can knowledge graphs reduce hallucinations in llms?: A survey," *arXiv preprint arXiv:2311.07914*, 2023.

[10] C. Tang, "Automatically block-structuring of bpmn models," Master's thesis, Eindhoven University of Technology, 2022.

[11] A. Polyvyanyy, J. Vanhatalo, and H. Völzer, "Simplified computation and generalization of the refined process structure tree," in *Web Services and Formal Methods: 7th International Workshop, WS-FM 2010, Hoboken, NJ, USA, September 16-17, 2010. Revised Selected Papers 7*, pp. 25–41, Springer, 2011.

[12] D. B. Johnson, "Finding all the elementary circuits of a directed graph," *SIAM Journal on Computing*, vol. 4, no. 1, pp. 77–84, 1975.

[13] L. Yujian and L. Bo, "A normalized levenshtein distance metric," *IEEE transactions on pattern analysis and machine intelligence*, vol. 29, no. 6, pp. 1091–1095, 2007.

[14] J. Cardoso, "Control-flow complexity measurement of processes and weyuker's properties," in *6th International Enformatika Conference*, vol. 8, pp. 213–218, 2005.

[15] Camunda Services GmbH, *Camunda 8 Documentation*, 2024. Accessed: 01.04.2025.