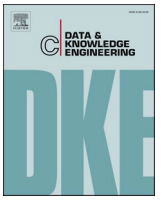
Data & Knowledge Engineering 164 (2026) 102601

Contents lists available at ScienceDirect

Data & Knowledge Engineering

journal homepage: www.elsevier.com/locate/datak



An optimized backbone-based process layout generation method using integer programming and heuristics to enhance user comprehension

Deoksang Lee [a,b], Minseok Song [a,b,*], Wil M.P. van der Aalst [c]

[a] Department of Industrial and Management Engineering, Pohang University of Science and Technology (POSTECH), Pohang, Gyeongbuk 37673, Republic of Korea
[b] van der Aalst Data & Process Science Research Center, Pohang University of Science and Technology (POSTECH), Pohang, Gyeongbuk 37673, Republic of Korea
[c] Process and Data Science Group, RWTH Aachen University, Ahornstrasse 55, Aachen, NRW 52074, Germany

ARTICLE INFO

Keywords:
Process layout
Backbone
Process model layout
Visualization
Integer programming
Heuristic method

ABSTRACT

Process mining is a discipline focused on extracting valuable insights about business processes from event logs collected within information systems. A key component of process mining is visualizing process models with the aim to transform complex processes into clear and intuitive graphical representations. Such visualizations create transparency and can be overlaid with conformance and performance diagnostics. Recently, a stable graph layout algorithm has been introduced to enhance model readability by preserving the process backbone—the sequence of activities that best represents the event log. The backbone-based process layout offers a key advantage: even when filters are applied, the backbone remains intact. As a result, node positions change minimally compared to other layout methods, making it easier for users to locate relevant nodes and edges in the filtered view. Despite these benefits, several challenges remain that hinder process comprehension. In this paper, we identified key issues in existing layout methods, including unnecessary back edges, restricted horizontal alignment, non-orthogonal backbone structures, and greedy-based node placement. To address these limitations, we propose an enhanced process layout generation method that integrates integer programming with heuristic techniques. We evaluate our method using real-world event logs, showing notable reductions in back edges, edge crossings, and edge lengths, along with improvements in node orthogonality and layout symmetry. A qualitative user study further confirms a strong preference for our layout over the existing benchmark method, demonstrating its effectiveness in improving process model comprehension.

1. Introduction

In recent years, Information and Communication Technology (ICT) has increasingly been leveraged to optimize business processes by identifying inefficiencies, reducing operational costs and maximizing revenue. Among these ICT technologies, process mining has

* Corresponding author at: Department of Industrial and Management Engineering, Pohang University of Science and Technology (POSTECH), Pohang, Gyeongbuk 37673, Republic of Korea.
E-mail address: mssong@postech.ac.kr (M. Song).

https://doi.org/10.1016/j.datak.2026.102601
Received 14 November 2025; Received in revised form 2 March 2026; Accepted 20 March 2026
Available online 22 March 2026

emerged as a powerful method for analyzing business processes using event logs—data collected from information systems at business sites [1–4]. As an academic discipline, process mining focuses on extracting valuable process-related knowledge from event logs through various techniques, including process discovery, conformance checking, and process enhancement [5].

By adopting process mining, stakeholders can gain a comprehensive understanding of business processes, enabling the detection and resolution of issues such as delays, rework, and bottlenecks [5–9]. Given these benefits, process mining has been widely adopted across domains, including healthcare, IT, finance, manufacturing, education, and government [10–13].

At the heart of process mining lies the process model, a structured representation of workflow capturing the sequence of activities and interactions among resources [14]. These process models support the diagnosis of inefficiencies and deviations, ultimately facilitating process improvement. However, the effectiveness of process models depends heavily on their visualization. Poor visualization obscures understanding, impedes decision-making, and reduces model usability [15–17]. Therefore, it is important to present process models in a visually clear and accessible manner.

To enhance comprehension, factors such as the presentation medium, primary notation, secondary notation, label design, model characteristics, comprehension tasks, and user attributes must be considered [18]. Among these, the process layout—a key component of secondary notation—plays a critical role in conveying the structure of a process model [19,20]. Well-designed layouts significantly improve readability and interpretability, while poor layouts hinder comprehension, as illustrated in Fig. 1.

Despite the crucial role of layout in enhancing the understandability of process models, relatively few studies have addressed methods for automatically generating high-quality process model layouts. Gschwind et al. [21] proposed an automated process layout generation method that focuses on computational efficiency and provides an important foundation for subsequent research on process layout. In parallel, graph layout research explored techniques for emphasizing the main structural characteristics of graphs. In this context, Misue et al. [22] introduced the concept of a backbone structure, which highlights the main structure of the graph. Building on the process layout research, Mennens et al. [23] adopted the backbone concept from graph layout and integrated it into process layout generation. Their backbone-based process layout method explicitly emphasizes the main execution flow of a process by identifying a sequence of activities that represent the most frequent behavior from start to end, while arranging less frequent or exceptional activities around this core structure (Fig. 2). In addition, by preserving node positions along the backbone during filtering, the approach improves layout stability, allowing users to track changes in the process model with minimal visual disruption, as illustrated in Fig. 3. This method was implemented in UiPath, a widely used commercial tool for RPA and process mining [24].

Despite its strengths, the backbone-based process layout approach has several limitations:

1. Unnecessary back edges, which disrupt the natural top-down flow and reduce overall process flow consistency.
2. Lack of horizontal edge support, which results in inefficient space utilization and may misleadingly suggest a sequential relationship between two nodes that should be interpreted as concurrent or unordered.
3. Non-linear placement of backbone nodes, which obscures the location and continuity of the process backbone, making it harder for users to follow the primary process flow.
4. Greedy node distribution, which leads to asymmetrical layouts, reducing visual balance.

Addressing these limitations is important not only for improving process model interpretability but also for providing high-quality and structurally consistent layouts that can serve as reliable references for future automated layout generation. While recent graph

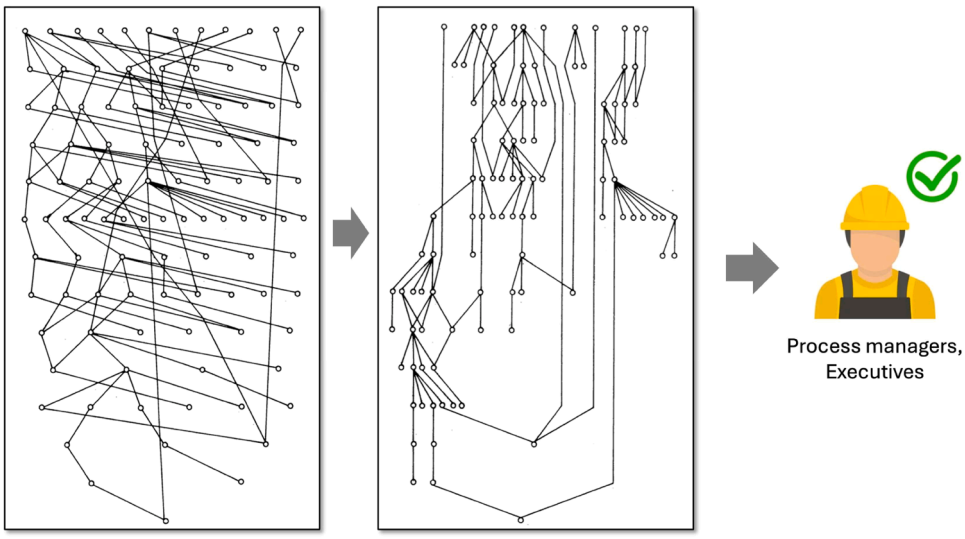


Fig. 1. Comparison of poorly and well-designed process layouts using the same graph. A well-designed layout significantly enhances user comprehension.

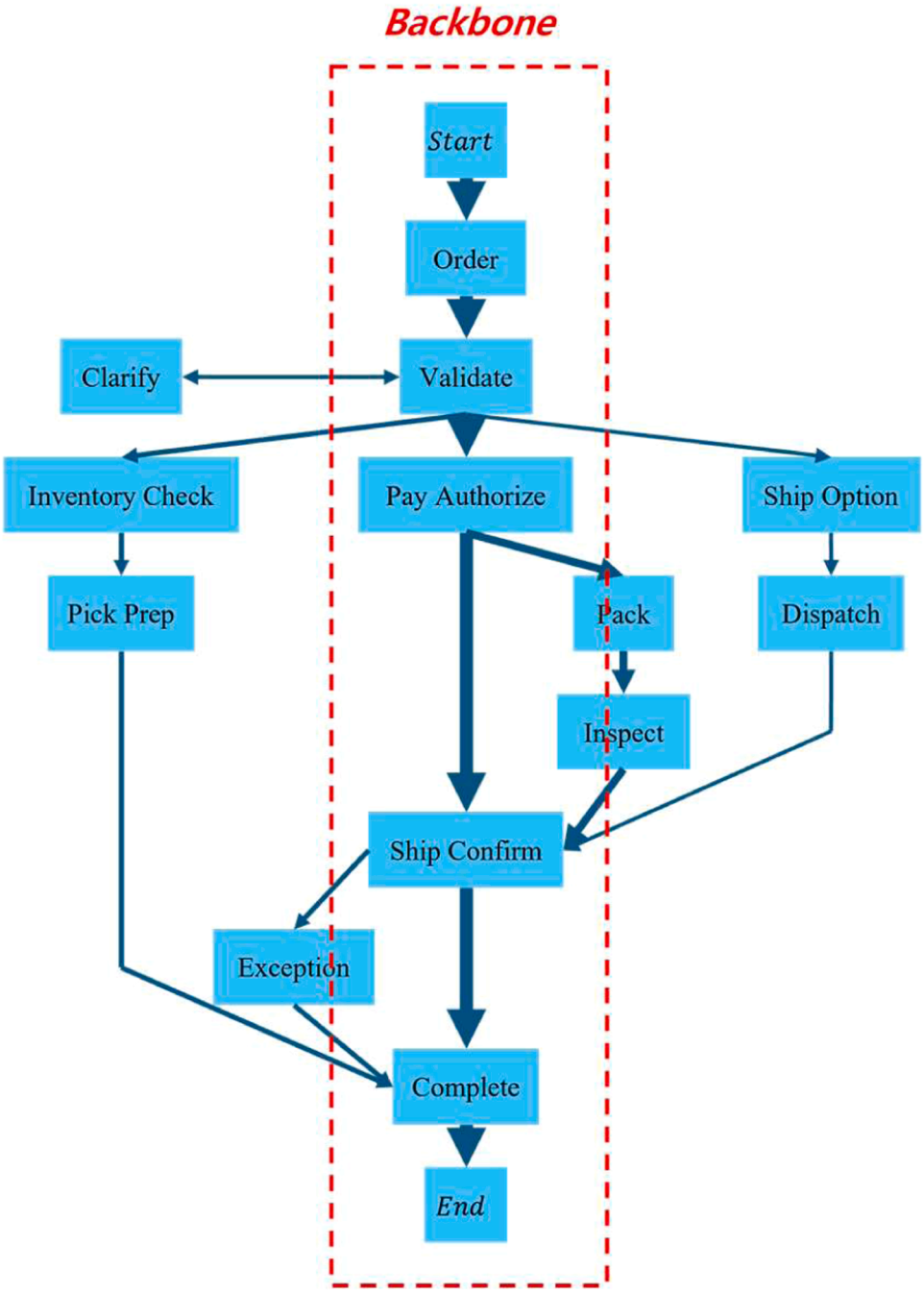


Fig. 2. An example of backbone-based process layout. The backbone usually consists of activities included in the most frequent process sequence.

layout research has explored AI-based approaches that adapt layouts to user preferences, such techniques have received limited attention in the context of process mining. In this setting, improving backbone-based process layouts is particularly important, as well-structured and interpretable layouts can be used as high-quality training data for learning-based process layout methods. Consequently, enhancing backbone-based layouts represents a necessary step toward more effective, extensible, and data-driven process visualization methods.

To this end, we propose an enhanced backbone-based process layout method that combines integer programming and heuristic techniques, applied specifically to the Directly Follows Graph (DFG)—a widely used process model format in process mining. The method is evaluated both quantitatively and qualitatively. Quantitative experiments on five event logs (weblog, hospital log, BPI 2017, BPI 2018, and BPI 2019) demonstrate significant improvements in layout quality. Qualitative assessments further confirm that our

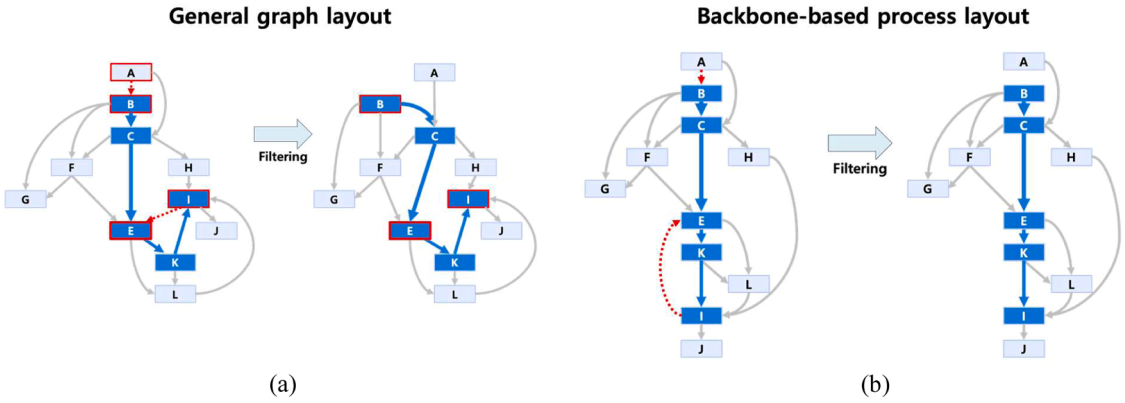


Fig. 3. (a) General graph layout before and after filtering edges *(A, B)* and *(I, E)*, showing significant node displacement. (b) Backbone-based layout under the same filtering condition, preserving node positions and improving traceability.

layout improves clarity and user preference over the existing method.

2. Related work

This section reviews prior research on general graph layout techniques for visualizing structured information and discusses how such techniques have evolved toward AI-based approaches. It then surveys process model layout methods designed to improve the interpretability of business processes, with particular attention to backbone-based process layouts.

2.1. Graph layout

Graph layouts aim to determine how nodes and edges of a graph should be positioned to produce visualizations that are easy for users to understand. A foundational contribution was made by Sugiyama et al. [25], who proposed a framework for hierarchical graph layout that arranges nodes into layers based on their relationships. Eades [26] introduced a force-directed layout method in which nodes are modeled as physical objects that attract or repel one another, allowing the graph to naturally settle into a balanced, readable configuration. Building on the Sugiyama framework, Gansner et al. [27] proposed concrete algorithms for generating layouts of directed graphs, including methods for node ordering, edge routing, and crossing minimization, thereby establishing a practical foundation for hierarchical graph drawing.

2.2. AI-based graph learning and layout generation

Recent studies have demonstrated a paradigm shift toward data-driven and adaptive model generation, in which advanced computational models are used to capture complex and nonlinear structures. Artificial Intelligence (AI) and graph-based techniques have shown strong potential in modeling intricate dependency patterns across a wide range of domains. In the context of graph-related analysis, several studies have explored AI-based approaches to learn graph structures from data. For example, Xu and Zhang [28] and Jin and Xu [29] combined vector error-correction models with directed acyclic graphs to identify causal relationships in housing and steel price dynamics across multiple cities. Zhao et al. [30] proposed a GNN-based method to detect power grid topology.

AI techniques have also been applied to generate graph layouts for large and complex networks. Wang et al. [31] proposed an LSTM-based model that learns node features and graph connectivity patterns to directly generate graph layouts. Kwon and Ma [32] introduced an encoder-decoder neural network that can produce multiple candidate layouts for a given graph. Tiezzi et al. [33] presented a graph drawing framework based on GNN that explicitly optimizes aesthetic criteria, in particular minimizing edge crossings through supervised learning. These studies demonstrate the potential of AI-based approaches to produce visually appealing layouts. However, they typically require large amounts of labeled training data and substantial computational resources, which limit their applicability in many practical settings.

2.3. Process layout and backbone-based process layout

Process layouts aim to visualize business processes in a way that enables users to clearly understand execution order, control-flow relations, and dominant behavioral patterns. Unlike general graph layouts, where aesthetic criteria such as symmetry or uniform edge lengths are primary concerns, process layout places greater emphasis on preserving the semantics of process flow, particularly causality, concurrency, and routing behavior. Relatively few layout algorithms have been developed for process models. Gschwind et al. [21] proposed one of the earliest dedicated process layout algorithms to enable linear-time layout computation.

More recently, Mennens et al. [23] introduced a backbone-based process layout approach that shifted the focus of process

visualization toward backbone-centered representations. Their method identifies the most frequent process flow in the event log as the backbone and places all other activities to the left or right of the backbone. The algorithm computes vertical and horizontal positions of nodes from the event log, and the positions remain stable when filtering is applied. This stability allows users to easily detect which parts of the process change when different subsets of the log are analyzed, thereby supporting interactive and exploratory process analysis. This backbone-based process layout paradigm has become highly influential in commercial process mining tools such as Celonis [34]. These systems implicitly confirm that backbone-based layouts are particularly effective for helping users understand complex, real-life business processes.

3. Problem definition: limitations of the existing approach

This section outlines the limitations of backbone-based process layouts proposed by Mennens et al. [23], which hinder users' ability to understand process models effectively. To identify the limitations, we conducted a survey to examine key layout factors that influence process comprehension. The findings highlight six critical factors that significantly affect how users interpret process layouts: node orthogonality, symmetry, edge crossings, edge lengths, edge orthogonality, and flow consistency, as summarized in Table 1. Several layout factors were excluded from the target factors for the following reasons. Colors were excluded because they do not depend on the layout algorithm itself. Edge bends were not considered, as their effect overlaps with edge orthogonality, which has been more studied. Node overlaps were not considered because the benchmark prohibits node overlaps by design, while edge overlaps are already captured by the edge crossing metric. Text-related factors, including text on nodes and edges, were excluded, since they are not handled by the benchmark layout. Finally, the total number of ending points and their placement were excluded because the benchmark process layout terminates at a single end node.

Node orthogonality refers to how efficiently the layout utilizes space, specifically the proportion of grid points occupied by nodes. Prior studies argued that efficient area utilization enhances process layout readability by improving the visibility of nodes and associated texts [36,39].

Symmetry concerns the visual balance of node placement. In backbone-based process layouts, symmetry is measured by comparing nodes on the left and right of the backbone. Purchase [35] demonstrated that symmetrical layouts reduce users' reaction time when responding to questions. Additionally, Schrepfer et al. [37] suggested that symmetrical layouts offer structured visual representations, facilitating better readability and comprehension.

Edge crossings count the intersections between edges. Minimizing edge crossings is widely recognized as essential for enhancing comprehension. Purchase [35] demonstrated that an increased number of edge crossings negatively impacts users' understanding. Effinger et al. [38] evaluated the impact of edge crossings across diverse user groups, revealing that fewer crossings enhance clarity and user preference.

Edge lengths refer to the sum of all edge lengths in the layout. Studies by Bernstein and Soffer [39] and Burattin et al. [40] recommended minimizing edge lengths, as shorter edges contribute to better layout comprehension based on findings from user interviews.

Edge orthogonality evaluates how closely edges align with a Cartesian grid. Purchase [36] suggested maximizing edge orthogonality to generate aesthetically pleasing layouts that enhance users' comprehension. Effinger et al. [38] provided empirical evidence that increased edge orthogonality positively influences process comprehension through experiments.

Flow consistency assesses whether the directional flow of processes (e.g., top-to-bottom or left-to-right) remains uniform throughout the layout. Purchase [36] emphasized the importance of maintaining a consistent flow direction to support users' process comprehension. Subsequent studies by Effinger et al. [38], Bernstein and Soffer [39], and Burattin et al. [40] reinforced this suggestion through user experiments and interviews, further validating the significance of flow consistency in improving process layout clarity.

Based on the six layout factors, we identified several limitations of the backbone-based process layouts. These limitations are the

Table 1
Layout factors influencing process model comprehension.

Category	Factor	Study						Target factor
		Purchase [35]	Purchase [36]	Schrepfer et al. [37]	Effinger et al. [38]	Bernstein and Soffer [39]	Burattin et al. [40]	
Area usage	Node orthogonality		O			O		✓
	Symmetry	O	O	O		O	O	✓
Coloring	Colors used				O			
Edge	Edge bends	O	O	O				
	Edge crossings	O	O	O	O	O	O	✓
	Edge lengths					O	O	✓
	Edge orthogonality	O	O		O	O	O	✓
	Flow consistency		O		O	O	O	✓
Elements	Overlap				O			
	Text on nodes				O			
	Text on edges					O	O	
Node	Total ending points					O		
	Placement of ending point					O	O	

foundation for proposing improvements to enhance users' understanding of process models. The identified limitations and their associated factors are outlined in Table 2.

3.1. Presence of unnecessary back edges

The first limitation is the presence of unnecessary back edges, which disrupts flow consistency. The existing study assigns node ranks heuristically without considering flow consistency, resulting in unnecessary back edges, as illustrated on the left side of Fig. 4. Although this heuristic approach is simple and intuitive, there remains potential for further optimization. Reducing back edges leads to more readable process layouts, as depicted on the right side of Fig. 4, improving flow consistency.

3.2. Restriction against horizontal edges

The second limitation is the restriction against horizontal edges, which affects edge lengths, node orthogonality, and flow consistency. In the existing method, node ranks are assigned under the constraint of avoiding horizontal edges to represent the execution flow intuitively. However, as shown on the left side of Fig. 5, this constraint leads to unnecessary back edges and longer edge lengths when cycles exist between nodes. Additionally, avoiding horizontal edges results in inefficient area utilization, as more ranks are required. Allowing horizontal edges, as illustrated on the right side of Fig. 5, reduces both back edges and edge lengths while making more efficient use of space. From a semantic perspective, positioning nodes involved in cycles at the same rank represents their precedence relationships more accurately, such as the transition between the green and blue nodes in Fig. 5.

3.3. Non-linear backbone alignment

The third limitation is non-orthogonal backbone edges, which negatively affect edge lengths and edge orthogonality. The prior method assigns node coordinates using the algorithm proposed by Gansner et al. [27]. However, this method does not account for the backbone structure, leading to a non-linear arrangement of backbone nodes, as shown on the left side of Fig. 6. This increases edge lengths and reduces edge orthogonality, making it more difficult for users to identify the primary flows of the processes. To address this limitation, backbone nodes need to be arranged linearly, as shown on the right side of Fig. 6. This approach improves edge lengths and edge orthogonality, enabling users to more easily discern the primary flows of processes.

3.4. Imbalanced node distribution

The fourth limitation arises from the heuristic left-right node placement approach, which often yields imbalanced layouts and degrades symmetry and edge crossings. In the existing method, all components are initially placed on the right side of the backbone, and then components are shifted to the left in descending order of size (the number of nodes included in a component) until the node count difference between the two sides is minimized. However, this heuristic does not always produce an optimal arrangement. For example, consider components whose sizes are 11, 5, 4, 3, 3, and 2 as illustrated in Fig. 7. The heuristic places components 11 and 5 on the left (16 nodes) and the remaining components on the right (12 nodes), resulting in a four-node difference. In contrast, an optimal arrangement places components 11 and 3 on the left and 5, 4, 3, and 2 on the right, achieving perfect balance. Such imbalances distort visual symmetry and tend to increase edge crossings, especially when many nodes are concentrated on one side of the backbone. Excessive edge crossings hinder users' ability to trace process flows clearly. To resolve this issue, a more advanced node placement strategy is required. This not only improves visual symmetry but also provides sufficient space for edge routing, ultimately improving overall layout readability.

4. Background

This section outlines the foundational concepts for this study. $B(A)$ is the set of all multisets over some set.

Definition 1. (*Universes*). *Let $\mathbb{U}_{ev}$ be the universe of events, $\mathbb{U}_{act}$ the universe of activities, $\mathbb{Z}$ the universe of integers.*

Definition 2. (*Event* log [41]). *An event* log $L \in B(\mathbb{U}_{act}^{*})$ *is a multiset of traces over a set of activities $\mathbb{U}_{act}$. A trace $\sigma = \langle a_1, a_2, \ldots, a_n \rangle$ is a finite sequence such that each $a_i \in \mathbb{U}_{act}$.*

For example, Table 3 shows two cases c_1 and c_2 with four possible activities, which are *oi*, *po*, *pi*, and *ri*. The trace of c_1 is $\langle oi, po, pi \rangle$

Table 2
Limitations of the existing backbone-based process layout approach.

Limitation	Factors
Presence of unnecessary back edges	Flow consistency
Restriction against horizontal edges	Edge lengths; Node orthogonality
Non-orthogonal backbone edges	Edge lengths; Edge orthogonality
Heuristic left-right node placement	Edge crossings; Symmetry

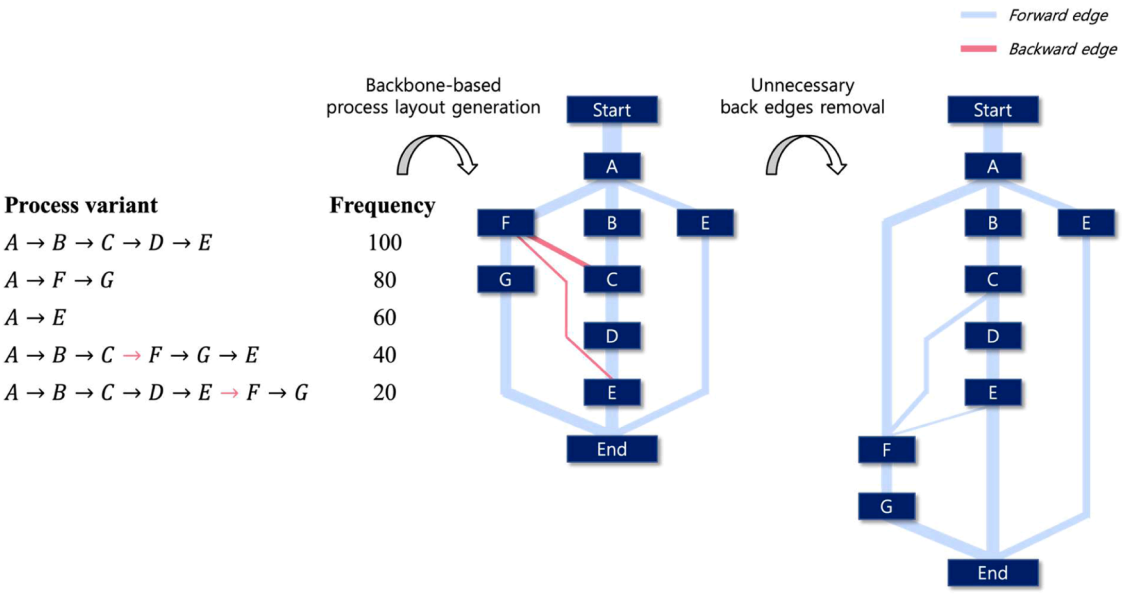


Fig. 4. Unnecessary back edges in the existing approach hinder flow consistency and can be removed for improved consistency.

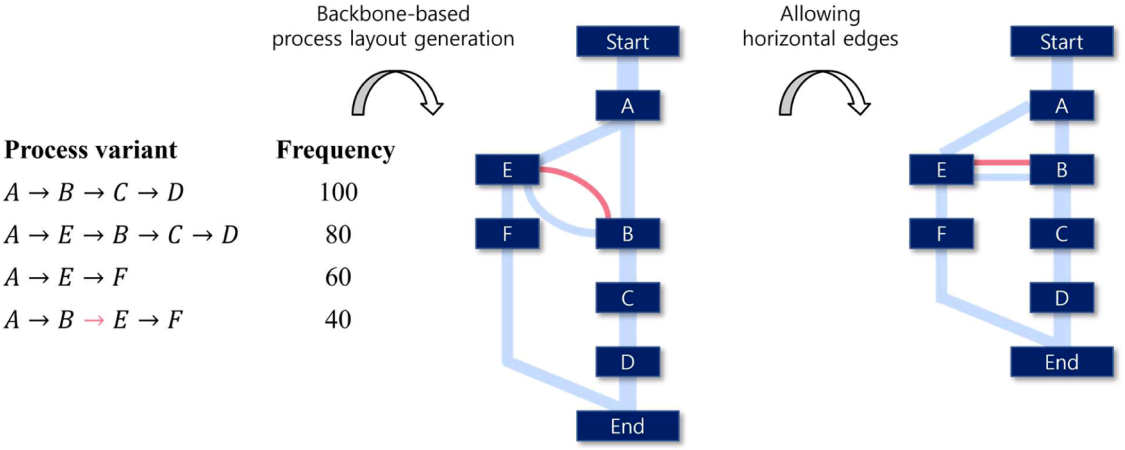


Fig. 5. In the existing approach, horizontal edges are not allowed, which may incorrectly imply a precedence relationship between the two nodes *B* and *E*. By permitting horizontal edges, such nodes are correctly interpreted as unordered or concurrent.

and the trace of c_2 is $\langle oi, po, ri \rangle$. Then, the event log is $L = [\langle oi, po, pi \rangle, \langle oi, po, ri \rangle]$.

Definition 3. (*Directly follows graph* [41]). *A DFG is a pair* $G = (N, E)$ *where* $N \subseteq \cup_{act}$ *is a set of activities and* $E \subseteq N \times N$ *is a set of directed edges.*

Definition 4. (*Directly follows graph discovery* [41]). *Let L be an event* log. *The DFG discovery function* $disc_{DFG}(L) = (N, E)$ *is defined as below:*

- $L' = \{\langle start \rangle \cdot \sigma \cdot \langle end \rangle \mid \sigma \in L\}$.
- $N = \{a \in \sigma \mid \sigma \in L'\}$ and $E = \{(a_i, a_{i+1}) \mid \langle a_1, \ldots, a_k \rangle \in L', 1 \leq i \leq k\}$.

Definition 5. (*Backbone*). *Let* $G = (N, E)$ *be a directed graph. A backbone of G is a pair* $BB = (BN, BE)$, *where* $BN = \langle a_1, a_2, \ldots, a_k \rangle$, $a_i \in N, a_i \neq a_j$, *and* $BE \subseteq \{(n_i, n_{i+1}) \mid 1 \leq i < k\}$.

Conceptually, the backbone serves as the central axis of the process model, representing a primary path in an event log. The primary path can be the most frequent sequence of activities, the longest path, or a user-selected reference path. The method for determining the backbone is explained in Section 5.2

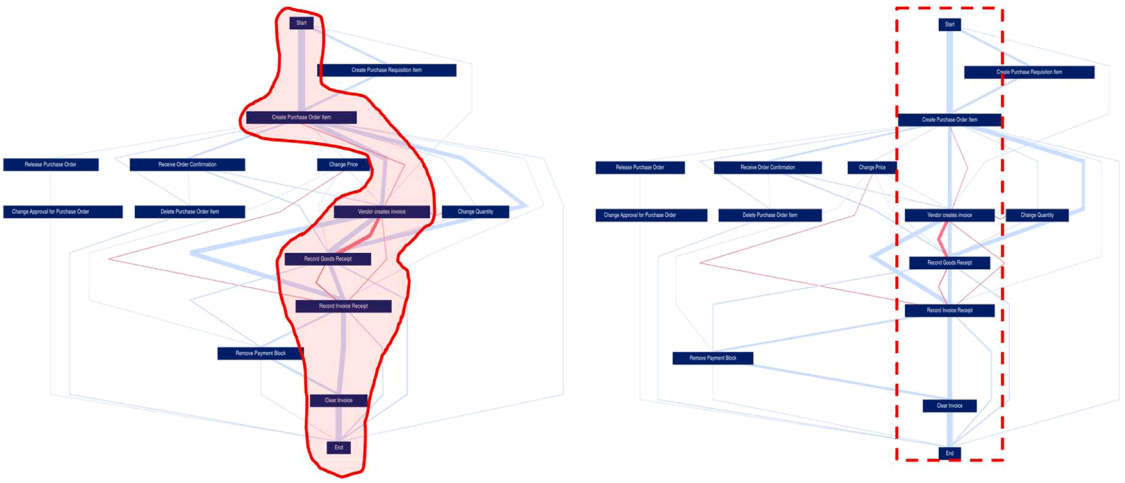


Fig. 6. The existing approach fails to preserve the backbone's linear structure, making it difficult for users to identify the main process flow. Maintaining a linear backbone improves readability and highlights the primary sequence of activities.

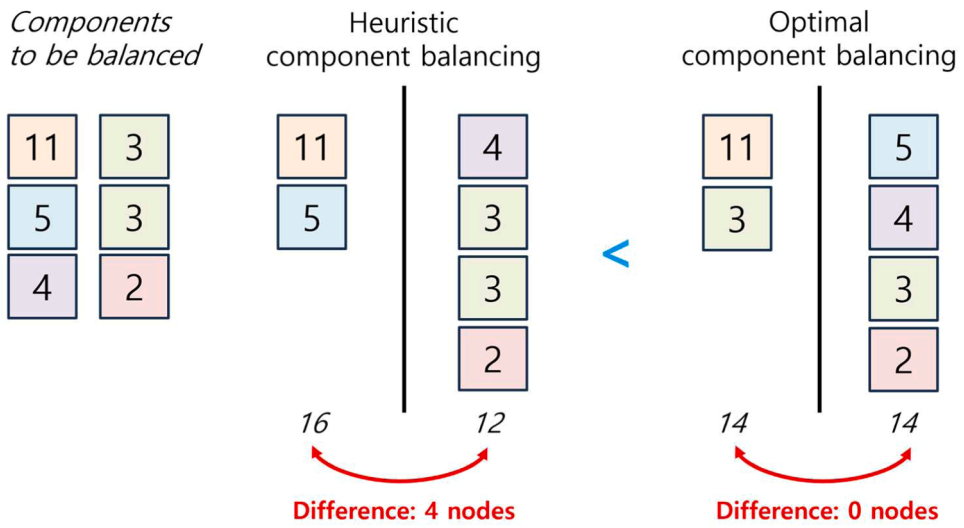


Fig. 7. Comparison between heuristic and optimal component balancing. Each number in the square represents the size of a component (i.e., the number of nodes it contains).

Table 3
A fragment of the event log.

Case	Activity	Timestamp
c_1	*order item* (*oi*)	18:10
c_1	*pay order* (*po*)	18:15
c_2	*order item* (*oi*)	18:16
c_2	*pay order* (*po*)	18:20
c_1	*pick item* (*pi*)	18:30
c_2	*refund item* (*ri*)	18:35

Definition 6. (*Rank and order*). *Let N be a set of nodes. Each node* $n \in N$ *has a rank* $r_n \in \mathbb{Z}^+$ *and an order* $o_n \in \mathbb{Z}$.

Rank determines the vertical placement of each node in the graph, while order specifies the horizontal alignment. As shown in Fig. 8., node *A* occupies the top layer with $r_A = 1$, and node *B*, positioned directly below, has $r_B = 2$. In the same figure, the node *B* has $o_B = 0$, whereas the node to its right, *H*, has $o_H = 1$.

Definition 7. (*Real and virtual nodes*). *Let L be an event* log. *Let $N = \{a \in \sigma \mid \sigma \in L\} \cup \{start, end\}$, be a set of nodes in the event* log *including the synthetic start and end nodes. Let $G = (M, E)$ be a directed graph. A node $n \in M$ is a real node iff $n \in N$; otherwise, it is a virtual node.*

Conceptually, real nodes represent activities recorded in the event log, whereas virtual nodes are artificially introduced nodes to connect real nodes positioned at non-adjacent ranks. For example, in Fig. 9(a)., there are edges linking two real nodes that empty ranks exist between them (i.e., $F \rightarrow E$, $G \rightarrow E$, and $H \rightarrow D$). The virtual nodes are introduced to occupy the empty ranks. For the edge $F \rightarrow E$, rank three and four are empty, prompting the creation of virtual nodes (vn_2 and vn_3) as depicted in Fig. 9(b). The detailed explanation for generating virtual nodes is in Section 5.3.

Definition 8. (*Component*). *Let $G = (N, E)$ be a directed graph, $BB = (BN, BE)$ be a backbone, $p(G, u, v)$ be a function returning 1 if there exists a path between nodes u and v in G, otherwise 0. Let $G' = (N', E')$ be a directed graph where $N' = N \setminus \{bn \mid bn \in BN\}$ and $E' = \{(u, v), (v, u) \mid (u, v) \in E \setminus BE\}$. A component c in G is a subset of N' such that for all $u, v \in c$ with $u \neq v$, $p(G', u, v) = 1$.*

In this definition, edges in E' are treated as undirected by including both (u, v) and (v, u). This construction is intended to identify weakly connected components after removing the backbone, allowing nodes that are connected regardless of edge direction to be grouped into the same component. For example, in Fig. 9(b), let $BN = \langle A, B, C, D, E \rangle$ and $BE = \{(A,B), (B,C), (C,D), (D,E)\}$. Then, there are two components which are $\{F, G, v_1, v_2, v_3\}$ and $\{H, v_4\}$.

Definition 9. (*Cross-minimization*). *Let G be a directed graph, R be a set of node ranks, and O be a set of node orders. $cm(G, R, O) = O'$ is a function that returns a new set of node orders that minimizes the number of edge crossings.*

In this study, we employ the cross-minimization method proposed by Gansner et al. [27]. This method iteratively checks whether swapping two consecutive nodes in the same rank reduces the number of edge crossings; if so, it performs the swap. This procedure is repeated, rank by rank, until a specified maximum number of iterations is reached.

Definition 10. (*Median position of nodes* [27]). *Let $G = (N, E)$ be a directed graph, R be a set of node ranks, X be a set of x-coordinates, $\pi_r : N \rightarrow R$ be a node rank mapping function, and $\pi_x : N \rightarrow X$ be a x-coordinate mapping function. $medianpos(G, R, X, n) = median\{\pi_x(m) \mid ((n, m) \in E \vee (m, n) \in E) \wedge \pi_r(m) = \pi_r(n) - 1\}$.*

This function determines the horizontal placement of nodes based on the x-coordinates of their neighbors in the adjacent rank. The basic idea is to place a node near the horizontal positions of the connected nodes so that these connected nodes are closely aligned.

5. Method

To address the limitations discussed in Section 3, we propose an enhanced backbone-based process layout generation method based on the Sugiyama framework [25]. The Sugiyama framework includes several key steps: rank assignment, node ordering, node

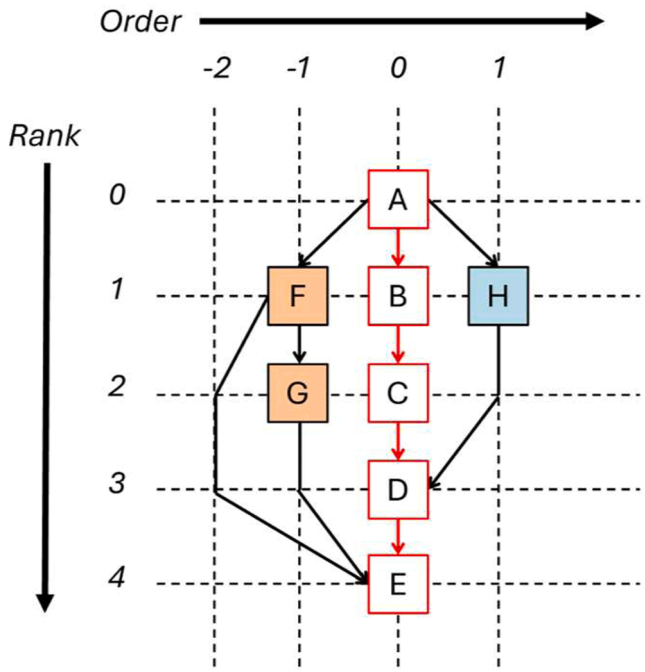


Fig. 8. Illustration of node rank and order in a process layout. Rank defines vertical placement, while order controls horizontal alignment.

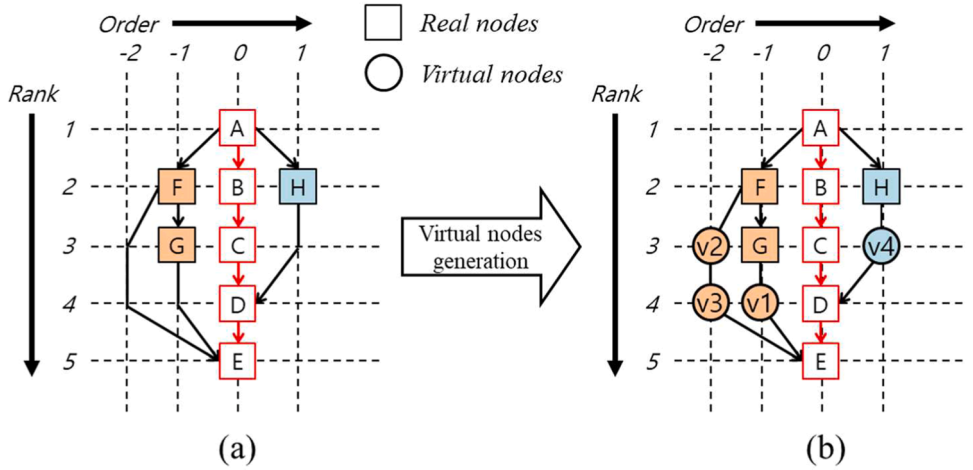


Fig. 9. (a) Process layout without virtual nodes. (b) Process layout with virtual nodes inserted to bridge rank gaps between real nodes.

positioning, and spline drawing. Building on this structure, our method involves the following steps: DFG generation from an event log, backbone determination, rank assignment, virtual node generation, component balancing, cross-minimization, and node positioning. Assuming an event log L is given, our method proceeds as follows:

1. DFG generation: A DFG G is derived from L in the DFG generation step (Section 5.1).
2. Backbone determination: A backbone BB is extracted from L during the backbone determination step (Section 5.2).
3. Rank assignment: A set of node ranks R is assigned using G and BB in the rank assignment step (Section 5.3).
4. Virtual node generation: A directed graph G^f, a backbone BB^f, a set of node ranks R^f with virtual nodes are constructed using G, BB, and R in the virtual node generation step (Section 5.4).
5. Component balancing: A set of node orders O is determined using G^f, BB^f in the component balancing step (Section 5.5).
6. Cross-minimization: A set of node orders O^f that minimizes the number of edge crossings is generated using G^f, BB^f, R^f, and O in the cross-minimization step (Section 5.6).
7. Node positioning: Sets of x-coordinates and y-coordinates of nodes X^f and Y^f are calculated using G^f, BB^f, R^f, and O^f in the node positioning step. Then, we obtain a backbone-based process layout BL (Section 5.7).

The following describes how each of the four limitations is handled within our method. Limitations 1 and 2 are addressed in the rank assignment step through integer programming. The objective function is designed to minimize the number of back edges and align mutually dependent nodes at the same rank. Limitation 3 is handled in the node positioning step by dividing the layout into three segments—left of the backbone, the backbone itself, and right of the backbone—and assigning node coordinates to preserve the backbone's linear structure. Limitation 4 is resolved in the component balancing step by minimizing the node count difference between the two sides of the backbone. In the following, we use toy process variants to explain the proposed method, as shown in Table 4.

5.1. DFG generation from the event log

Given an event log L, we construct a DFG $G = (N,E)$, where N is a set of nodes and E is a set of edges, as defined in Definition 4. Both the event log L and the derived DFG G are used in subsequent steps. Using the toy example in Table 4, we obtain the DFG $G = (N,E)$, where $N = \{Start, A, B, C, D, E, F, G, H, I, J, K, L, End\}$ and $E = \{(Start,A),(A,B),(B,C),(C,D),(D,E),(E,End),(C,F),(F,E),(B,G),(G,B),(B,H),(H,I),(I,D),(C,J),(J,K),(K,D),(D,L),(L,E)\}$.

Table 4
Toy process variants.

Variant	Frequency
$\langle A,B,C,D,E\rangle$	210
$\langle A,B,C,F,E\rangle$	90
$\langle A,B,G,B,C,D,E\rangle$	80
$\langle A,B,H,I,D,E\rangle$	70
$\langle A,B,C,J,K,D,L,E\rangle$	55

5.2. Backbone determination

In this step, we construct the backbone of the process using the most frequent variant as defined in Definition 11. The backbone nodes are the unique activities extracted from the most frequent variant, arranged in order of their first appearance. The backbone edges are edges between each pair of consecutive backbone nodes that exist in the event log.

Definition 11. (*Determining the backbone*). *Let L be an event* log, $G = (N, E)$ *be a DFG derived from L, and* $mv = \langle a_1, a_2, \ldots, a_n \rangle$ *be the most frequent variant in L.* $detBackbone(G, mv) = (BN, BE)$ *where* $BN = \langle b_1, \ldots, b_k \rangle = \langle a_i \mid 1 \le i < n \wedge a_i \notin \{a_1, \ldots, a_{i-1}\} \rangle$ *and* $BE = \{(b_j, b_{j+1}) \mid 1 \le j < k \wedge (b_j, b_{j+1}) \in E\}$.

In the toy example, the most frequent variant is $mv = \langle Start, A, B, C, D, E, End \rangle$. The backbone is determined as $detBackbone(G, mv) = (BN, BE)$ where $BN = \langle Start, A, B, C, D, E, End \rangle$ and $BE = \{(Start, A), (A, B), (B, C), (C, D), (D, E), (E, End)\}$.

5.3. Rank assignment

Node ranks are determined through the rank assignment. To address limitations 1 and 2, we formulate an integer programming model. We assume that the DFG $G = (N, E)$ and the backbone $BB = (BN, BE)$ are derived from an event log L using Definition 4 and Definition 11, respectively. As a result of this step, each node $n \in N$ is assigned an integer rank r_n. The rank assignment results for the toy example are shown in Fig. 10.

We introduce three types of decision variables: r_n, y_e, and α_e. The variable r_n is a positive integer variable representing the rank of node $n \in N$. The binary variable y_e is defined for each edge $e = (u, v) \in E$ and takes the value 1 if u is placed below v, i.e., $r_u - r_v \ge 1$. Thus, $y_e = 1$ indicates that edge e is a backward edge. The variable α_e is a non-negative integer measuring the severity of a precedence violation on edge $e = (u, v)$, that is, how far v is placed above u against the top-to-bottom (downward) flow. Therefore, $\alpha_e = 0$ when the edge is downward, $\alpha_e = 1$ when the edge is horizontal, and $\alpha_e > 1$ when the edge is upward. For the bidirectional edge pair (B, G) and (G, B) in Fig. 10, it is impossible to arrange both edges downward. The variable α_e captures these unavoidable violations. If $r_B = 3$ and $r_G = 4$, then $\alpha_{(B,G)} = 0$ (downward) while $\alpha_{(G,B)} = 2$ (upward), which means that for edge (G, B) the node B have to be shifted two ranks downward to make (I, B) downward.

The objective is to minimize the total precedence violations (i.e., upward edges) and the rank of the end node, as shown in Eq. (1).

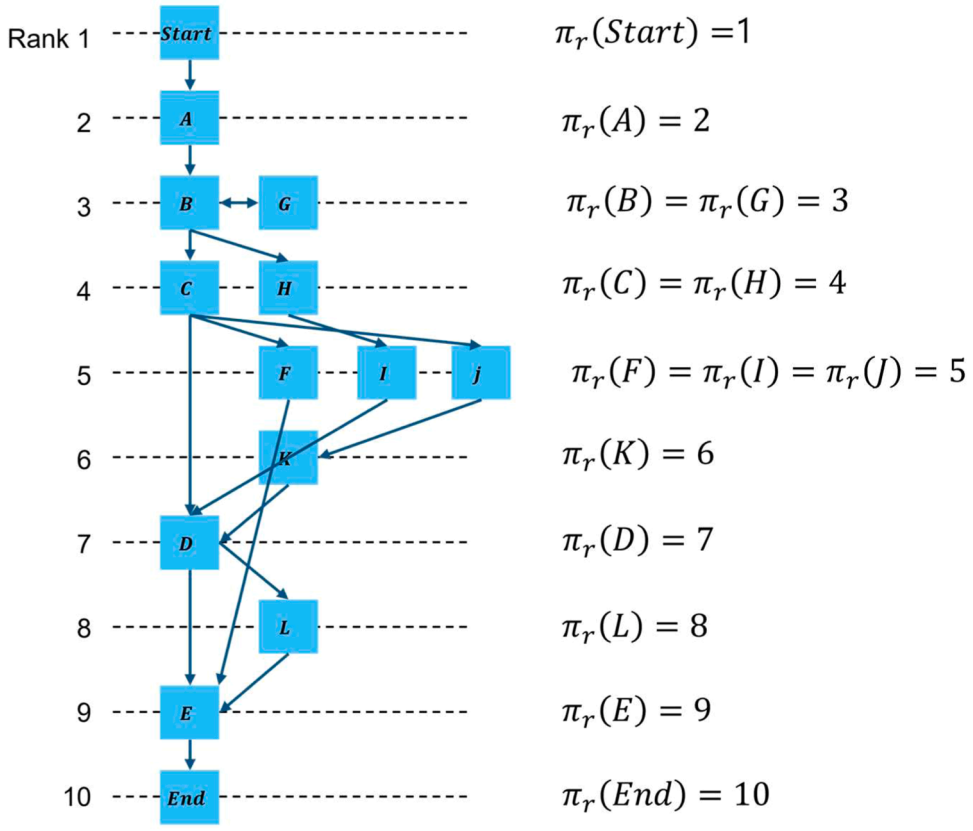


Fig. 10. Rank assignment results for the toy example.

Minimizing α_e enforces the intended top-to-bottom flow, while squaring α_e encourages nodes connected by bidirectional edges to be placed on the same rank. In the toy example, nodes B and G form a bidirectional pair. With a linear penalty, placing them at the same rank or at adjacent ranks yields the same cost. For instance, if $r_B = 3$ and $r_G = 2$, then $\alpha_{(B,G)} = 2$ and $\alpha_{(G,B)} = 0$, giving a total cost of 2 ($\alpha_{(B,G)} + \alpha_{(G,B)} = 2$). If $r_B = r_G = 3$, then $\alpha_{(B,G)} = \alpha_{(G,B)} = 1$, again yielding a cost of 2. In contrast, the quadratic penalty yields $\alpha^2_{(B,G)} + \alpha^2_{(G,B)} = 2^2 + 0^2 = 4$ versus $1^2 + 1^2 = 2$, thus preferring that the two nodes share the same rank. Finally, we minimize the rank of end node r_{end}. Without this term, the layout can be stretched vertically. For example, even if the rank of node *End* changes from 10 to 100 in Fig. 10, the objective $\sum_{e\in E}\alpha_e^2 = 2$ remains unchanged because the edge (E, End) is still downward. By including r_{end} in the objective, the value becomes $\sum_{e\in E}\alpha_e^2 + r_{end} = 2 + 10 = 12$ versus $2 + 100 = 102$. Therefore, the model selects the compact layout where $r_{end} = 10$.

$$Min \sum_{e\in E}\alpha_e^2 + r_{end} \tag{1}$$

Constraint 1 (Eq. (2)) ensures that the destination node v is placed below its source node u, up to the allowed violation captured by α_e. Constraint 2 (Eq. (3)) mandates a top-to-bottom ordering of the backbone nodes. In the toy example, the constraint $r_{Start} < r_A < r_B < r_C < r_D < r_E < r_{End}$ is imposed to preserve the vertical order of the backbone nodes $BN = \langle Start, A, B, C, D, E, End\rangle$.

Constraint 3 and 4 (Eqs. (4) and (5)) ensure that two nodes connected by an asymmetric edge cannot be placed on the same rank. An asymmetric edge (u, v) means that there exists a directed edge from u to v, but no reverse edge from v to u. In such a case, placing u and v on the same rank obscures the directional relationship between the two nodes. Without this restriction, nodes connected by asymmetric edges can be assigned to the same rank, and in the worst case, all nodes could collapse into a single rank. Therefore, for any asymmetric edge, the model enforces that the two nodes must be assigned to different ranks, i.e., $r_u \neq r_v$. Since integer programming does not directly support inequality constraint of $r_u \neq r_v$, this condition is implemented using Big-M formulation as Eqs. (4) and (5). If $y_{(u,v)} = 0$ (downward), constraints 3 and 4 become to $r_u - r_v \leq -1$ and $r_u - r_v \geq 1 - M$ (M is a large constant), which implies $r_u < r_v$. If $y_{(u,v)} = 1$ (upward edge), constraints 3 and 4 become to $r_u - r_v \leq M - 1$ and $r_u - r_v \geq 1$, which implies $r_u > r_v$. In both cases, $r_u = r_v$ is impossible. We choose the Big-M constant proportional to the number of nodes in the event log and set $M = 3|N|$ in our implementation. Although this is a Big-M formulation, M remains small in practice because process models typically contain a moderate number of activities.

Constraint 5 (Eq. (6)) anchors the start node at the top of the layout by forcing all other nodes to be placed below it, while constraint 6 (Eq. (7)) anchors the end node at the bottom by forcing all other nodes to be placed above it. As a result, minimizing r_{end} in the objective function directly leads to a vertical compression of the entire layout. At the same time, the layout does not collapse into a single rank, since constraints 3 and 4 enforce rank separation.

$$r_u + 1 \leq r_v + \alpha_e \ \forall e = (u, v) \in E \tag{2}$$

$$r_{bn_i} + 1 \leq r_{bn_{i+1}} \ \forall 1 \leq i < |BN| \tag{3}$$

$$r_u - r_v \leq -1 + My_e \ \forall e = (u, v) \in E : \ (v, u) \notin E \tag{4}$$

$$r_u - r_v \geq 1 - (1 - y_e)M \ \forall e = (u, v) \in E : \ (v, u) \notin E \tag{5}$$

$$r_n \geq r_{start} + 1 \ \forall n \in N - \{start\} \tag{6}$$

$$r_n + 1 \leq r_{end} \ \forall n \in N - \{end\} \tag{7}$$

After solving the integer programming, the rank of node r_n is obtained. In the toy example, we obtain the set of ranks $R = \{1, 2, 3, \ldots, 9, 10\}$ and mapping function as $\pi_r(Start) = 1$, $\pi_r(A) = 2$, $\pi_r(B) = \pi_r(G) = 3$, $\pi_r(C) = \pi_r(H) = 4$, $\pi_r(F) = \pi_r(I) = \pi_r(J) = 5$, $\pi_r(K) = 6$, $\pi_r(D) = 7$, $\pi_r(L) = 8$, $\pi_r(E) = 9$, and $\pi_r(End) = 10$. Let $R = \{r_n \mid n \in N\}$ denote the set of assigned ranks, and $\pi_r : N \to R$ be the rank mapping function that associates each node with its rank.

5.4. Virtual node generation

After assigning ranks, we insert virtual nodes whenever the ranks of an edge's endpoints differ by more than 1. These nodes fill the gap so that every edge in the resulting layout graph connects whose ranks are consecutive. As a result of this step, a backbone layout graph and layout graph including virtual nodes are generated.

Definition 12. (*Virtual node generation for backbone*). *Let $BB = (BN, BE)$ be a backbone where $BN = \langle n_1, \ldots, n_m\rangle$ and $BE \subseteq \{(n_i, n_{i+1}) \mid 1 \leq i < m\}$. Let R be a set of node ranks and $\pi_r : BN \to R$ be a mapping function. Let $insert(s, e, i)$ be a function that insert a sequence e at the index i of a sequence s and $idx(s, e)$ a function that returns the index of element e in a sequence s. $virgenBB(BB, R) = (BB', R')$ is defined as:*

1. $BN' = BN$, $BE' = BE$, and $R' = R$.
2. For $e = (u, v) \in BE$, $d_e = |\pi_r(u) - \pi_r(v)|$ and $\delta_e = sign(\pi_r(u) - \pi_r(v)) \in \{-1, 1\}$.
3. For $e = (u, v) \in BE$, if $d_e > 1$, $insert\big(BN', \langle vn_e^1, \ldots, vn_e^k\rangle, idx(BN', u)\big)$, $1 \leq k \leq d_e - 1$ and $BE' = BE' \cup \big\{\big(u, vn_e^1\big), \big(vn_e^{d_e-1}, v\big)\big\} \cup \big\{\big(vn_e^i, vn_e^{i+1}\big) \mid 1 \leq i < d_e - 1\big\} \setminus \{(u, v)\}$.

4. $BB' = (BN', BE')$
5. $R' = R' \cup \left\{ r_{vn_e^k} = \pi_r(u) + k \times \delta_e \mid e = (u,v) \in BE \wedge 1 \leq k < d_e \right\}$

Definition 13. (*Virtual node generation*). *Let $G = (N,E)$ be a DFG, $BB = (BN, BE)$ be a backbone of G, R be a set of node ranks, and $\pi_r : N \rightarrow R$ be a mapping function. virgen$(G, BB, R) = (G', BB', R')$ is defined as:*

1. $N' = N \setminus \{n \mid n \in BN\}$ and $E' = E \setminus BE$.
2. For $e = (u,v) \in E'$, $d_e = |\pi_r(u) - \pi_r(v)|$ and $\delta_e = sign(\pi_r(u) - \pi_r(v)) \in \{-1, 1\}$.
3. For $e = (u,v) \in E'$, if $d_e > 1$, $N' = N' \cup \{vn_e^1, \ldots, vn_e^k\}$, $1 \leq k \leq d_e - 1$ and $E' = E' \cup \{(u, vn_e^1), (vn_e^{d_e - 1}, v)\} \cup \{(vn_e^i, vn_e^{i+1}) \mid 1 \leq i < d_e - 1\} \setminus \{(u,v)\}$.
4. $R' = R \cup \left\{ r_{vn_e^k} = \pi_r(u) + k \times \delta_e \mid e = (u,v) \in E' \wedge 1 \leq k < d_e \right\}$.
5. $(BB', R'') = virgenBB(BB, R)$ where $BB' = (BN', BE')$ as Definition 12.
6. $N' = N' \cup \{bn \mid bn \in BN'\}$, $E' = E' \cup BE'$.
7. $G' = (N', E')$, $BB' = BB'$, and $R' = R' \cup R''$

From Section 5.1, 5.2, and 5.3, we already obtained the DFG, backbone, and the set of node ranks, i.e., $G = (N,E)$, $BB = (BN, BE)$, and R. Using these outputs, we obtain DFG, backbone, and the set of node ranks with virtual nodes as $(G', BB', R') = virgen(G, BB, R)$ as Definition 13.

First, we generate virtual nodes for the backbone using the toy example illustrated in Fig. 11(a). Two backbone edges (C,D) and (D,E) have rank differences greater than 1. Specifically, $d_{(C,D)} = |\pi_r(C) - \pi_r(D)| = |4 - 7| = 3$ and $d_{(D,E)} = |\pi_r(D) - \pi_r(E)| = |7 - 9| = 2$. Since both values are positive, we obtain $\delta_{(C,D)} = \delta_{(D,E)} = 1$, indicating both edges are downward. Because $d_{(C,D)} = 3$, two virtual nodes are inserted. Let $vn^1_{(C,D)}$ and $vn^2_{(C,D)}$ be denoted as v_1 and v_2, respectively. The updated backbone node sequence becomes $BN' = \langle Start, A, B, C, v_1, v_2, D, E, End \rangle$ and the set of backbone edges becomes $BE' = BE' \cup \{(C,v_1),(v_2,D)\} \cup \{(v_1,v_2)\} \setminus \{(C,D)\} = \{(Start,A),(A,B),(B,C),\ (D,E),(E,End),(C,v_1),(v_1,v_2),(v_2,D)\}$, where original edge (C,D) is replaced by (C,v_1), (v_1,v_2), and (v_2,D). The ranks of the virtual nodes are determined as $r_{v_1} = \pi_r(C) + 1 \times \delta_{(C,D)} = 4 + 1 = 5$ and $r_{v_2} = \pi_r(C) + 2 \times \delta_{(C,D)} = 4 + 2 = 6$. Similarly, since $d_{(D,E)} = 2$, one virtual node $vn^1_{(D,E)}$ denoted as v_3 is inserted, yielding $BN' = \langle Start, A, B, C, v_1, v_2, D, v_3, E, End \rangle$ and $BE' = BE' \cup \{(D,v_3),(v_3,E)\} \setminus \{(D,E)\} = \{(Start,A),(A,B),(B,C),(E,End),(C,v_1),(v_1,v_2),(v_2,D),(D,v_3),\ (v_3,E)\}$. The rank of the virtual node is determined as $r_{v_3} = \pi_r(D) + 1 \times \delta_{(D,E)} = 7 + 1 = 8$.

Then, we generate virtual nodes for non-backbone nodes and edges for the toy example as illustrated in Fig. 11(b). We obtain $N' = \{F, G, H, I, J, K, L\}$ and $E' = \{(C,F),(F,E),(B,G),(G,B),(B,H),(H,I),(I,D),(C,J),(J,K),(K,D),(D,L),(L,E)\}$. There are two edges (F,E) and (I,D) that $d_e > 1$.Since $d_{(F,E)}$ and $d_{(I,D)}$ are positive, we obtain $\delta_{(F,E)} = \delta_{(I,D)} = 1$, which means downward. Similar to virtual node generation for backbone, the virtual nodes and edges are added into the sets as $N' = \{F, G, H, I, J, K, L, M, v_4, v_5, v_6, v_7\}$ and $E' = \{(C,F),$

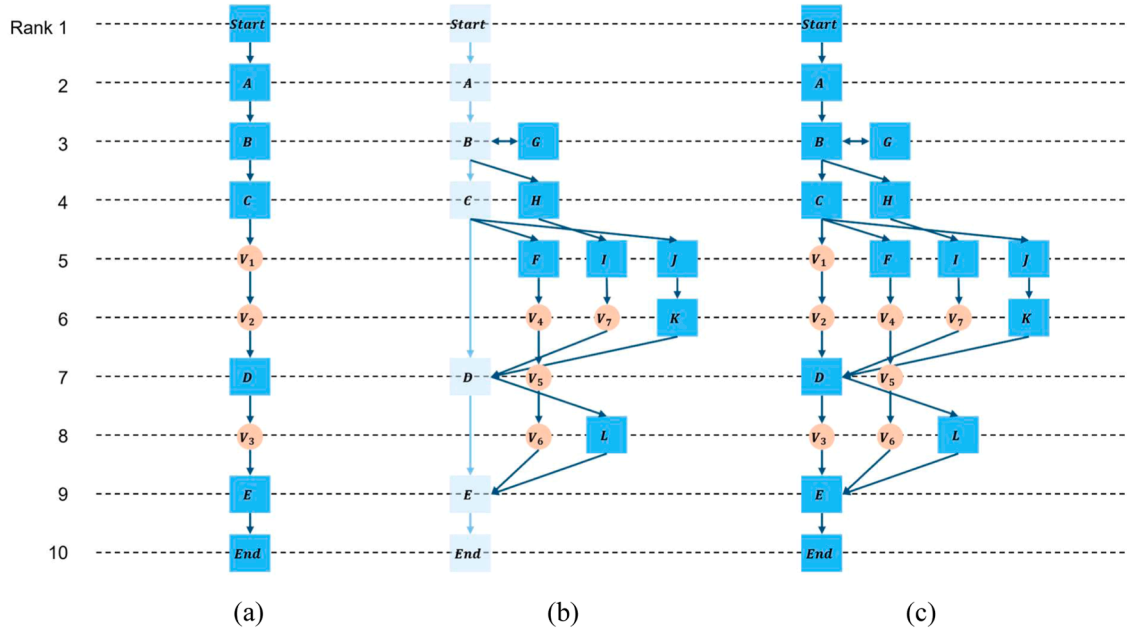


Fig. 11. (a) virtual node generation for backbone, (b) virtual node generation for non-backbone nodes, and (c) the final process layout with all virtual nodes.

$(F, v_4), (v_4, v_5), (v_5, v_6), (v_6, E), (B, G), (G, B), (B, H), (H, I), (I, v_7), (v_7, D), (C, J), (J, K), (K, D), (D, L), (L, E)\}$. The ranks of virtual nodes are determined as $\pi_r(v_4) = 6,\ \pi_r(v_5) = 7,\ \pi_r(v_6) = 8$, and $\pi_r(v_7) = 6$. Finally, we generate the new DFG with virtual nodes combining the two results as illustrated in Fig. 11(c).

5.5. Component balancing

This step positions the components on the left and right side of the backbone to distribute the nodes evenly using integer programming (addressing issue #4). From Section 5.4., we obtain G and BB, both including virtual nodes. Based on Definition 8, a set of components C is extracted from G with respect to BB. In the toy example, five components are identified: $\{F, v_4, v_5, v_6\}$, $\{G\}$, $\{H, I, v_7\}$, $\{J, K\}$, and $\{L\}$.

To balance these components, we formulate an integer programming model with two decision variables: d_c and y. The binary variable d_c indicates the placement of component $c \in C$, where $d_c = 1$ indicates that c is placed on the left side of the backbone and $d_c = 0$ indicates placement on the right side. The integer variable y represents the absolute difference between the number of nodes placed on the left and right sides.

The objective is to minimize the imbalance, defined as $|\sum_{c \in C: d_c = 1} size(c) - \sum_{c \in C: d_c = 0} size(c)|$, where $size(c)$ denotes the number of nodes contained in component c. Since integer programming cannot directly handle absolute values, we introduce the auxiliary variable y and minimize it subject to the linear constraints in Eqs. (9) and (10), which ensure that y correctly captures the difference in node counts between the two sides of the backbone.

$$Min\ y \tag{8}$$

$$y \geq \sum_{c \in C} size(c) \times d_c - size(c) \times (1 - d_c)\ \forall c \in C \tag{9}$$

$$y \geq -\sum_{i=1}^{|C|} size(c) \times d_c - size(c) \times (1 - d_c)\ \forall c \in C \tag{10}$$

After solving the integer programming, the component direction d_c is determined for every component $c \in C$. In the toy example, the component sizes are as follows: $size(\{F, v_4, v_5, v_6\}) = 4$, $size(\{G\}) = 1$, $size(\{H, I, v_7\}) = 3$, $size(\{J, K\}) = 2$, and $size(\{L\}) = 1$. Consider the feasible solution shown in Fig. 12(a), where only component $\{G\}$ is placed on the left side and all other components are placed on the right side. In this case, the numbers of nodes on the left and right sides becomes 1 and 10, respectively, resulting in $y = 9$. Although this assignment satisfies all constraints, it is not optimal, and the integer programming solver continues searching for a better

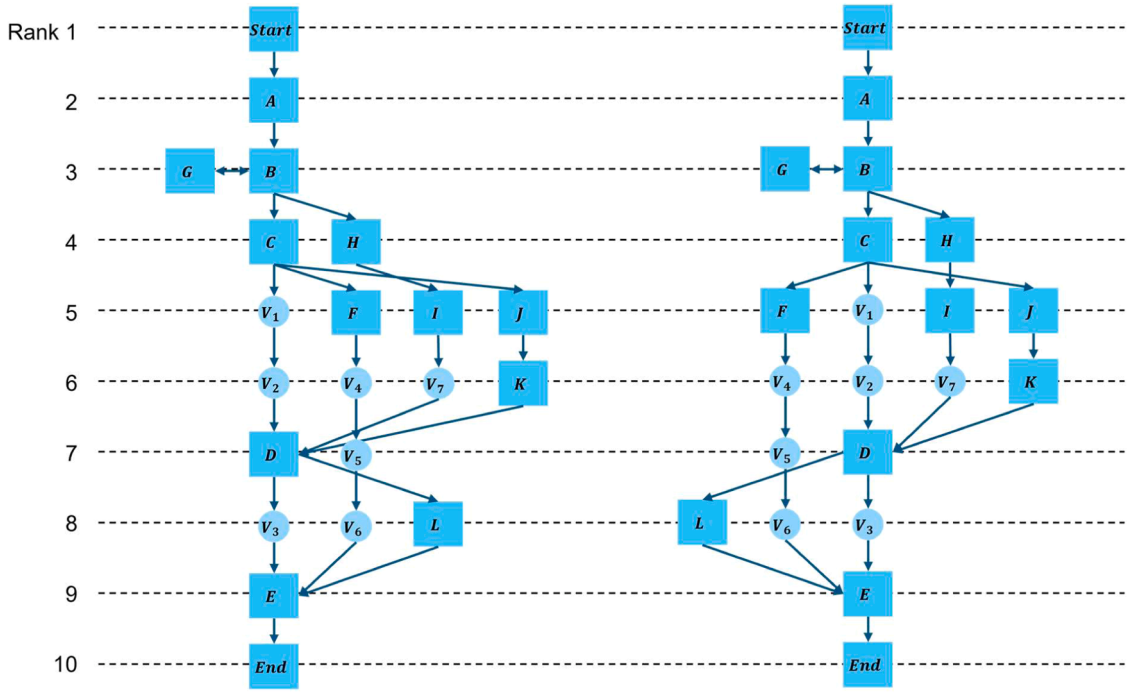


Fig. 12. (a) A feasible but not optimal solution for component balancing in the toy example. (b) An optimal solution in which the difference between the numbers of nodes on the left and right sides of the backbone is one.

balance between two sides. An optimal solution is presented in Fig. 12(b). In this solution, the components $\{F, v_4, v_5, v_6\}$, $\{G\}$ and $\{L\}$ are assigned to the left side, while $\{H, I, v_7\}$ and $\{J, K\}$ are assigned to the right. This yields 6 nodes on the left and 5 nodes on the right, resulting in $y = 1$. The corresponding direction values are $\pi_d\{F, v_4, v_5, v_6\} = 1$, $\pi_d(\{G\}) = 1$, $\pi_d(\{H, I, v_7\}) = 0$, $\pi_d(\{J, K\}) = 0$, and $\pi_d(\{L\}) = 1$. In the following, we denote $\pi_d : C \to \{0, 1\}$ as the component direction mapping function.

Definition 14. (*Initial order assignment*). *Let $G = (N, E)$ be a DFG, $BB = (BN, BE)$ be a backbone of G, R be a set of node ranks, C be a set of components in G, $\pi_r : N \to R$ is a node rank mapping function, $\pi_d : C \to \{0, 1\}$ be a component direction mapping function, and $idx(s, e)$ a function that returns the index of element e in a sequence s. $initOrder(G, BB, C) = O$ is defined as:*

1. $k = \max(\{\pi_r(n) \mid n \in N\})$.
2. $LN_a = \langle n_1, \ldots, n_k \rangle = \langle n \in N \mid \exists c \in C : n \in c \wedge \pi_d(c) = 1 \wedge \pi_r(n) = a \rangle \ \forall a \in \{1, \ldots, k\}$.
3. $RN_a = \langle n_1, \ldots, n_l \rangle = \langle n \in N \mid \exists c \in C : n \in c \wedge \pi_d(c) = 0 \wedge \pi_r(n) = a \rangle \ \forall a \in \{1, \ldots, k\}$.
4. For $n \in N$, $o_n = \begin{cases} 0 & \text{if } n \in BN \\ -idx(LN_{\pi_r(n)}, n) & \text{if } n \in LN_{\pi_r(n)} \\ idx(RN_{\pi_r(n)}, n) & \text{if } n \in RN_{\pi_r(n)} \end{cases}$.
5. $O = \{o_n \mid n \in N\}$.

Then, we initialize the node orders according to Definition 14. In the definition, LN_a and RN_a denote the sequences of left and right nodes positioned at rank a. In the toy example, the left node sequences are $LN_1 = LN_2 = LN_4 = LN_9 = LN_{10} = \langle\ \rangle$, $LN_3 = \langle G \rangle$, $LN_5 = \langle F \rangle$, $LN_6 = \langle v_4 \rangle$, $LN_7 = \langle v_5 \rangle$, and $LN_8 = \langle v_6, L \rangle$. The right node sequences are $RN_1 = RN_2 = RN_3 = RN_7 = RN_8 = RN_9 = RN_{10} = \langle\ \rangle$, $RN_4 = \langle H \rangle$, $RN_5 = \langle I, J \rangle$, and $RN_6 = \langle v_7, K \rangle$. Based on these sequences, the initial node order o_n is computed for each node $n \in N$, resulting in the layout shown in Fig. 13. The backbone nodes *Start*, *A*, *B*, *C*, *D*, *E*, *and End* are assigned order of zero. For left-side nodes, the order equals the negative index of the node in the corresponding *LN* sequence, whereas for right-side nodes it equals the positive index in the *RN*. For example, v_6 is the first element in $LN_8 = \langle v_6, L \rangle$, its order is $o_{v_6} = -1$, and since L is the second element, its order is $o_L = -2$. Similarly, because I and J is the first and second elements in RN_5, their orders are $o_I = 1$ and $o_J = 2$, respectively. In the following, we denote O as the set of node orders of $G = (N, E)$ and $\pi_o : N \to O$ as the node order mapping function.

5.6. Cross-minimization

In this step, the number of edge crossings is minimized. In this study, we extend the edge cross- minimization method proposed by Gansner et al. [27]. Using $G = (N, E)$, $BB = (BN, BE)$, and R are from the Section 5.4, and O from the Section 5.5, we minimize the number of edge crossings existing in G.

Definition 15. (*Backbone-aware cross-minimization*). *Let $G = (N, E)$ be a DFG, $BB = (BN, BE)$ be a backbone of G, R be a set of node ranks,*

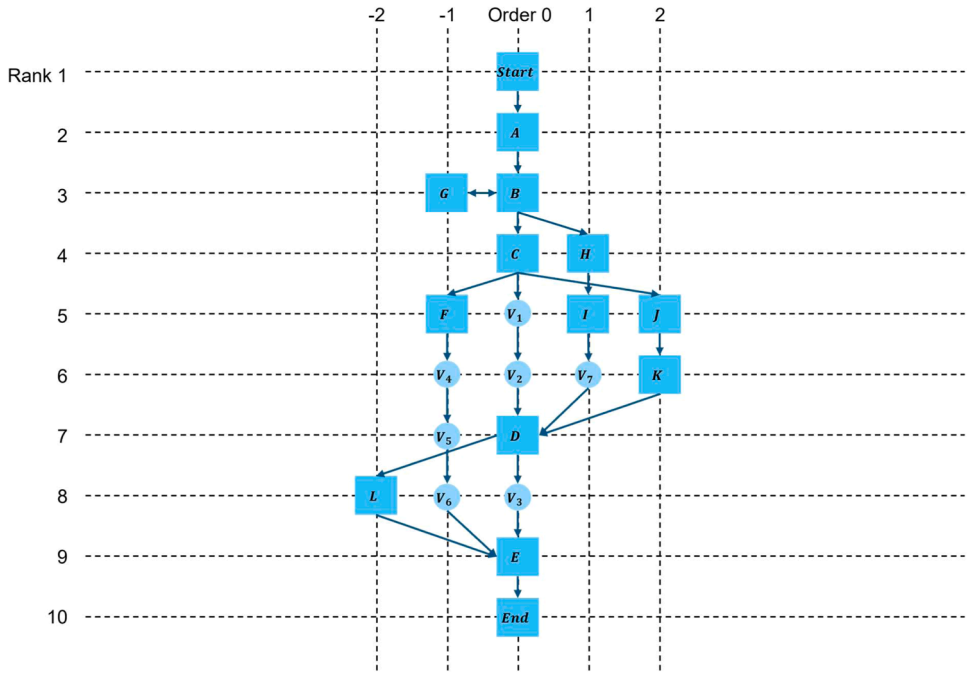


Fig. 13. Initial order assignment of nodes after component balancing. Backbone nodes are assigned order zero, while non-backbone nodes are assigned positive or negative orders based on their indices in the corresponding right and left node sequences, respectively.

O be a set of node orders, π_r : N→R be a node rank mapping function, π_o : N→O be a node order mapping function, and $cm(G,R,O) = O'$ be a cross-minimization function for a directed graph. $bcm(BL) = O'$ is defined as:

1. $N_L = \{n \in N \mid \pi_o(n) < 0\}$ and $N_R = \{n \in N \mid \pi_o(n) > 0\}$.
2. $E_L = \{(u,v) \in E \mid u \in N_L \vee v \in N_L\}$ and $E_R = \{(u,v) \in E \mid u \in N_R \vee v \in N_R\}$.
3. $G_L = (N_L, E_L)$ and $G_R = (N_R, E_R)$.
4. $R_L = \{r_n \mid n \in N_L\}$ and $R_R = \{r_n \mid n \in N_R\}$.
5. $O_L = \{o_n \mid n \in N_L\}$ and $O_R = \{o_n \mid n \in N_R\}$.
6. $O' = cm(G_L, R_L, O_L) \cup \{\pi_o(bn) \mid bn \in BN\} \cup cm(G_R, R_R, O_R)$.

In the toy example, the left-side subgraph consists of $N_L = \{F, G, v_4, v_5, v_6, L\}$ and $E_L = \{(G,B), (B,G), (F,v_4), (v_4,v_5), (v_5,v_6), (v_6,E), (D,L), (L,E)\}$. The right-side subgraph consists of $N_R = \{H,I,v_7,J,K\}$ and $E_R = \{(B,H), (H,I), (I,v_7), (v_7,D), (C,J), (J,K), (K,D)\}$. Using these subgraphs with their node ranks and orders, the crossing minimization is performed independently on the left and right graphs. The algorithm iteratively examines adjacent nodes within the same rank and swaps them if the exchange reduces the number of edge crossings between consecutive ranks. As shown in Fig. 14, ranks 3 to 7 in the left graph contain a single node each, and thus provide no adjacent node pairs to swap. At rank 8, two nodes v_6 and L exist. Swapping them decreases the number of edge crossings between ranks 7 and 8 from one to zero, and the swap is accepted. A similar procedure is applied to the right graph (Fig. 15). Ranks 1 to 4 contain only one node and are skipped. At rank 5, exchanging I and J reduces the crossings between ranks 4 and 5 from one to zero. Subsequently, at rank 6, swapping v_7 and K further reduces the crossings between ranks 5 and 6 from one to zero. After completing cross-minimization, the updated node orders are combined with the backbone to generate the final layout.

5.7. Node positioning

We propose a heuristic node positioning method tailored to backbone-based process layouts, based on the method introduced by Gansner et al. [27] to solveissue #3. The layout method proposed by Mennens et al. [23] also utilized the approach of Gansner et al. [27]. However, their method does not consider the backbone, leading to an inability to maintain the backbone in a linear arrangement. To address this limitation, we propose a heuristic node positioning method to ensure the backbone remains linear.

Definition 16. (*Initializing the initial x and y coordinates of nodes*). *Let $G = (N,E)$ be a DFG, R be a set of node ranks, O be set of node orders, h be a vertical distance between adjacent ranks, w be a horizontal distance between adjacent orders, π_r : N→R be a node rank mapping*

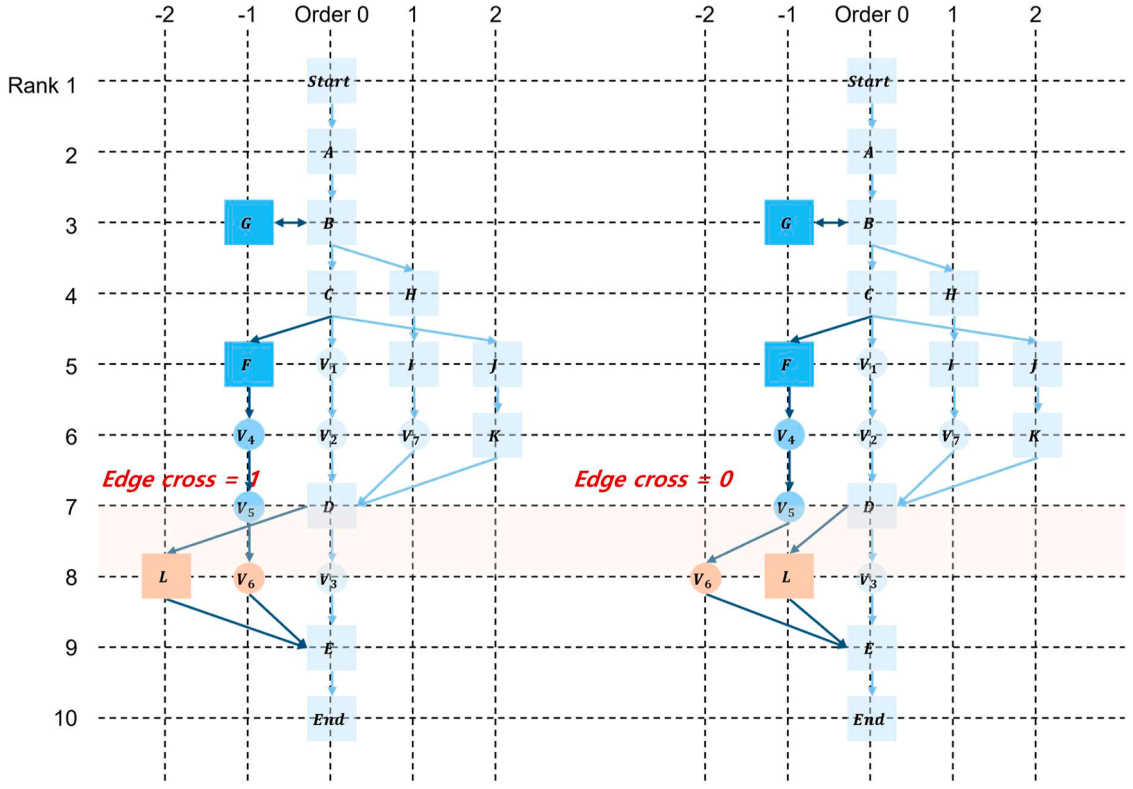


Fig. 14. Cross-minimization for the left subgraph. Ranks from 3 to 7 contain only a single node and are therefore skipped. At rank 8, swapping two nodes $\boldsymbol{v_6}$ and $\boldsymbol{M}$ recudecs the number of edge crossings between rank 7 and 8; thus, the swap is executed.

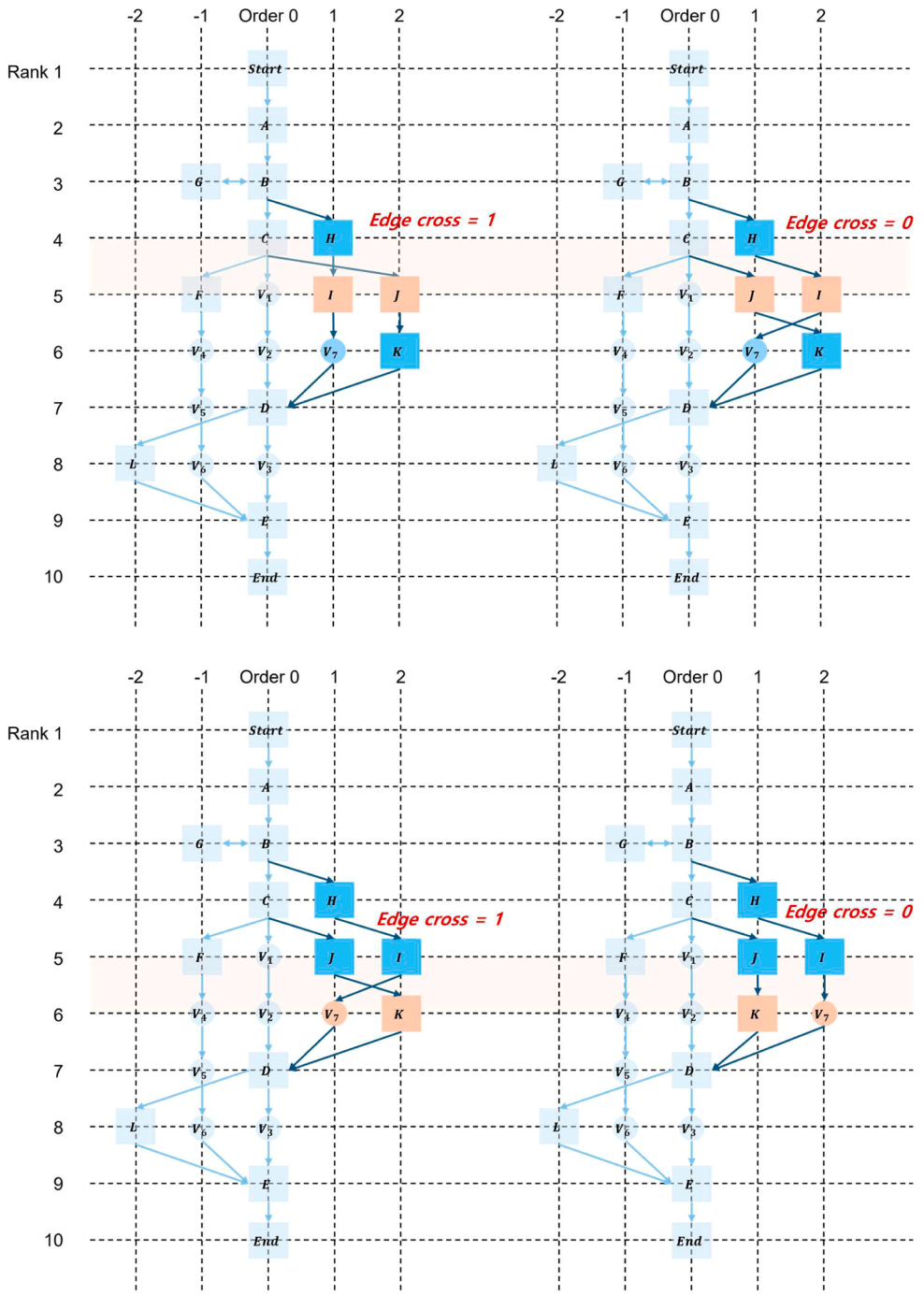


Fig. 15. Cross-minimization for the right subgraph. At rank 5, swapping nodes ***K*** and ***L*** reduces the crossings between ranks 4 and 5 from one to zero. After that, at rank 6, swapping $\boldsymbol{v_7}$ and ***H*** further reduces the crossings between ranks 5 and 6 from one to zero.

function, and $\pi_o : N \to O$ *be a node order mapping function.* $initcoord(G, R, O, w, h) = (X, Y)$, *where* $X = \{x_n \mid x_n = w \times \pi_o(n)\}$ *and* $Y = \{y_n \mid y_n = h \times \pi_r(n)\}$.

Definition 17. (*Initializing the width of nodes*). *Let* $G = (N, E)$ *be a DFG,* $len : \cup_{act} \to \mathbb{Z}$ *be a function that returns the length of node names, and* u *be the length for each text unit.* $initwidth(G, u) = W$, *where* $W = \{w_n \mid w_n = len(n) \times u \wedge n \in N\}$.

Before positioning nodes, we initialize the x and y coordinates of nodes using the node orders as defined in Definition 16. As a result, we obtain the set of x-coordinates and y-coordinates of nodes as $(X, Y) = initcoord(G, R, O, w, h)$ where $G = (N, E)$ and R are from

Section 5.4, O is from Section 5.6, w and h are given by the user. Also, we initialize the node widths through $W = initwidth(G,u)$, where u is given by the user. Fig. 16 illustrates the result of coordinate initialization for the toy example, where we assume that $w = h = 100$. For instance, the node *Order* has rank 2 and order 0, resulting in coordinates $(x,y) = (0,200)$. Throughout the toy example, we use the following labels: *A: Order, B: Validate, C: Pay Authorize, D: Ship Confirm, E: Complete, F: Pick Prep, G: Clarify, H: Ship Option, I: Dispatch, J: Pack, K: Inspect,* and *L: Exception.* Node widths are determined based on these labels.

Definition 18. (*Left node function*). *Let $G = (N,E)$ be a directed graph, R be a set of node ranks, O be a set of node orders, $\pi_r : N \to R$ be a node rank mapping function, and $\pi_o : N \to O$ be a node order mapping function. The left-node function $left(G,R,O,n) = m$ returns the node $m \in N$ such that $\pi_r(m) = \pi_r(n)$ and $\pi_o(m) = \pi_o(n) - 1$.*

That is, $left(G,R,O,n)$ identifies the immediate left neighbor of node n on the same rank. For example, $left(G,R,O,'Exception')$ returns the node v_6 in the example.

Definition 19. (*Packcut*). *Let $G = (N,E)$ be a directed graph, X be a set of x-coordinates, W be a set of node widths, ϵ be an allowed horizontal gap between two nodes, $\pi_x : N \to X$ be a x-coordinate mapping function, and $\pi_w : N \to W$ be a node width mapping function. $packcut(G,X,W,\epsilon) = X'$ is defined as:*

1. *$A = \langle n_1,\ldots,n_k \rangle$ where $\{n_1,\ldots,n_k\} = N$ and $\pi_x(n_1) \le \pi_x(n_2) \le \ldots \le \pi_x(n_k)$.*
2. *Define $X' = X$ and $\pi'_x : N \to X'$.*
3. *For all $i \in \{1,\ldots,k-1\}$,*
   - *$d = \pi'_x(n_{i+1}) - \pi'_x(n_i)$.*
   - *If $d > 0$, $x'_{n_j} = \max(\pi'_x(n_j) - d, \pi_x(ln) + (\pi_w(ln) + \pi_w(n_j))/2) + \epsilon)$ for all $j \in \{i+1,\ldots,k\}$, $ln = left(G,R,O,n_j)$.*

Using the packcut function, we first position nodes on the left side of the backbone. The result of applying packcut to the left-side nodes of the toy example is illustrated in Fig. 17. In the example, the sorted node sequence is $A = \langle v_6, Clarify, Pick\ Prep, v_4, v_5, Exception \rangle$. For nodes v_6 and *Clarify*, the horizontal distance is $d = \pi_x(Clarify) - \pi_x(v_6) = (-100) - (-200) = 100$. Since the nodes from *Clarify* to v_5 have no left neighbors on the same rank, the x-coordinates are shifted left by d, resulting in $x_n = \pi_x(n) - d = -100 - 100 = -200$ for node $n \in \{Clarify, v_4, v_5\}$. However, the node *Exception* has the left node v_6. Therefore, its updated position is constrained by the left node boundary: $x_{Exception} = \max(\pi_x(Exception) - d, \pi_x(v_6) + (\pi_w(v_6) + \pi_w(Exception))/2 + \epsilon)$. As a result, *Exception* is positioned to the right of v_6, maintaining a minimum horizontal separation of ϵ.

Definition 20. (*Packcut for backbone nodes*). *Let $G = (N,E)$ be a directed graph, $BB = (BN,BE)$ be a backbone, X be a set of x-coordinates,*

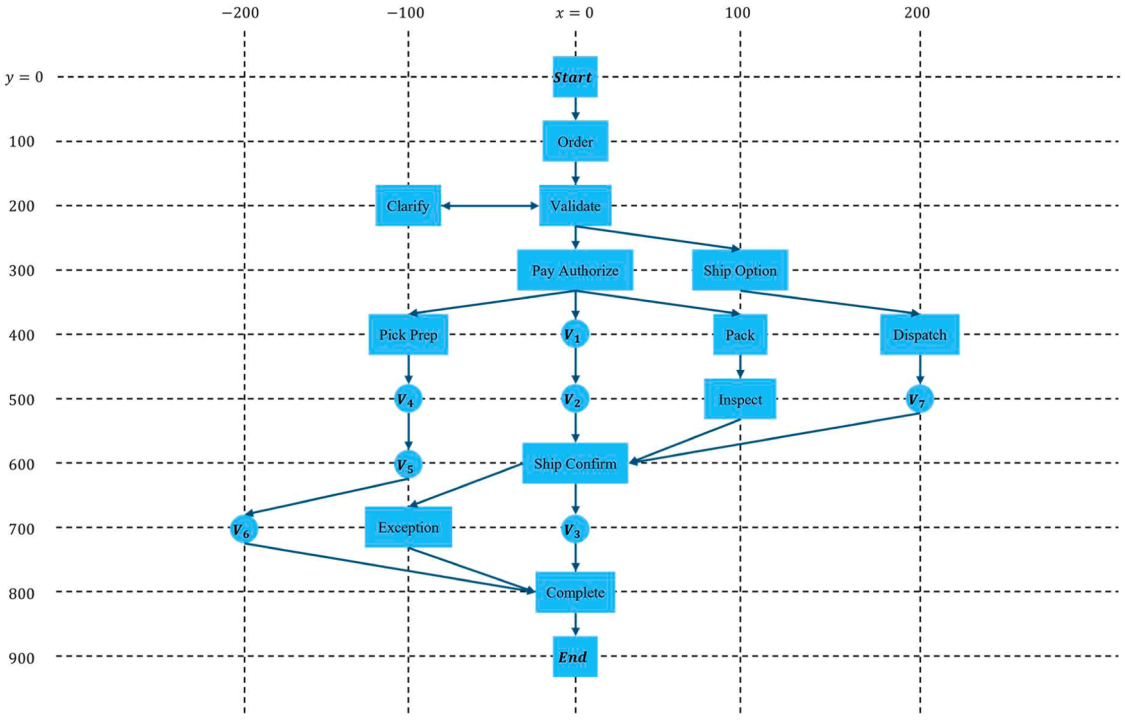


Fig. 16. Initial node positioning based on node ranks and orders. Node coordinates are initialized using user-defined horizontal and vertical spacings ($\boldsymbol{w} = \boldsymbol{h} = 100$), and node widths are determined from label lengths.

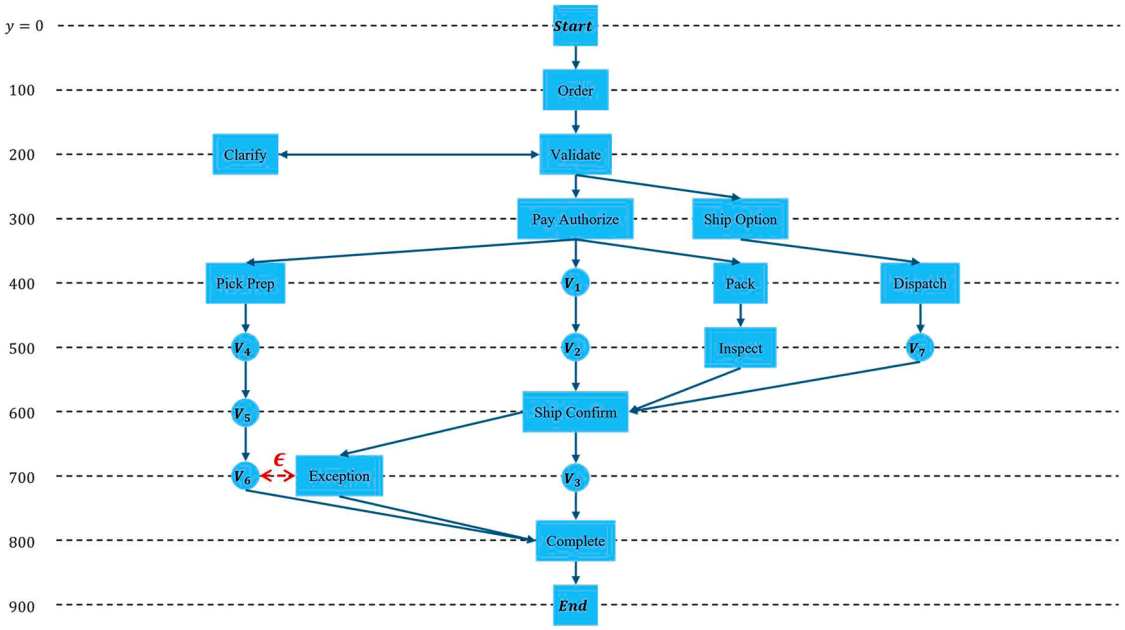


Fig. 17. Result of applying packcut to nodes on the left side of the backbone.

W be a set of node width, $\pi_x : N \to X$ be a x-coordinate, $\pi_w : N \to W$ be a node width mapping function, mapping function, $left(G, R, O, n)$ be a left node mapping function, and ϵ be the allowed horizontal gap between two nodes. $packcutBN(G, BB, X, W) = X^f$ is defined as:

1. For all $bn \in BN$, the left margin of backbone nodes is computed as $lm(bn) = \begin{cases} \pi_x(n) + \frac{\pi_w(n)}{2} + \epsilon, & \text{if } n = left(G, R, O, bn) \text{ exists} \\ -\infty, & \text{otherwise} \end{cases}$.
2. For all $bn \in BN$, $x'_{bn} = \max\{lm(bn) \mid bn \in BN\} + \pi_w(bn)/2$.
3. $X^f = \{x'_{bn} \mid bn \in BN\}$.

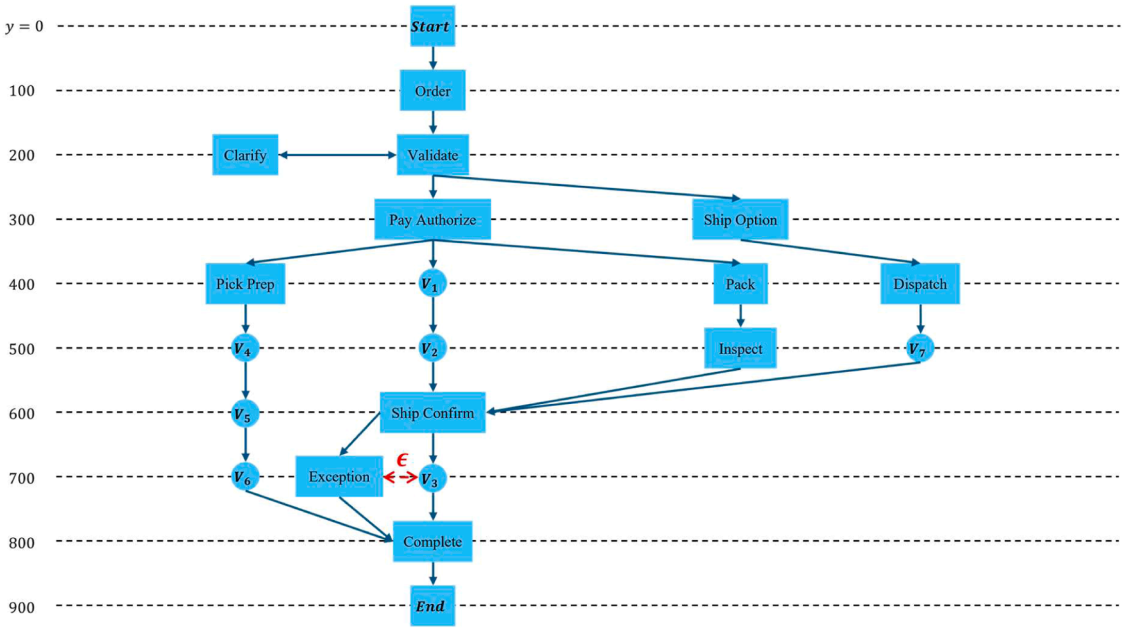


Fig. 18. Result of applying packcut to backbone nodes. All backbone nodes are uniformly shifted according to the maximum left-margin constraint, preserving the linear structure of the backbone while avoiding overlaps with non-backbone nodes.

After shrinking the layout on the left side of the backbone, we shift the backbone leftward to eliminate unnecessary horizontal whitespace. For this, we calculate the left margin for each backbone node and move the backbone uniformly based on the largest left margin value. As shown in Fig. 18, the backbone nodes *Start, Order, Complete*, and *End* do not have left neighbors on the same rank; therefore, their left margin values are set to $-\infty$. For backbone nodes with left neighbors, the left margin is computed using the x-coordinate and width of the left node. For instance, $lm(Validate) = \pi_x(Clarify) + \frac{\pi_w(Clarify)}{2} + \epsilon$. Among all backbone nodes, the largest left margin value is obtained from node v_3, whose left neighbor *Exception* is positioned farthest to the right. Consequently, all backbone nodes are shifted uniformly so that the node v_3 is placed ϵ to the right of its left neighbor, *Exception*. This preserves the backbone's linear structure while avoiding overlaps with non-backbone nodes. After processing the left side and the backbone, the same packcut procedure is applied to the right side of the backbone, as illustrated in Fig. 19.

Definition 21. (*Node positioning*). *Let $G = (N, E)$ be a DFG, R be a set of node ranks, O be a set of node orders, W be a set of node width, w be the horizontal distance between adjacent orders, h be the vertical distance between adjacent ranks, and ϵ be the allows horizontal gap between two nodes. $positionNode(G, R, O, W, w, h) = (X^f, Y^f)$ is defined as:*

1. $(X, Y) = initcoord(G, R, O, w, h)$ and let $\pi_x : N \rightarrow X$ be a x-coordinate mapping function.
2. $nonBN = N \setminus BN$.
3. $X'_{med} = \{medianpos(G, R, X, n) \mid n \in nonBN\}$ and $X' = \{\pi_x(bn) \mid bn \in BN\} \cup X'_{med}$.
4. $N_L = \{n \in N \mid \pi_o(n) < 0\}$ and $E_L = \{e \mid e = (u, v) \in E \wedge u, v \in N_L\}$.
5. $N_R = \{n \in N \mid \pi_o(n) > 0\}$ and $E_R = \{e \mid e = (u, v) \in E \wedge u, v \in N_R\}$.
6. $G_L = (N_L, E_L)$ and $G_R = (N_R, E_R)$.
7. $X'_L = packcut(G_L, X', W, \epsilon)$ and $X' = \{x_n \mid n \in N \wedge \pi_o(n) \geq 0\} \cup X'_L$.
8. $X'_{BB} = packcutBN(G, BB, X', W, \epsilon)$ and $X' = \{x_n \mid n \in N \wedge \pi_o(n) \neq 0\} \cup X'_{BB}$.
9. $X'_R = packcut(G_R, X', W, \epsilon)$ and $X' = \{x_n \mid n \in N \wedge \pi_o(n) \leq 0\} \cup X'_R$.
10. For $i \in \{1, \ldots, T\}$, repeat the steps from 3 to 9, where T is the given iteration number.
11. $X^f = X'$ and $Y^f = Y$.

Conceptually, node positioning places nodes at the median position of x-coordinates of adjacent nodes positioned in the prior rank. Then, the graph is divided into three parts, which are the left of the backbone, the backbone, and the right of the backbone. Then packcuts are executed for each part sequentially. And this procedure is repeated until the given iteration is met. We position the nodes using $G = (N, E)$ and R are from Section 5.4, O is from Section 5.6, W from Section 5.7, w and h are given by users. Then, we obtain the x and y coordinates of nodes as $(X, Y) = positionNode(G, R, O, W, w, h)$. In this study, we fixed the vertical distance between adjacent ranks $h = 150$ and the horizontal distance between adjacent orders $w = 200$. The length of each text unit was fixed $u = 20$, and the allowed horizontal gap between two nodes was set to ϵ as 100.

Definition 22. (*Backbone-based layout*). *A backbone-based layout is a tuple $BL = (N, E, BN, BE, R, O, X, Y, \pi_r, \pi_o, \pi_x, \pi_y)$ where:*

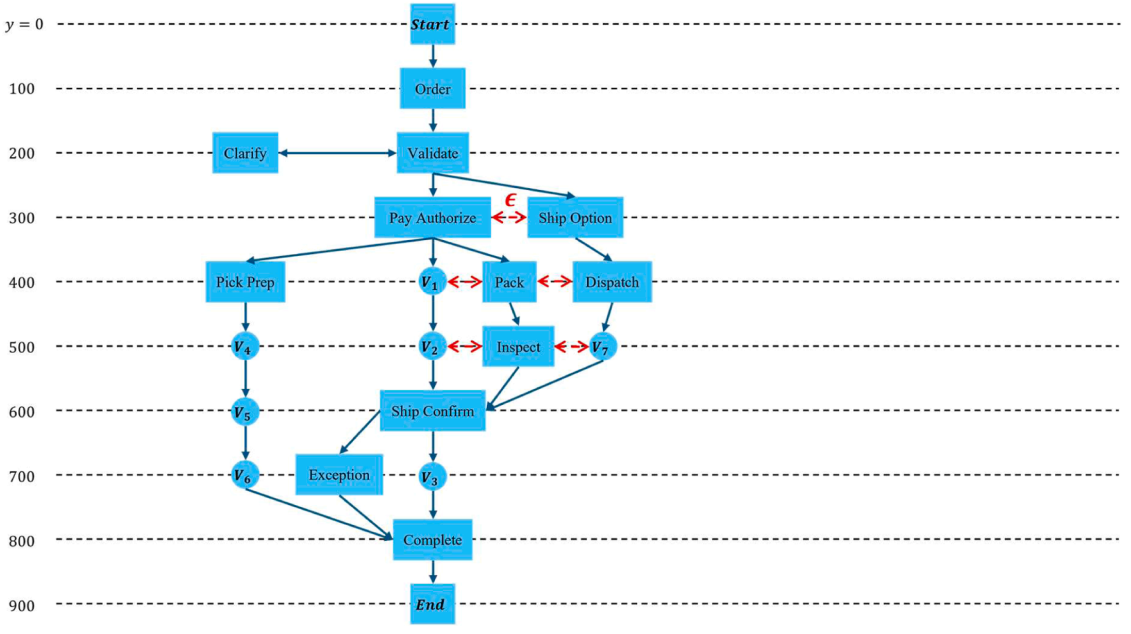


Fig. 19. Result of applying packcut to nodes on the right side of the backbone. Non-backbone nodes are compacted toward the backbone while maintaining left–right ordering constraints and minimum horizontal separation.

- $N \subseteq \mathbb{U}_{act}$ is a set of nodes.
- $E \subseteq N \times N$ is a set of edges.
- BN is a sequence of backbone nodes.
- BE is a set of backbone edges.
- R is a set of node ranks.
- O is a set of node orders.
- X is a set of x-coordinate of nodes.
- Y is a set of y-coordinate of nodes.
- $\pi_r : N \rightarrow R$ associates a node to each rank.
- $\pi_o : N \rightarrow O$ associates a node to each order.
- $\pi_x : N \rightarrow X$ associates a node to each x-coordinate.
- $\pi_y : N \rightarrow Y$ associates a node to each x-coordinate.

In summary, the backbone determination produces backbone nodes and edges BN and BE. The rank assignment defines the rank set R and the mapping π_R. The virtual node generation updates the set of nodes and edges with virtual nodes. The component balancing determines the node order set O and the mapping π_o. The node positioning produces the coordinates sets X and Y with the mappings π_x and π_y. Consequently, the backbone-based process layout is defined as $BL = (N, E, BN, BE, R, O, X, Y, \pi_r, \pi_o, \pi_x, \pi_y)$.

6. Evaluation & results

This section presents the quantitative and qualitative evaluation of the proposed method. We compare our method with the backbone-based process layout generation method of Mennens et al. [23], which we adopt as the benchmark. Although layout design plays a crucial role in process model comprehension, relatively little research has focused on optimizing layouts in process mining. Among existing backbone-based approaches, the benchmark is the only one with a detailed and reproducible description publicly available. In addition, general graph layout algorithms primarily emphasize connectivity and crossing reduction and do not explicitly preserve top-down flow semantics, which are essential for process interpretation. For these reasons, we select Mennens et al. [23] as the benchmark.

6.1. Quantitative evaluation

We conduct quantitative evaluation using five event logs and six layout metrics. The results indicate that our method shows good performance in five metrics compared to the benchmark.

6.1.1. Event logs

This study utilizes five distinct event logs, as summarized in Table 5. The first event log, BPI 2017, originates from the 2017 BPI Challenge and contains data related to a personal loan application process [42]. The BPI 2018 captures the EU's direct payment process to German farmers under the Common Agricultural Policy [43]. BPI 2019 focuses on a purchase-to-pay process involving various goods and services procured from multiple vendors [44]. The fourth log records user interaction data on a website, reflecting browsing behavior and page transitions. The final log captures administrative processes in a hospital environment. To evaluate our method, we conducted 25 experiments across these logs, applying five different variant filtering levels: 100%, 80%, 60%, 40%, and 20%.

6.1.2. Metrics

To select appropriate quantitative evaluation metrics, we reviewed prior studies focusing on process model layout evaluation as shown in Table 1. Then, we selected six key metrics: 1) the number of back edges, 2) balance degree (symmetry), 3) the number of edge crossings, 4) edge length, 5) edge orthogonality, and 6) node orthogonality. We excluded edge bends from consideration because edge orthogonality sufficiently captures the degree of edge bends.

Definition 23. (*Metrics*). *Let* $BL = (N, E, BN, BE, R, O, X, Y, \pi_r, \pi_o, \pi_x, \pi_y)$ *be a backbone-based layout. Letec* $: E, E \rightarrow \{0, 1\}$ *be a function that determines whether the given two edges are crossing or not,* $deg(v_i, v_j)$ *be a function that returns the degree between two vectors, and* $\pi_{real} : N \rightarrow \{0, 1\}$ *be a mapping function that checks whether the given node is real or not. We define the following metrics on BL:*

Table 5
Event log description.

Event log	Description	Case	Event	Variant	Activity	Edge
BPI 2017	Event log for personal loan application	31,509	1202,267	15,930	26	178
BPI 2018	Event log for payment process of common agricultural policy	43,809	2514,266	28,489	41	592
BPI 2019	Event log for purchase order handling process	251,734	1595,923	11,973	42	517
Hospital log	Event log of hospital administration process	5626	46,874	2000	20	250
Web log	Event log of events related to users' web page movement	285	10,000	1016	41	173

- QM_{be}: The number of back edges $QM_{be} = |\{e \mid e = (u,v) \in E \wedge \pi_r(u) \rangle \pi_r(v)\}|$. This metric assesses how consistently the layout represents the process flow.
- QM_{bal}: Balance degree $QM_{bal} = (abs(|\{n \in N \mid \pi_o(n) < 0\}| - |\{n \in N \mid \pi_o(n) \rangle 0\}|))$. This metric evaluates the symmetry of the layout through calculating the difference between the number of nodes in the left and right sides of the backbone.
- QM_{ec}: The number of edge crossings $QM_{ec} = \left|\left\{\left(e_i, e_j\right) \mid e_i, e_j \in E \wedge ec\left(e_i, e_j\right) = 1\right\}\right|$. This metric measures the number of crossings between edges.
- QM_{el}: Total edge length $QM_{el} = \sum_{e=(u,v)\in E} sqrt\left((\pi_x(u) - \pi_x(v))^2 + \left(\pi_y(u) - \pi_y(v)\right)^2\right)$.
- QM_{eo}: Edge orthogonality $QM_{eo} = 1 - \frac{1}{|N|} \sum_{e\in E} min(\theta, \widehat{90}^{\circ} - \theta, \widehat{180}^{\circ} - \theta)/45^{\circ}$, where $\theta = \deg\left(\overrightarrow{e}, \overrightarrow{(1,0)}\right)$. This metric gauge the adherence of edges to an imaginary Cartesian grid.
- QM_{no}: Node orthogonality $QM_{no} = |\{n \in N \mid \pi_{real}(n) = 1\}|/(\max(\{\pi_r(n) \mid n \in N\}) \times \max(\{\pi_o(n) \mid n \in N\}))$. This criterion evaluates whether the rank and order spaces are utilized efficiently.

6.1.3. Implementation

The proposed method is implemented in Python using Gurobi optimizer (v13.0.1, academic license) for solving the integer programming. As no public implementation of the benchmark is available, we reimplemented the method of Mennes et al. [23] based on the descriptions in their paper. The source code is available at https://github.com/deoksanglee/BackboneLayoutIP.

6.1.4. Results

An overview of the evaluation result is presented in Table 6, with detailed results for each dataset and filter ratio shown in Appendix A.

The evaluation demonstrates that our algorithm outperforms the benchmark across multiple criteria, including the number of back edges, balance degree, edge crossings, edge length, and node orthogonality. Specifically, our algorithm produced fewer back edges in 21 out of 25 experiments. This is attributed to the use of integer programming, which considers backbone structure and minimizes precedence violations during the rank assignment. In contrast, the benchmark relies on heuristic rank assignments, which frequently introduce unnecessary back edges. In terms of balance degree, our method showed superior results in 23 out of 25 cases. The benchmark's greedy left-right node placement often results in asymmetrical layouts, whereas our integer programming approach ensures a more even distribution of nodes on either side of the backbone. For edge crossings, edge length, and node orthogonality, our method consistently outperformed the benchmark. In the benchmark method, heuristic rank assignments often increase the number of vertical layers (ranks), which elongates edge paths and increases the chance of edge crossings. Our method, however, places unordered node pairs within the same rank where possible, reducing the overall number of ranks used. This results in more compact vertical layouts, shorter edge lengths, and fewer crossings. The reduced rank usage also contributes to improved spatial efficiency, enhancing node orthogonality. Regarding edge orthogonality, the benchmark exhibited better performance in 19 out of 25 experiments, whereas our method surpassed it in 6. This marginal difference can be explained by the benchmark's tendency to produce a larger number of vertically aligned (straight) edges due to its use of more ranks. These vertically elongated edges naturally align more closely with the Cartesian grid, which can marginally enhance edge orthogonality.

Fig. 20 shows examples of layout results. In these figures, the blue lines represent the downward edges and the red lines indicate the upward edges. Fig. 20(a) and Fig. 20(b) display the layouts generated by our and prior methods using a subset of hospital log. Fig. 20(c) and Fig. 20(d) show layouts generated from the BPI 2019 event log.

6.2. Qualitative evaluation

We conduct a qualitative evaluation with 27 participants using layouts generated from the BPI 2019 and hospital logs. Participants compared our layout with the benchmark in terms of task helpfulness, readability, and aesthetic preference. Overall, our method was preferred across both logs and all three criteria.

6.2.1. Evaluation method

Prior research in process visualization evaluated specific layout and visual design elements by examining users' preferences after interacting with them [45,46]. Building on these works, we adopt a preference-based evaluation approach in which participants first understand the processes and subsequently indicate their preferences.

Table 6
Overview of quantitative evaluation results.

Method	Metric					
	QM_{be}	QM_{bal}	QM_{ec}	QM_{el}	QM_{eo}	QM_{no}
Our	21	23	23	24	6	24
Benchmark [23]	2	1	1	1	19	0
Tie	2	1	1	0	0	1

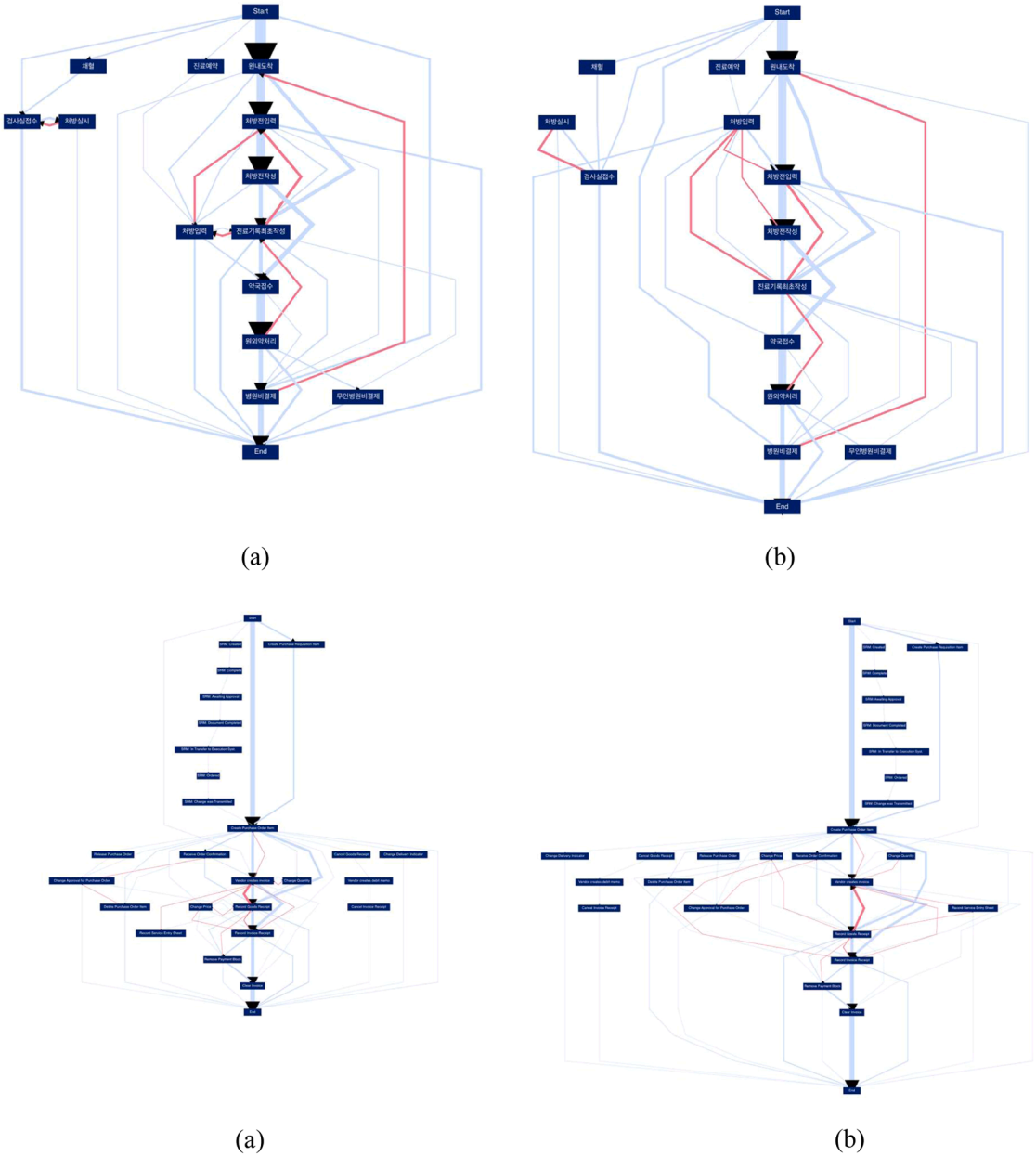


Fig. 20. (a) Our layout for the hospital log with four back edges and a balance degree of 0.99. The layout yields 26 edge crossings, a total edge length of 213.2, an edge orthogonality of 0.62, and a node orthogonality of 0.63. (b) Benchmark layout for the hospital log with seven back edges and a balance degree of 0.99. The layout yields 29 edge crossings, the total edge length of 235.2, the edge orthogonality of 0.67, and the node orthogonality of 0.58. (c) Our layout for the BPI 2019 log has ten back edges and a balance degree of 1.00. The layout yields 46 edge crossings, the total edge length of 269.4, the edge orthogonality of 0.71, and the node orthogonality of 0.50. (d) Benchmark layout for the BPI 2019 log with eleven back edges and a balance degree of 0.97. The layout yields 55 edge crossings, the total edge length of 420.4, the edge orthogonality of 0.72, and the node orthogonality of 0.39.

The evaluation was conducted using layouts generated from the hospital and BPI 2019 logs. The hospital log was selected because its relatively small size allows participants to clearly follow the overall process flow, making it suitable for assessing layout readability and structure. In contrast, the BPI 2019 log represents a more complex and realistic business process with multiple cycles and alternative paths, while remaining interpretable for qualitative analysis.

A total of 27 participants took part in the study, including graduate students and industry practitioners with experience in business process analysis and process mining visualization tools. For each event log, participants were presented with two layouts—one generated by our method and one by the benchmark—and were asked to perform the tasks listed in Table 7 using both layouts. After

completing the tasks, participants indicated their preferred layout.

Layout preferences were assessed according to three criteria (Table 8): (i) which layout better supports task-based process understanding, (ii) which layout is easier to read and comprehend overall, and (iii) which layout is perceived as more aesthetically pleasing. Participants also provided brief explanations for their choices to support qualitative interpretation of the results.

6.2.2. Results

Overall, the qualitative evaluation shows a clear preference for our layout across all three aspects—task helpfulness (Q1), general readability (Q3), and aesthetic preference (Q5)—for both event logs as shown in Fig. 21. Participants most frequently attributed this preference to improved cycle and back-edge interpretability, compact placement of related activities, and a balanced structure centered around the main flow. At the same time, the responses reveal an important tension between compactness and perceived visual clarity, with a minority of participants consistently preferring the benchmark layout.

For the BPI 2019 log, our layout received 77.8% preference for task helpfulness (Q1), compared to 14.8% for the benchmark (with 7.4% ties). Participants frequently reported that our layout makes cycle-related structures and back-edge relations easier to identify, often because cyclic node pairs appear in close proximity and are visually salient. In the open responses, the most common reasons for choosing our layout in Q1 were cycle/back-edge visibility and easier tracing of precedence relations, such as identifying cyclic pairs without extensive visual scanning. For general readability (Q3), our layout was preferred by 74.1% of participants (benchmark 25.9%). The dominant reason was compactness, which reduced navigation costs (e.g., less panning and zooming) and made activity labels appear larger and easier to read. For aesthetic preference (Q5), our layout received 63.0% preference (benchmark 22.2%, 14.8%), where participants most often cited symmetry and balance, as well as a more compact overall footprint. Importantly, participants who preferred the benchmark layout for the BPI 2019 log consistently described it as “less cluttered” or “more spacious.” These participants noted that the benchmark’s longer edges, increased number of ranks, and larger layout area introduce more whitespace, which reduced the feeling of visual congestion and made the layout appear clearer at first glance. In contrast, several participants remarked that although our layout supports efficient task execution, its high information density can make the layout feel more complex, highlighting a trade-off between compactness and perceived visual simplicity.

For the hospital log, preference for our layout was even stronger. Our layout achieved 74.1% preference for task helpfulness (Q1) (benchmark 18.5%, ties 7.4%). For general readability (Q3) and aesthetic preference (Q5), our layout was preferred by 92.6% of participants in both cases (benchmark 3.7% and 7.4%, respectively). In the open responses, participants repeatedly emphasized that our layout improves interpretability by presenting cycle-related pairs in visually structured and recognizable patterns, often described as distinct cyclic shapes. Additional reasons included reduced edge bends, shorter edge lengths, and a more stable and balanced node distribution around the main flow.

When comparing preference trends across event logs, the overall preference for our method (averaged across the three aspects) was higher for the hospital log (86.4%) than for BPI 2019 (71.6%)This difference can be explained by the hospital model’s focus on clinically meaningful feedback loops and local precedence relations, where a compact layout and clearly visible cycles offer immediate practical advantages.In contrast, the denser structure of the BPI 2019 model amplifies the trade-off between information density and whitespace, making the benchmark’s more spacious layout appealing to a subset of users.

Taken together, these results indicate that user preference is not solely determined by geometric simplicity, but also by how visual density and whitespace interact with task demands. While our layout supports analytical tasks by emphasizing structural relations and reducing navigation effort, the benchmark layout can appear clearer for users who prioritize low perceived visual complexity. This finding underscores the importance of considering task-dependent and perception-driven factors when evaluating process visualization layouts.

6.3. Running time

We analyze the running time of our approach by measuring the elapsed time for the three main steps, rank assignment, component balancing, and node positioning. The cross-minimization step is excluded because both methods use the same technique as in prior work. For rank assignment and component balancing, we measure the elapsed time until the optimal solution is found. All experiments are conducted on a machine with 16GB RAM and a 12th Gen Intel Core i3–12100F CPU at 3.30 GHz. The results are reported in Table 9 and results for each filter value are reported in Appendix B. Overall, our method requires less computation time than the benchmark in all three steps. For both methods, the running time increases as the number of variants and edges grows.

In the rank assignment step, our method is faster despite relying on integer programming. This can be attributed to the efficiency of modern optimization solvers, which exploit advanced presolve, pruning, and parallel processing techniques, whereas the benchmark processes variants sequentially and incrementally updates node ranks based on previously processed variants. This sequential

Table 7
Tasks for participants.

Index	Task
T1	Find the most frequent sequence of activities from start node to end node?
T2	Find the activities (nodes) performed directly before the activity ‘Record Service Entry Sheet’?
T3	Find the activities (nodes) performed directly after the activity ‘Change Quantity’?
T4	Find the pair of nodes (A, B) forms a cycle, where there are edges from A to B and from B to A?

Table 8
Questions for assessing layout preferences.

Index	Question	Answer
Q1	Which layout is more helpful for answering the questions?	A / B / No preference
Q2	Why?	Open
Q3	Which layout is easier to read and understand regardless of questions?	A / B / No preference
Q4	Why?	Open
Q5	Which layout do you prefer considering the aesthetic elements of layout?	A / B / No preference
Q6	Why?	Open

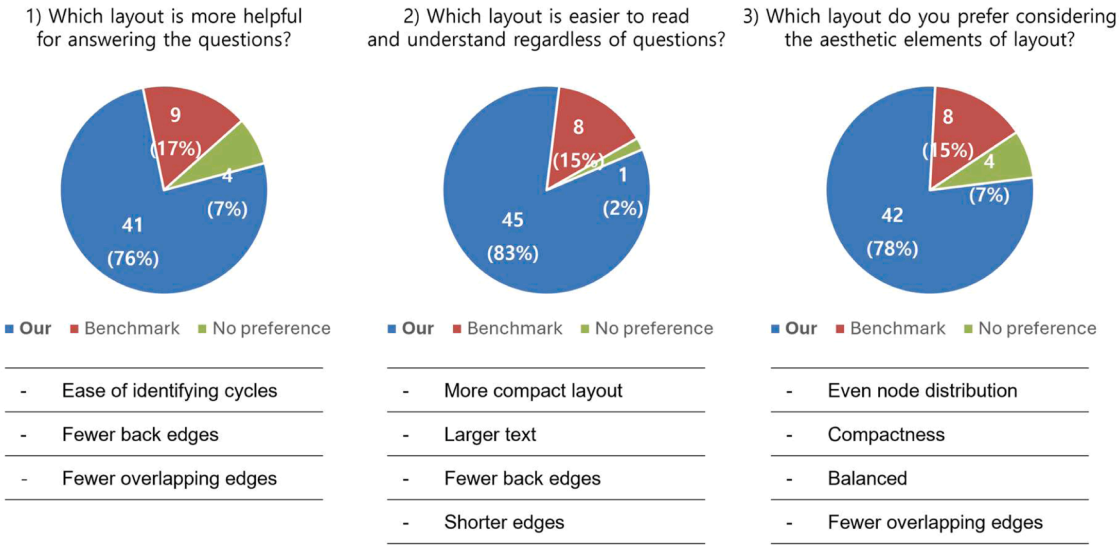


Fig. 21. Results of the qualitative evaluation. Participants compared our layout to the benchmark across three aspects: task helpfulness, general readability, and aesthetic preference. Our layout was preferred nearly five times more, with users noting better cycle identification, fewer edge overlaps, and improved balance and clarity.

Table 9
Elapsed time for each step for the five event logs.

Dataset	Method	Elapsed time (seconds)		
		Rank assignment	Component balancing	Node positioning
BPI2017	Our	0.38	0.24	0.50
	Benchmark	2.32	0.08	3.38
BPI2018	Our	9.09	0.05	9.56
	Benchmark	272.82	2.62	131.46
BPI2019	Our	128.50	0.06	1.08
	Benchmark	131.13	2.30	77.64
Hospital log	Our	0.46	0.06	0.53
	Benchmark	4.62	0.09	3.82
Web log	Our	0.95	0.05	1.55
	Benchmark	1.72	0.03	2.76

dependency limits opportunities for parallelization and leads to longer running times.

A similar trend is observed in the component balancing step. Our approach solves the balancing problem using an optimization-based formulation, while the benchmark balances components by sequentially moving them between the left and right sides, resulting in higher execution time.

In the node positioning step, our method also outperforms the benchmark. This is mainly due to the more compact layouts produced by our rank assignment, which use fewer ranks and therefore introduce fewer virtual nodes. In contrast, the benchmark uses more ranks, resulting in a larger number of virtual nodes that must be positioned and an increased computational cost.

7. Discussion

This section discusses the sensitivity of the proposed backbone-based layouts, their limitations, and directions for future research.

7.1. Layout sensitivity to backbone selection

To assess the sensitivity of the proposed layout to backbone selection, we examined how layout quality metrics vary across different backbone variants. Specifically, we used the BPI 2017 and hospital logs and generated layouts using the top three most frequent variants as candidate backbones. In practice, backbones are often chosen based on variant frequency. Therefore, focusing analysis on the top three variants provides a realistic evaluation of robustness among dominant process behaviors. The quantitative metrics are shown in Table 10. Although minor variations are observed, selecting the backbone from within the top three most frequent variants does not substantially affect overall layout quality. This indicates that the proposed layout method is relatively robust to backbone selection, provided that the backbone represents a dominant process variant.

7.2. Limitations

While this study demonstrates that the proposed backbone-based layout improves flow consistency, cycle interpretability, compactness, and overall user preference in qualitative evaluations, several limitations remain.

From a methodological perspective, the proposed approach has the following limitations. First, while allowing horizontal edges improves space efficiency and facilitates the identification of cycle-related activities, it may partially disrupt the vertical sequencing of activities. In our approach, nodes involved in cycles are intentionally placed on the same rank, which allows horizontal edges to highlight cyclic structures more clearly. Although this design choice is effective for identifying cycles, it may interfere with users' ability to follow the overall process flow in a strictly top-down manner. Second, the proposed layout generation relies on a fixed set of optimization criteria, such as minimizing back edges, preserving backbone linearity, and achieving compact placement. However, user preferences for process visualizations are subjective and task dependent. The proposed method does not adaptively adjust its optimization objectives based on individual preferences or task contexts, which limits its flexibility. Supporting customizable or adaptive weighting of layout criteria remains an important direction for future work.

In terms of evaluation, several additional limitations should be noted. First, the evaluation was limited to a comparison with the benchmark algorithm. As a result, the proposed approach was not evaluated against widely used commercial process mining tools, such as Celonis and Apromore. Comparing the proposed method with such tools is a clear next step to better assess its practical effectiveness. Second, the evaluation did not include comparisons with general-purpose graph layout algorithms. Process layout is as a specific instance of the graph layout, and graph layout methods can offer useful perspectives on process model comprehension. However, this study focused on backbone-based process layouts, and a systematic comparison with graph layout algorithms remains an open direction for future evaluation. Third, the qualitative evaluation was limited to relatively basic tasks, such as precedence identification and cycle detection. More advanced tasks that require a deeper understanding of process models, such as identifying bottlenecks and comparing two variants, were not included. In addition, aesthetic perception was assessed without a rigorous or formally defined operationalization, leaving its interpretation dependent on participants' subjective judgment. Fourth, the impact of objective functions and constraints in the rank assignment and component balancing steps was not analyzed. In particular, the node-separation constraints impose a strong restriction by forcing nodes connected by a non-mutual relation onto different ranks. While this prevents degenerate layouts, it may also increase edge lengths or limit compactness. Finally, we did not consider interactive filtering scenarios in which a complete layout is first generated and then filtered without re-computation. Such interaction patterns are common in process analysis tools, and the stability of node positions under filtering remains unexplored.

7.3. Future research

The limitations above suggest several promising research directions. First, the role of horizontal edges in process understanding warrants deeper investigation. In our approach, nodes involved in cycles are intentionally placed at the same rank, allowing horizontal edges to highlight cyclic structures more clearly. This design choice was effective for cycle identification tasks; however, horizontal edges may partially disrupt the perceived top-down sequencing of activities, potentially hindering users' understanding of global

Table 10
Layout metrics under different backbone selections.

Dataset	Backbone	Metric					
		QM_{be}	QM_{bal}	QM_{ec}	QM_{el}	QM_{eo}	QM_{no}
BPI2017	Top1 variant	61	0.73	1308	3286.36	0.72	0.03
	Top2 variant	62	0.71	1373	2943.29	0.75	0.04
	Top3 variant	61	0.73	1380	3012.93	0.75	0.03
Hospital log	Top1 variant	87	0.74	2993	4260.53	0.73	0.04
	Top2 variant	87	0.74	2869	4086.96	0.75	0.04
	Top3 variant	91	0.70	2948	4912.19	0.75	0.03

process flow. Such investigations could further inform us when horizontal edges should be encouraged or restricted, depending on the analytical goal.

Second, adaptive and preference-aware layout generation remains an open challenge. Qualitative feedback revealed that users differ in how they perceive visual complexity: some associate compactness with clarity and efficiency, while others prefer layouts with more whitespace that feel less cluttered. Future research may explore adaptive layout strategies that dynamically adjust optimization objectives based on user preferences, task contexts, or analysis goals.

Third, the stability of layout under filtering deserves focused attention. In this study, the proposed method was evaluated by re-running the layout algorithm on filtered event logs, assessing performance across varying model sizes and structures. However, we did not examine scenarios in which a complete layout is generated once and then interactively filtered without recomputing the layout. Future work should define quantitative metrics of stability and evaluate how much layouts change under filtering.

Beyond these directions, AI-based process layout generation represents a promising but underexplored research avenue. Although AI-based graph layout methods have gained increasing attention, they have primarily focused on general graph drawing rather than process-specific structures. As illustrated in Fig. 22, most existing process layout methods are positioned within non-AI paradigms, leaving a large research space at the intersection of process layout and AI-based methods. A key challenge for AI-based process layout is the lack of high-quality training data tailored to process models. In this context, the optimization-based backbone layouts proposed in this paper naturally occupy an intermediate position between classical heuristic methods and future AI-based approaches. By producing consistent and principled layout examples, it may serve as supervision or refinement targets for future AI models.

8. Conclusion

This study introduced an improved backbone-based process layout generation method, aimed at addressing key limitations in existing approaches, such as unnecessary back edges, restricted use of horizontal edges, non-linear backbone alignment, and imbalanced node distribution. By integrating integer programming with heuristic techniques, the proposed framework enhances rank assignment, virtual node generation, component balancing, and node positioning.

The method was evaluated on five real-world event logs with varying filter ratios. Results demonstrated improvements across core metrics, including the number of back edges, edge crossings, edge lengths, node orthogonality, and layout symmetry, when compared with a benchmark method. These improvements were consistent across different levels of process complexity and filtering conditions.

A qualitative user study further validated these findings. Participants expressed a preference for the proposed layout, citing its clearer structure, more balanced visual composition, and greater support for interpreting process flows. Users also found it easier to trace paths, detect cycles, and read activity labels, thanks to the layout's organized presentation.

Declaration of generative AI and AI-assisted technologies in the writing process

During the preparation of this work, the author(s) used ChatGPT-5.2 in order to improve readability and language. After using this tool/service, the author(s) reviewed and edited the content as needed and take(s) full responsibility for the content of the publication.

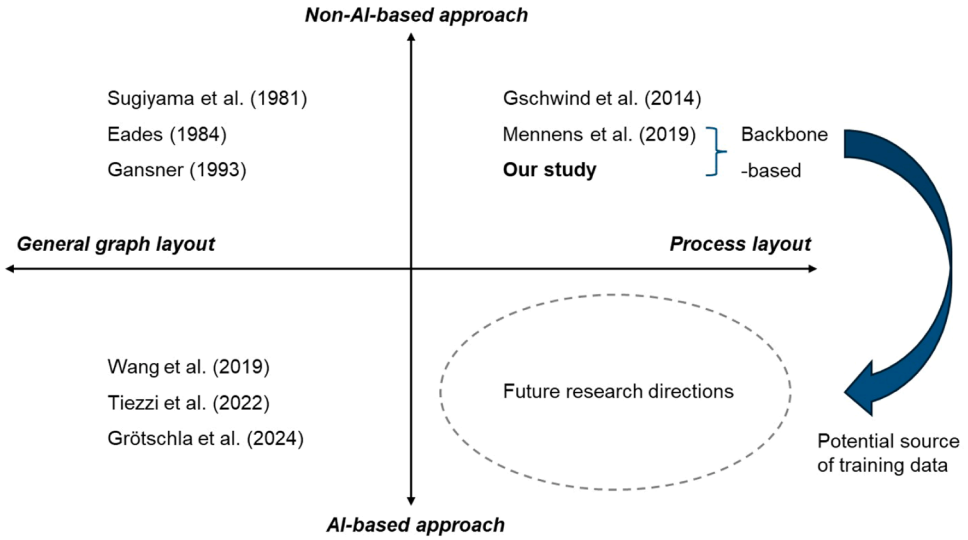


Fig. 22. Research landscape and future directions for process layout.

CRediT authorship contribution statement

Deoksang Lee: Writing – original draft, Methodology, Data curation, Conceptualization. **Minseok Song:** Writing – review & editing, Supervision, Methodology, Funding acquisition, Conceptualization. **Wil M.P. van der Aalst:** Writing – review & editing, Methodology, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. RS-2024-00357330).

Appendix A. Quantitative evaluation results for each event log and filter ratio

Dataset	Method	Filter	Metric					
			QM_{be}	QM_{bal}	QM_{ec}	QM_{el}	QM_{eo}	QM_{no}
BPI2017	Our	80	44	1.00	562	1594.06	0.74	0.04
		60	55	0.82	923	2573.22	0.75	0.03
		40	61	0.69	1103	2760.47	0.74	0.03
		20	61	0.73	1384	3050.97	0.73	0.03
		0	61	0.73	1308	3286.36	0.72	0.03
	Benchmark	80	49	0.49	943	3192.72	0.81	0.02
		60	60	0.41	2310	5480.88	0.87	0.01
		40	66	0.42	1890	5333.44	0.80	0.01
		20	71	0.40	2161	6022.05	0.81	0.01
		0	72	0.38	2323	6524.29	0.73	0.01
BPI2018	Our	80	159	0.80	5819	11,628.20	0.74	0.02
		60	201	0.69	11,023	20,286.09	0.76	0.02
		40	234	0.64	14,573	27,113.57	0.75	0.02
		20	237	0.63	16,967	31,810.08	0.77	0.02
		0	241	0.62	18,690	31,750.78	0.79	0.02
	Benchmark	80	190	0.56	16,478	36,813.16	0.88	0.01
		60	232	0.47	29,793	66,884.94	0.82	0.01
		40	269	0.41	54,741	99,549.58	0.99	0.00
		20	287	0.39	54,473	118,266.67	0.85	0.00
		0	294	0.39	46,273	113,496.88	0.85	0.00
BPI2019	Our	80	83	0.24	5902	7767.12	0.85	0.07
		60	116	0.19	9459	10,985.58	0.83	0.05
		40	146	0.14	14,374	16,488.35	0.88	0.05
		20	150	0.13	18,120	16,082.22	0.85	0.05
		0	149	0.14	18,068	15,282.10	0.87	0.05
	Benchmark	80	132	0.13	14,058	34,798.48	0.75	0.01
		60	175	0.10	34,698	69,557.14	0.85	0.01
		40	207	0.06	45,488	100,984.14	0.82	0.01
		20	219	0.06	51,765	114,540.99	0.86	0.00
		0	221	0.06	51,657	114,770.39	0.85	0.00
Hospital log	Our	80	28	1.00	409	1266.82	0.62	0.05
		60	52	0.69	924	2504.10	0.68	0.04
		40	83	0.70	2188	4096.13	0.72	0.04
		20	87	0.72	2849	4200.52	0.75	0.04
		0	87	0.74	2993	4260.53	0.73	0.04
	Benchmark	80	31	0.87	404	1693.42	0.70	0.03
		60	52	0.5	1071	3452.08	0.75	0.02
		40	93	0.37	3335	9212.08	0.80	0.01
		20	106	0.36	4545	13,684.74	0.80	0.01
		0	109	0.38	4262	12,227.39	0.76	0.01
Web log	Our	80	7	0.99	12	134.74	0.85	0.24
		60	26	1.00	106	516.03	0.67	0.11
		40	38	1.00	355	1226.28	0.68	0.07
		20	53	0.79	772	2317.22	0.70	0.05
		0	62	0.82	1042	3128.36	0.70	0.04
	Benchmark	80	7	0.99	12	125.26	0.78	0.24

(*continued on next page*)

(*continued*)

Dataset	Method	Filter	Metric					
			QM_{be}	QM_{bal}	QM_{ec}	QM_{el}	QM_{eo}	QM_{no}
		60	20	1.97	160	672.05	0.69	0.07
		40	37	0.95	473	1583.28	0.62	0.05
		20	55	0.67	1295	3396.90	0.66	0.03
		0	64	0.70	1910	4782.77	0.73	0.03

Appendix B. Elapsed time for each event log and filter ratio

Dataset	Method	Filter	Elapsed time (seconds)		
			Rank assignment	Component balancing	Node positioning
BPI2017	Our	80	0.20	0.06	0.42
		60	0.17	0.06	0.42
		40	0.31	0.05	0.6
		20	0.20	0.05	0.47
		0	0.38	0.24	0.50
	Benchmark	80	0.48	0.05	1.08
		60	1.23	0.04	2.05
		40	1.74	0.05	3.08
		20	2.45	0.06	3.42
		0	2.32	0.08	3.38
BPI2018	Our	80	1.97	0.06	2.68
		60	1.74	0.06	5.67
		40	5.09	0.07	7.26
		20	15.58	0.05	7.44
		0	9.09	0.05	9.56
	Benchmark	80	50.40	0.60	41.83
		60	90.75	0.94	51.36
		40	210.03	1.81	93.19
		20	234.98	2.05	119.26
		0	272.82	2.62	131.46
BPI2019	Our	80	22.53	0.05	0.36
		60	18.00	0.05	0.80
		40	124.45	0.05	0.98
		20	196.82	0.05	1.40
		0	128.50	0.06	1.08
	Benchmark	80	18.72	0.45	8.83
		60	51.69	0.94	30.84
		40	93.56	1.42	51.56
		20	122.47	2.18	79.49
		0	131.13	2.30	77.64
Hospital log	Our	80	0.12	0.04	0.26
		60	0.46	0.04	0.37
		40	0.45	0.05	0.41
		20	0.39	0.05	0.41
		0	0.46	0.06	0.53
	Benchmark	80	0.15	0.01	0.40
		60	0.59	0.01	0.75
		40	3.72	0.07	2.33
		20	4.85	0.10	4.48
		0	4.62	0.09	3.82
Web log	Our	80	0.02	0.05	0.07
		60	0.89	0.06	0.20
		40	0.57	0.06	0.47
		20	0.80	0.06	0.99
		0	0.95	0.05	1.55
	Benchmark	80	0.01	0.01	0.04
		60	0.07	0.01	0.44
		40	0.35	0.01	0.84
		20	1.09	0.02	1.39
		0	1.72	0.03	2.76

Data availability

Data will be made available on request.

References

[1] T. Ahmad, A. Van Looy, Business process management and digital innovations: a systematic literature review, Sustainability 12 (17) (2020) 6827, https://doi.org/10.3390/SU12176827.
[2] A. De Ramón Fernández, D. Ruiz Fernández, Y. Sabuco García, Business process management for optimizing clinical processes: a systematic literature review, Health Inform. J. 26 (2) (2020) 1306–1320, https://doi.org/10.1177/1460458219877092.
[3] D. Koraca, How ICT affects business processes, in: Proceedings of the International Conference on Human Systems Engineering and Design. Future Trends and Applications, 2021, https://doi.org/10.54941/ahfe1001123.
[4] N. Melão, E-business processes and e-business process modelling: a state-of-the-art overview, Int. J. Serv. Technol. Manag. 11 (3) (2009) 293–322, https://doi.org/10.1504/IJSTM.2009.024094.
[5] W. van der Aalst, Process mining: overview and opportunities, ACM Trans. Manag. Inf. Syst. 3 (2) (2012) 1–17, https://doi.org/10.1145/2229156.2229157.
[6] G. Lugaresi, A. Matta, Automated digital twin generation of manufacturing systems with complex material flows: graph model completion, Comput. Ind. 151 (2023) 103977, https://doi.org/10.1016/j.compind.2023.103977.
[7] G. Park, D. Schuster, W. van der Aalst, Pattern-based action engine: generating process management actions using temporal patterns of process-centric problems, Comput. Ind. 153 (2023) (2023) 104020, https://doi.org/10.1016/j.compind.2023.104020.
[8] W. van der Aalst, A. Adriansyah, A.K.A. De Medeiros, F. Arcieri, T. Baier, T. Blickle, J.C. Bose, P. Van Den Brand, R. Brandtjen, J. Buijs, A. Burattin, J. Carmona, M. Castellanos, J. Claes, J. Cook, N. Costantini, F. Curbera, E. Damiani, M. De Leoni, P. Delias, B.F. Van Dongen, M. Dumas, S. Dustdar, D. Fahland, D.R. Ferreira, W. Gaaloul, F. Van Geffen, S. Goel, C. Günther, A. Guzzo, P. Harmon, A. Ter Hofstede, J. Hoogland, J.E. Ingvaldsen, K. Kato, R. Kuhn, A. Kumar, M. La Rosa, F. Maggi, D. Malerba, R.S. Mans, A. Manuel, M. McCreesh, P. Mello, J. Mendling, M. Montali, H.R. Motahari-Nezhad, M. Zur Muehlen, J. Munoz-Gama, L. Pontieri, J. Ribeiro, A. Rozinat, H. Seguel Pérez, R. Seguel Pérez, M. Sepúlveda, J. Sinur, P. Soffer, M. Song, A. Sperduti, G. Stilo, C. Stoel, K. Swenson, M. Talamo, W. Tan, C. Turner, J. Vanthienen, G. Varvaressos, E. Verbeek, M. Verdonk, R. Vigo, J. Wang, B. Weber, M. Weidlich, T. Weijters, L. Wen, M. Westergaard, M. Wynn, Process mining manifesto, in: Proceedings of the International Workshop on Business Process Management, 2012, pp. 169–194, https://doi.org/10.1007/978-3-642-28108-2_19.
[9] M. Song, W. van der Aalst, Supporting process mining by showing events at a glance, in: Proceedings of the International Workshop on Information Technologies and Systems, 2007, pp. 139–145.
[10] M. Cho, K. Kim, J. Lim, H. Baek, S. Kim, H. Hwang, M. Song, S. Yoo, Developing data-driven clinical pathways using electronic health records: the cases of total laparoscopic hysterectomy and rotator cuff tears, Int. J. Med. Inform. 133 (1) (2020) 104015, https://doi.org/10.1016/j.ijmedinf.2019.104015.
[11] M. Cho, G. Park, M. Song, J. Lee, B. Lee, E. Kum, Discovery of resource-oriented transition systems for yield enhancement in semiconductor manufacturing, IEEE Trans. Semicond. Manuf. 34 (1) (2021) 17–24, https://doi.org/10.1109/TSM.2020.3045686.
[12] D. Dakic, D. Stefanovic, I. Cosic, T. Lolic, M. Medojevic, Business process mining application: a literature review, in: Proceedings of the International Conference on Annals of DAAAM, 2018, https://doi.org/10.2507/29th.daaam.proceedings.125.
[13] K. Park, M. Cho, M. Song, S. Yoo, H. Baek, S. Kim, K. Kim, Exploring the potential of OMOP common data model for process mining in healthcare, PLoS ONE 18 (1) (2023) e0279641, https://doi.org/10.1371/journal.pone.0279641.
[14] J. Becker, M. Rosemann, C. von Uthmann, Guidelines of business process modeling, in: W. van der Aalst, J. Desel, A. Oberweis (Eds.), Business Process Management. LNCS, Springer, Berlin, Heidelberg, 2000, pp. 30–49, https://doi.org/10.1007/3-540-45594-9_3, 1806.
[15] S.H. Kim, Understanding the role of visualizations on decision making: a study on working memory, Informatics 7 (4) (2020) 53, https://doi.org/10.3390/informatics7040053.
[16] J. Moore, Data visualization in support of executive decision making, Interdiscip. J. Inf. Knowl. Manag. 12 (2017) 125, https://doi.org/10.28945/3687.
[17] S. Park, B. Bekemeier, A. Flaxman, M. Schultz, Impact of data visualization on decision-making and its implications for public health practice: a systematic literature review, Inform. Health Soc. Care 47 (2) (2022) 175–193, https://doi.org/10.1080/17538157.2021.1982949.
[18] K. Figl, Comprehension of procedural visual business process models: a literature review, Bus. Inf. Syst. Eng. 59 (1) (2017) 41–67.
[19] H.A. Reijers, J. Mendling, A study into the factors that influence the understandability of business process models, IEEE Trans. Syst. Man Cybern. 41 (3) (2011) 449–462, https://doi.org/10.1109/TSMCA.2010.2087017.
[20] V. Stein Dani, C.M. Dal Sasso Freitas, L.H. Thom, Ten years of visualization of business process models: a systematic literature review, Comput. Stand. Interfaces 66 (2019) 103347, https://doi.org/10.1016/j.csi.2019.04.006.
[21] T. Gschwind, J. Pinggera, S. Zugal, H.A. Reijers, B. Weber, A linear time layout algorithm for business process models, J. Vis. Lang. Comput. 25 (2) (2014) 117–132, https://doi.org/10.1016/j.jvlc.2013.11.002.
[22] K. Misue, P. Eades, W. Lai, K. Sugiyama, Layout adjustment and the mental map, J. Vis. Lang. Comput. 6 (2) (1995) 183–210, https://doi.org/10.1006/jvlc.1995.1010.
[23] R.J.P. Mennens, R. Scheepens, M.A. Westenberg, A stable graph layout algorithm for processes, Comput. Graph. Forum 38 (3) (2019) 725–737, https://doi.org/10.1111/cgf.13723.
[24] UIPath, 2026. Evolve your processes, discover better outcomes. https://www.uipath.com/product/process-mining.
[25] K. Sugiyama, S. Tagawa, M. Toda, Methods for visual understanding of hierarchical system structures, IEEE Trans. Syst. Man Cybern. 11 (2) (1981) 109–125, https://doi.org/10.1109/TSMC.1981.4308636.
[26] P. Eades, A heuristic for graph drawing, Congr. Numer. 42 (11) (1984) 149–160.
[27] E.R. Gansner, E. Koutsofios, S.C. North, K.P. Vo, A technique for drawing directed graphs, IEEE Trans. Softw. Eng. 19 (3) (1993) 214–230, https://doi.org/10.1109/32.221135.
[28] X. Xu, Y. Zhang, An integrated vector error correction and directed acyclic graph method for investigating contemporaneous causalities, Decis. Anal. J.l 7 (2023) 100229, https://doi.org/10.1016/j.dajour.2023.100229.
[29] B. Jin, X. Xu, A study of contemporaneous residential real estate price causation across major Jiangsu province cities: methodology using vector error-correction models and directed acyclic graphs, Econ. Open (2025) 2550008, https://doi.org/10.1142/S308284142550008X.
[30] C. Zhao, X. Li, Y. Cai, A power grid topology detection method based on edge graph attentional neural network, Electr. Power Syst. Res. 239 (2025) 111219, https://doi.org/10.1016/j.epsr.2024.111219.
[31] Y. Wang, Z. Jin, Q. Wang, W. Cui, T. Ma, H. Qu, Deepdrawing: a deep learning approach to graph drawing, IEEE Trans. Vis. Comput. Graph. 26 (1) (2019) 676–686, https://doi.org/10.1109/TVCG.2019.2934798.
[32] O.H. Kwon, K.L. Ma, A deep generative model for graph layout, IEEE Trans. Vis. Comput. Graph. 26 (1) (2019) 665–675, https://doi.org/10.1109/TVCG.2019.2934396.
[33] M. Tiezzi, G. Ciravegna, M. Gori, Graph neural networks for graph drawing, IEEE Trans. Neural Netw. Learn. Syst. 35 (4) (2022) 4668–4681, https://doi.org/10.1109/TNNLS.2022.3184967.
[34] Celonis, 2026. Find and capture the value hiding in your processes. https://www.celonis.com.
[35] H.C. Purchase, Which aesthetic has the greatest effect on human understanding?, in: Proceedings of the International Symposium on Graph Drawing, 1997, pp. 248–261, https://doi.org/10.1007/3-540-63938-1_67.
[36] H.C. Purchase, Metrics for graph drawing aesthetics, J. Vis. Lang. Comput. 13 (5) (2002) 501–516, https://doi.org/10.1016/S1045-926X(02)90232-6.

[37] M. Schrepfer, J. Wolf, J. Mendling, H.A. Reijers, The impact of secondary notation on process model understanding, in: Proceedings of the International Conference on the Practice of Enterprise Modeling, 2009, pp. 161–175, https://doi.org/10.1007/978-3-642-05352-8_13.
[38] P. Effinger, N. Jogsch, S. Seiz, On a study of layout aesthetics for business process models using BPMN, in: Proceedings of the International Conference on Business Process Modeling Notation, 2010, pp. 31–45, https://doi.org/10.1007/978-3-642-16298-5_5.
[39] V. Bernstein, P. Soffer, Identifying and quantifying visual layout features of business process models, in: Proceedings of the International Conference on Enterprise, Business-Process and Information System Modeling, 2015, pp. 200–213, https://doi.org/10.1007/978-3-319-19237-6_13.
[40] A. Burattin, V. Bernstein, M. Neurauter, P. Soffer, B. Weber, Detection and quantification of flow consistency in business process models, Softw. Syst. Model. 17 (2018) 633–654, https://doi.org/10.1007/s10270-017-0576-y.
[41] W. van der Aalst, in: J. Carmona (Ed.), Process Mining Handbook, Springer, Cham, 2022, https://doi.org/10.1007/978-3-031-08848-3.
[42] BPI Challenge 2017, 4TU. ResearchData. https://data.4tu.nl/articles/dataset/BPI_Challenge_2017/12696884 (accessed 18 February 2026).
[43] BPI Challenge 2018, 4TU. ResearchData. https://data.4tu.nl/articles/dataset/BPI_Challenge_2018/12688355/1 (accessed 18 February 2026).
[44] BPI Challenge 2019, 4TU. ResearchData. https://data.4tu.nl/articles/dataset/BPI_Challenge_2019/12715853/1 (accessed 18 February 2026).
[45] R. Brinkman, F. Mannhardt, R.J. Mennens, R.J. Scheepens, Interpretability challenges for discovered process models: a user study and prototype solution, in: Proceedings of the EuroVis Workshop on Visual Process Analytics, 2024, https://doi.org/10.2312/vipra.20241103.
[46] K. Figl, J. Recker, Exploring cognitive style and task-specific preferences for process representations, Requir. Eng. 21 (1) (2016) 63–85.

Deoksang Lee received a Ph.D in industrial & management engineering from Pohang University of Science and Technology (POSTECH) in 2026. He is currently a researcher at Samsung Electronics. His research interests include deep learning, process mining, data science, and data analytics in manufacturing. He has participated in several research projects funded by the National Research Foundation of Korea, Samsung Electronics, POSCO, etc.

Minseok Song received a Ph.D. in industrial & management engineering from the Pohang University of Science and Technology (POSTECH) in 2006, where he is currently a full Professor with the Department of Industrial and Management Engineering. Before this, he was with the Information Systems Department of the Technology Management at Eindhoven University of Technology as a Postdoctoral Researcher from 2006 to 2009. Also, he was an Assistant/Associate Professor with the Ulsan National Institute of Science and Technology. He has published over 100 scientific papers in top-tier venues, including Decision Support Systems, Information Systems, the Journal of Information Technology, and the International Journal of Medical Informatics. His research interests include business process management, process mining, business analytics, simulation, and social network analysis.

Wil M.P. van der Aalst is an Alexander von Humboldt professor at RWTH Aachen University, leading the Process and Data Science (PADS) group. He is also the Chief Scientist at Celonis, part-time affiliated with the Fraunhofer FIT. His research interests include process mining, Petri nets, business process management, workflow automation, and data science. According to Research.com, he is the second-highest-ranked computer scientist in Germany and ranked 9th worldwide (2025 ranking). Van der Aalst is an IFIP Fellow, IEEE Fellow, and ACM Fellow, as well as an elected member of the Royal Netherlands Academy of Arts and Sciences, the Royal Holland Society of Sciences and Humanities, the Academy of Europe, and the German Academy of Science and Engineering.