Eurographics Conference on Visualization (EuroVis) 2019
M. Gleicher, H. Leitte, and I. Viola
(Guest Editors)

Volume 38 (2019), Number 3

A stable graph layout algorithm for processes

Robin J.P. Mennens[1,2], Roeland Scheepens[2] and Michel A. Westenberg[1]

[1]Department of Mathematics and Computer Science, Eindhoven University of Technology, The Netherlands
[2]ProcessGold, The Netherlands

Abstract

Process mining enables organizations to analyze data about their (business) processes. Visualization is key to gaining insight into these processes and the associated data. Process visualization requires a high-quality graph layout that intuitively represents the semantics of the process. Process analysis additionally requires interactive filtering to explore the process data and process graph. The ideal process visualization therefore provides a high-quality, intuitive layout and preserves the mental map of the user during the visual exploration. The current industry standard used for process visualization does not satisfy either of these requirements. In this paper, we propose a novel layout algorithm for processes based on the Sugiyama framework. Our approach consists of novel ranking and order constraint algorithms and a novel crossing minimization algorithm. These algorithms make use of the process data to compute stable, high-quality layouts. In addition, we use phased animation to further improve mental map preservation. Quantitative and qualitative evaluations show that our approach computes layouts of higher quality and preserves the mental map better than the industry standard. Additionally, our approach is substantially faster, especially for graphs with more than 250 edges.

CCS Concepts

• ***Human-centered computing*** → ***Graph drawings;*** • ***Applied computing*** → ***Business process monitoring;***

1. Introduction

Process mining [Aal16] is a discipline that is quickly gaining popularity among businesses. Using process mining, organizations are moving from a gut-feeling approach, in which organizations have to spend many hours to understand their processes, towards a fact-based approach, where organizations can make data-driven decisions. Processes are managed using process-aware information systems [VdA09] that record the process in an event log. This event log can then be used by a business analyst to understand and improve the process. The process is typically visualized as a weighted, directed graph (see Figure 1), and the analyst can explore the process by interactive filtering.

Interactive filtering causes the graph to be shown to change during the exploration. We can look at this from the perspective of dynamic graph drawing. In *online* dynamic graph drawing [BBDW17], graph layouts need to be computed for a sequence of graphs without knowing the full sequence from the beginning. In contrast, in *offline* dynamic graph drawing [BBDW17], the whole sequence of graphs is known up front. Our setting is offline in the sense that we know which graphs we can potentially encounter, since any filtered graph will be a subset of the data found in the event log, but it is also online because we do not know the exact sequence of graphs beforehand.

We want to preserve the mental map to reduce the cognitive effort of the user when a new graph layout is displayed. In this work, we use the three models of Misue *et al.* [MELS95] to represent the mental map. These models state that a layout adjustment should preserve the direction of node n to node m for each pair of nodes n and m, that nodes that are close together should remain close together, and that graphical objects in a region should stay in that region. We require our layout to remain *stable* to preserve the mental map of the user, but we also require our layout to be of high *quality*. Stability and quality are two conflicting requirements: graph layout stability helps preserve the mental map of the user [PHG06, ZKS11], but also restricts the graph layout algorithm in optimizing layout quality. A way of dealing with this conflict is to allow somewhat larger changes to the layout and to make use of animation and transitioning as a secondary approach to mental map preservation. We believe that the combination of layout stability and transitioning provides the best approach to preserving the mental map.

Our proposed approach also aims at showing the underlying process in an insightful way. Standard graph layout algorithms fail in this aspect, because they only use the graph topology to compute a layout and do not make use of the process data. For example, most processes have some sort of a main path [RBRB06, AEHK10] that can be thought of as the most frequent behavior. It would be sensible to centralize this path in the layout and also to keep it as straight as possible. While existing techniques fail to properly do this [GKN15], our approach tackles this problem, see Figure 1.

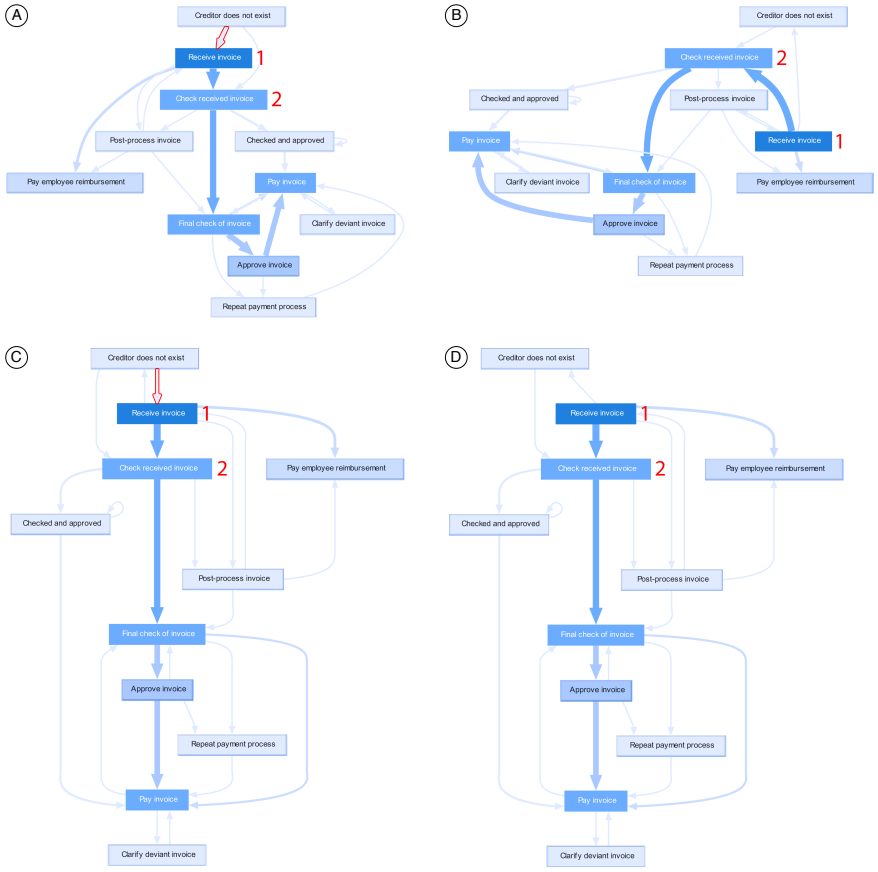

Figure 1: *Graphs A and B show the same process as graphs C and D, respectively. The layouts in A and B are computed by the industry standard [GKN15], where the actual process is poorly represented, while graphs C and D are computed by our novel graph layout algorithm where the process is easy to follow. Graphs B and D are obtained after removing the edge highlighted in red from A and C. As we can see, B differs significantly from A; especially note how nodes 1 and 2 swap vertically. Consequently, the mental map of the user is lost. On the other hand, C and D barely differ, preserving the mental map of the user.*

The contribution of our paper is a novel stable layout algorithm for process graphs that computes layouts that intuitively represent the semantics of the process. Our algorithm is based on the Sugiyama framework [STT81] and we introduce:

- A novel ranking algorithm;
- A novel order constraint computation algorithm;
- A novel crossing minimization algorithm called RELMINCROSS.

The scope of our work is the novel layout algorithm, which we have implemented in the business intelligence and process mining platform ProcessGold [1]. We make use of its existing graph visualization and interaction techniques, such as filtering, zooming, and panning. The visual attributes of the nodes and edges, such as color, shape, and edge width, are inherent to the ProcessGold platform as well. The visual encoding and design are therefore outside the scope of this paper.

1.1. Preliminaries

In process mining [Aal16], process data is stored in an *event log*, which contains sequences of events. An *event* is an occurrence of a certain *activity*, a recordable step in the process, *e.g.*, *Approve invoice* in an invoicing process as shown in Figure 1, at a certain time, for a specific *case*. A case, *e.g.*, a specific invoice, is then a sequence of activities ordered by the time of their events. Repetitions of the same activity are generally represented as self-loops and do not affect the topology of the graph; therefore, we ignore all repetitions in the sequences. In this paper, we are interested in sets of cases that follow the same sequence of activities, which are called *variations*. We denote the set of all variations by $\mathcal{V}$.

A *process graph* is a graph $G = (V, E)$ where V is a set of nodes that represent activities and E a set of directed edges based on the transitions between these activities observed in the event log, *i.e.*, an edge (n, m) is an ordered pair with $n, m \in V$. Each edge $e = (n, m)$ has an associated weight $weight(e) \in \mathbb{N}^+$, which is the number of transitions of (n, m). We further define the complete process graph $\overline{G} = (\overline{V}, \overline{E})$ where $\overline{V}$ and $\overline{E}$ are the sets of all nodes

and edges available in the event log. The user can interactively filter this graph by selecting sets of cases of interest. For any graph $G_i = (V_i, E_i)$ that is the result of this filtering, it then holds that $G_i \subseteq \overline{G}$, meaning $V_i \subseteq \overline{V}$ and $E_i \subseteq \overline{E}$.

In this work, graphs are drawn in a layered manner. Nodes are placed on *ranks*, which are parallel layers, such that, for any two nodes n and m for which $rank(n) = rank(m)$, we have that $y(n) = y(m)$. The rank assignment results in a direction for every edge that is not a self loop. An edge (n,m) is considered a forward edge when $rank(n) < rank(m)$ and a back edge when $rank(n) > rank(m)$. Hierarchical layouts are usually computed using an algorithm following the Sugiyama framework [STT81], which consists of the following steps:

1. **Cycle Removal:** To obtain a Directed Acyclic Graph, cycles are removed by reversing back edges and removing self loops.
2. **Rank Assignment:** Every node is assigned to a rank.
3. **Node Ordering:** Edges that cross multiple ranks are split up such that no edge *skips* a rank. This is done by replacing these edges by a chain of *virtual* nodes and edges for each rank it crosses. Following this, the order in which nodes (both real and virtual) are placed on the ranks is determined. This order determines the number of edge crossings in the graph and should therefore be optimized.
4. **Node Positioning:** x, y positions are computed for each node.
5. **Spline Drawing:** The edges are drawn using their virtual nodes.

1.2. Problem Description

Given an event log describing a process, we want to visualize this process using a weighted, directed graph layout of high quality that is readable and understandable and allows the user to quickly understand and build a mental map [MELS95, CP96] of the underlying process.

We want to preserve the mental map by ensuring graph layout stability. Stability, however, restricts the graph layout algorithm in optimizing layout quality, which is expressed in terms of aesthetic criteria [Pur02, HEHL13]. Depending on the domain [PCA02] and type of graph, some aesthetic criteria are more relevant than others. In related fields (business process visualization [RBRB06, ESK09, AEHK10, GPZ*14, BS15], flowchart visualization [San95, ST01], and workflow visualization [DDK*02, YLS*04]), commonly considered aesthetics include: minimal edge crossings, minimal edge bends, minimal edge length, maximal consistent flow direction, and no node overlap. In more generic settings, the minimal area and aspect ratio [JMM*16, JMS18] aesthetic criteria are often also considered, but in this work, we deem the above-listed aesthetic criteria as more important. Additionally, since horizontal edges, *i.e.*, edges on a single rank, either intersect nodes or are very short, they are hard to read and therefore are avoided. Lastly, we require our approach to be deterministic since the same data should always be represented by the same graph layout.

Considering the above, we formulate the following requirements for our approach:

R1 The graph layouts should intuitively represent the semantics of the actual process:
   a. The vertical order of nodes should represent the order in which activities take place in the process.
   b. The main path/structure of the process should be centralized in the graph layout.
   c. Sequential structures in the process should be as straight and short as possible.

R2 The number of edge crossings [PCJ97, Pur00] should be as small as possible.

R3 The number of edge bends should be as small and edge length should be as short as possible.

R4 The graph layout should not contain any *horizontal* edges.

R5 The mental map [MELS95] of the user should be preserved, *i.e.*, the graph layout should remain as stable as possible.

2. Related Work

Graph layout stability has been addressed in many works, especially in the context of dynamic graph drawing [BBDW17]. In this area, it is often essential to preserve the mental map of the user. In addition, since we have processes, we consider the field of business process visualization, which contains relevant work regarding visualizing process semantics.

2.1. Graph Layout Stability

Approaches towards ensuring graph layout stability can be split into different categories. Several works [BP90, HM98, Wad00] ensure graph layout stability by using *layout constraints* that reduce node movement. These techniques either let the user specify the constraints or the technique computes constraints based on which parts of a layout are not affected by a graph modification. However, in our case, users have no process information yet and therefore cannot properly specify constraints, and computing the constraints based on the previous layout results in a non-deterministic algorithm. *Online* dynamic graph drawing techniques [Nor95, CDBTT95, SP08, FT08] try to ensure graph stability by using the most recent graph layout as a basis for the new layout to be computed. By doing this, however, the techniques are non-deterministic. *Offline* dynamic graph drawing techniques [DGK01, EHK*03, GBPD04, RPD09, FWSL12] assume they already know the complete (linear) sequence of graphs $G_1, \ldots, G_n$ for which a layout must be computed. Since we do not have this information, none of these techniques are directly applicable to our problem. *Metric optimization* techniques [BW97, PKL04, LLY06] use metrics to measure different aspects of layout quality and layout stability. These metrics are often incorporated in a cost function that is then optimized. Consequently, depending on the metrics used, a trade-off between layout quality and layout stability can be achieved. However, it is not clear how to weigh the metrics and which metrics should be used to obtain the best results.

2.2. Business Process Visualization

In the field of business process visualization, graph drawing techniques focus on optimizing layout semantics for processes. Often, in contrast to our work, process information is already provided in the Business Process Execution Language (BPEL) [AAA*07]

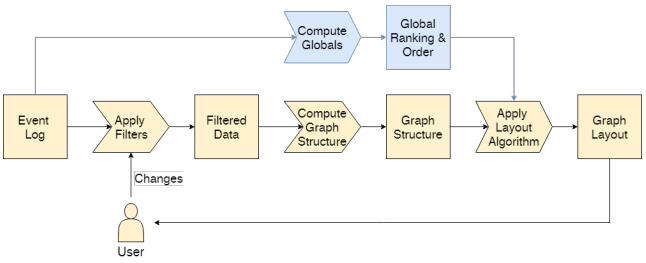

Figure 2: *An overview of the system. Beige represents the current system while blue represents our contribution.*

Sugiyama step	Our Approach
Cycle Removal	Follows from Global Ranking
Rank Assignment	
Node Ordering	RELMINCROSS using Global Order
Node Positioning	See Gansner *et al.* [GKNV93]
Spline Drawing	See Mennens *et al.* [Men17]

Table 1: *Our approach follows the Sugiyama framework.*

or Business Process Model and Notation (BPMN) [CT12] language. With respect to graph stability, some techniques [DDK*02, YLS*04] are incremental and therefore suffer from the same drawbacks as online dynamic graph drawing approaches. Effinger *et al.* [ESK09] employ a technique based on a Bayesian paradigm [BW97], which therefore suffers from the same drawbacks as the metric optimization techniques.

3. Approach

In Figure 2, we show an overview of the system. Elements in beige represent how the system currently works, while blue elements highlight our novel contributions. The current process starts with an event log, containing the process data, which can be interactively filtered by the user such that only the data of interest remains. After that, the filtered data is converted into a directed weighted graph structure. Activities are converted into nodes, while an occurrence of two consecutive activities contributes to the weight of the respective edge. Finally, a graph layout algorithm computes a hierarchical layout [STT81], which is shown on the screen. This whole process is repeated when the user applies new filters. Mining the graph structure and the variations from the event log is outside of the scope of this paper and is considered to be given in a process mining setting.

Our approach is based on the Sugiyama framework [STT81], where we use a *global ranking* and *global order* based on the event log to preserve horizontal and vertical node order (R1a.) and show the structure of the process (R1b. and R1c.). This enables the preservation of the mental map [MELS95] when the user explores subsets of the event log through filtering (R5). More specifically, the global ranking defines a ranking for every graph $G_i \subseteq \overline{G}$, while the global order defines order constraints for the nodes in G_i for every rank. The global ranking and order only have to be computed once for an event log because they can be reused during the graph layout computation for every $G_i \subseteq \overline{G}$. In Table 1, we show how our approach fits into the Sugiyama framework. Note that the *cycle removal* step is no longer required due to the use of a global ranking. For the *node positioning* and *spline drawing* steps, we use existing methods.

Even under minimal change, change blindness [SL97] should be considered. Research [BB99, SIG07, SI08, ZKS11, BPF14, AP16] shows that animation helps a user follow the changes between layouts and thereby aids mental map preservation. Therefore, similar to other works [FE02, FT08, RPD09, ZKS11, BPF14], we implement phased animation to further improve mental map preservation (R5)–see supplementary video.

3.1. Global Ranking

The global ranking is a partitioning of the set of all nodes $\overline{V}$. More specifically, the global ranking $GR = (\psi_0, ..., \psi_n)$ is an ordered list of sets of nodes. Consequently, every node $n \in \overline{V}$ is present on exactly one global rank ψ_i. Recall that we disallow horizontal edges (R4), and therefore, GR should be computed such that no edge (which is not a self-loop) $(n,m) \in \overline{E}$ is horizontal.

3.1.1. Building the global ranking

To represent the underlying process (R1), we cannot simply apply a typical ranking algorithm that determines a ranking based on the structure of the input graph [GKN15] since that does not take the structure of the process into account, as shown in Figures 1A and 1B. Instead, we use the event log to discover the main structure of the process. For example, in Figure 1, this main structure consists of the path ⟨*Receive invoice*, *Check received invoice*, *Final check of invoice*, *Approve invoice*, *Pay invoice*⟩. The main structure of a process (graph) can be seen as (a set) of path(s), *i.e.*, sequence(s) of activities (cases). By using the cases in the event log, we already obtain some semantic information. However, by just considering the cases, we do not know yet which cases contribute to the main process structure; therefore, we consider the variations. The variation $v \in \mathcal{V}$ that contains the most cases, *i.e.*, has largest $|v|$, describes the most frequent process behavior. Subsequent variations of smaller size describe less frequent behavior. Therefore, our algorithm (see Algorithm 1) processes the variations $\mathcal{V}$, obtained from the event log one by one from most important to least important (lines 2 and 3). While doing this, we incrementally build a hierarchical graph structure $\mathbb{RG}$ such that all global ranking requirements (R4) remain satisfied (lines 4 and 5). After processing all variations, $\mathbb{RG}$ contains all nodes $\overline{V}$. Every node n in $\mathbb{RG}$ is positioned on a rank, and while building $\mathbb{RG}$, the rank of n may change. Therefore, we normalize the node ranks (line 6) to make sure the smallest rank value becomes 0. Finally, we extract the global ranking from $\mathbb{RG}$ by directly using the rank values of the nodes (line 7).

In our algorithm, we start by sorting the variations based on variation importance $imp(v)$ (line 2). We have experimented with several definitions of variation importance and found that the sorting

result is quite sensitive to the choice [Men18]. In our setting, the best performance is obtained by $imp(v) = \sum_{w \in W} w^2 \cdot |v|^2$, where W is a multi-set that contains the edge weights of the edges present in v. By squaring the weights w, we prioritize variations that contain edges with a high weight, and by squaring $|v|$, we prioritize variations that occur frequently overall. We then process the variations from most important to least important. This means that we *discover* the most important process behavior first. By doing this, we ensure that the vertical order of the nodes represents the order of the nodes in the process (R1a.).

Algorithm 1 Compute Global Ranking

```
1: procedure ComputeGlobalRanking(𝒱)
2:     ImportanceSort(𝒱)
3:     for each v in 𝒱 do
4:         while s = NewSequence(v) do
5:             Update ℝ𝔾 given s
6:     NormalizeRanks(ℝ𝔾)
7:     GR ← ExtractGR(ℝ𝔾)
```

A variation can be translated into a sequence of interleaved nodes and edges. For example, let v be a variation with the sequence $\langle A, B, C \rangle$. It can be translated into $\langle A, (A,B), B, (B,C), C \rangle$. When processing a variation, we extract all continuous sequences that only contain nodes and/or edges that are not yet in the rank graph $\mathbb{RG}$ (line 4) and add these sequences to $\mathbb{RG}$ based on the *type* of the sequence (line 5). We distinguish six different types of sequences, which are illustrated in Figure 3A. The distinction is made based on the way that the head and tail of a sequence are connected or not to $\mathbb{RG}$. Furthermore, isolated nodes and edges are distinguished as special cases. Adding a sequence to $\mathbb{RG}$ is done such that no horizontal edges are created (R4). The update procedure for every sequence type is explained below. Since types I and V, and III and IV are added in a similar manner, we explain them together. While processing sequences, we keep track of connected components for the nodes/edges we have already seen.

Type I and Type V: For a sequence s that starts and ends with a node, we place the first node in s on the lowest rank in $\mathbb{RG}$ and all subsequent nodes on subsequent ranks. The node(s) in s form a new connected component.

Type II: When we have only a single edge (n,m), this implies that n and m have been seen before, and therefore, are already assigned to a rank in $\mathbb{RG}$. Consequently, given $\mathbb{RG}$, we can encounter four different scenarios:

1. When (n,m) is a forward edge, we can simply add it.
2. When (n,m) is a back edge and n and m are part of the same component, we do nothing. While in some cases, it would be possible to move m and other nodes such that we transform (n,m) into a forward edge, this is generally not beneficial because this will make some other (more important) edges longer (R1c.).
3. When (n,m) is an edge that connects two connected components, we create a forward edge of minimal length by moving the component containing m down, such that $rank(m) = rank(n)+1$. The two connected components now form a single connected component.
4. When (n,m) is a horizontal edge, we use Algorithm 2 with parameters n, m, and 1 to move m, and all nodes in $\mathbb{RG}$ that are reachable via a downward graph traversal, one rank down to create a forward edge (R4). With the downward traversal Traverse (line 2), we discover nodes that also need to be moved such that we do not create horizontal edges (R4). Traverse ignores the direction of edges, is started from m, and only edges of length at most *numRanks* are traversed downward. Nodes that are visited during the downward traversal are marked as such. An example of running Algorithm 2 is shown in Figure 3B. On the left and right, we have $\mathbb{RG}$ before and after running Algorithm 2 respectively. As we can see, the horizontal edge is *fixed* by moving m and all nodes reachable via a downward traversal.

Algorithm 2 Shift Nodes

```
1: procedure ShiftNodes(n, m, numRanks)
2:     Traverse(m, numRanks)
3:     for each node x ∈ ℝ𝔾 do
4:         if x was visited by Traverse and x ≠ n then
5:             rank(x) ← rank(x) + numRanks
```

Type III and IV: Let (n,x) and (y,m) be the first and last edge in a Type IV and III sequence, respectively. To create forward edges (R1a.), nodes in a Node-Edge sequence (III) are placed above m and nodes in an Edge-Node sequence (IV) are placed below n, see Figure 3C.

Type VI: Let s be an Edge-Edge sequence and let (n,x) and (y,m) be the first and last edge, respectively. Similar to the Type II Single Edge case, nodes n and m are already present in $\mathbb{RG}$ and, therefore, we can encounter three different scenarios:

1. When n and m are part of different connected components, we place the sequence of nodes on the ranks below n to create forward edges (R1a.). Then, we turn (y,m) into a forward edge by moving the component containing m down such that $rank(m) = rank(n) + 1$ (R3).
2. When $rank(m)$ is the same as or higher than $rank(n)$, we place the sequence of nodes in between n and m. Depending on the number of nodes and number of ranks between n and m, there are two scenarios: First, if there are enough ranks between n and m, we simply place the sequence of nodes on those free ranks (starting on the rank below n). Second, if the number of free ranks is too small, we make space by running Algorithm 2 with parameters y, m, and $rank(y) - rank(m) + 1$. An illustration of this procedure is shown in Figure 3D.
3. When $rank(m)$ is smaller than $rank(n)$, there are quite some ways in which we can handle this. Considering, however, that m is already on a lower rank than n, it is very likely that m occurs before n in the process. Hence, the sequence of nodes in s represents a sequence of activities that *loop back* to an earlier activity in the process. Therefore, to show that we go back in the process (R1), we add the nodes in s such that we obtain a sequence of back edges. Similar to before, if there are enough free ranks between n and m, we simply place the sequence of nodes between n and m; otherwise, we make room using Algorithm 2 as shown in Figure 3E.

Since in the worst case, every variation $v \in \mathcal{V}$ contains all edges $e \in$

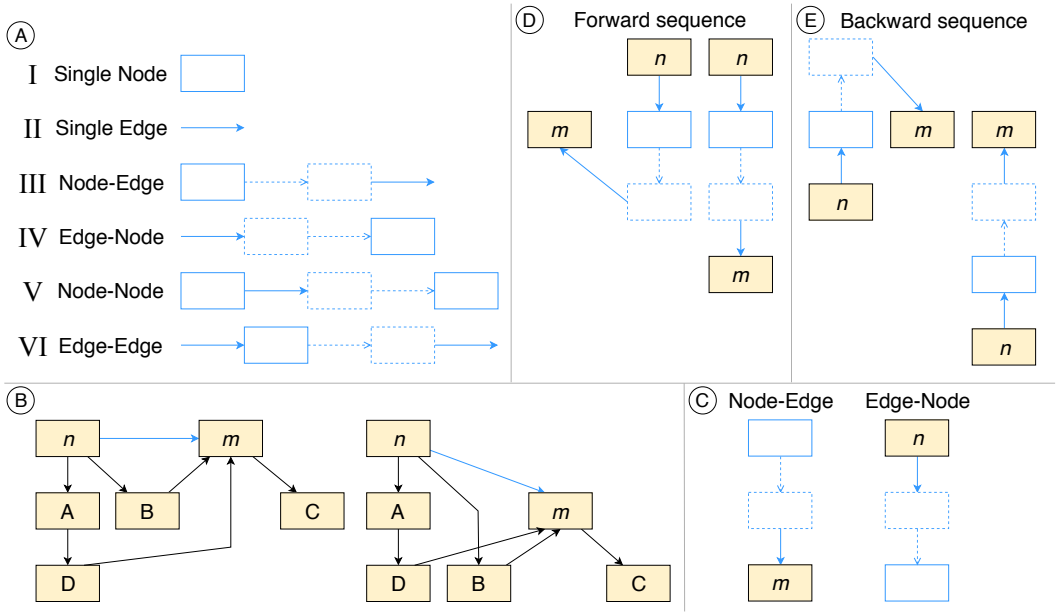

Figure 3: *Connected blue elements, where dashed nodes and edges represent an arbitrary number of nodes and edges, represent sequences that still need to be added or are being added to $\mathbb{RG}$. Beige rectangles and black arrows represent nodes and edges in $\mathbb{RG}$. A) The different types of sequences we consider. B) An illustration of running Algorithm 2 with parameters: n, m, and 1 with before and after shown left and right, respectively. Edge (n,m) is the horizontal edge we are fixing. Note that node D is not moved down because edge (D,m) has a length that is longer than 1. C) A visual representation of how we create forward edges when processing sequence types Type III Node-Edge or Type IV Edge-Node. For Type III, we add the chain of nodes to the ranks above node m and for Type IV, we add the chain of nodes to the ranks below node n. D) and E) An illustration of how sequence Type VI Edge-Edge is handled for forward and backward sequences respectively when there are insufficient ranks between n and m. Using Algorithm 2, we obtain sequences of forward and back edges, respectively.*

$\overline{E}$, computing the variation importance $imp(v)$ runs in $O(|\mathcal{V}||\overline{E}|)$, while the sorting runs in $O(|\mathcal{V}|\log|\mathcal{V}|)$ [CLRS09]. While processing variations, every node and edge in $\overline{V}$ and $\overline{E}$, respectively, is handled exactly once. Moreover, the worst case running time for processing the sequence types, except for Type I, is $O(|\overline{V}|)$. This leads to a total running time of $O(|\mathcal{V}|\log|\mathcal{V}|+|\mathcal{V}||\overline{E}|+|\overline{V}||\overline{E}|)$. The storage complexity of $\mathbb{RG}$ is $O(|\overline{V}|+|\overline{E}|)$ since we add all nodes and edges to this structure. The storage complexity of the global ranking GR is $O(|\overline{V}|)$ since it stores a global rank per node.

3.1.2. Applying the global ranking

To apply the global ranking in a layout for some $G_i \subseteq \overline{G}$, we can directly use GR to obtain a valid ranking. Then, since not necessarily every node is present, we remove empty ranks to make sure the graph is as compact as possible.

While $\overline{E}$ is the set of all edges available in the event log, we can still encounter other edges when filtering out an activity. For example, filtering out activity B from the sequence $\langle A,B,C\rangle$ results in the edge (A,C). This is a rare occurrence, as the user generally filters whole cases. Also, these edges can simply be added to the graph, except when A and C happen to be on the same global rank, leading to a horizontal edge (R4). To still be able to compute a graph layout, these horizontal edges are added in a similar way to Type II sequences. To preserve the node order (R5) of the global ranking, an extra rank is created, as shown in Figure 4.

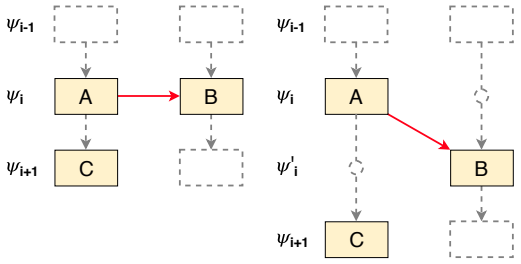

Figure 4: *A horizontal edge (red) on a rank ψ_i is fixed. Dashed nodes and edges represent an arbitrary set of nodes and edges on the adjacent ranks, and circles represent virtual nodes that are created. By moving node B down to a new rank ψ'_i, we obtain a forward edge. The original adjacent ranks to ψ_i are are not modified.*

3.2. Global Order

In the Sugiyama framework [STT81], edges that cross multiple ranks are split up into a chain of *virtual* nodes and edges such that no edge *skips* a rank, see Figure 5. Horizontal positions are then determined in the node ordering and positioning steps. First, the order of real and virtual nodes is determined for every rank and then actual x-coordinates for the nodes are computed based on the node order [GKNV93].

In traditional layout algorithms [GKNV93], the horizontal movement of nodes is unconstrained. We, however, constrain horizontal node movement in the node ordering step by first computing a global order that defines order constraints on node pairs that are on the same rank. Then, by using a crossing minimization algorithm that adheres to the specified order constraints, we make sure the order constraints are satisfied.

3.3. Building the global order

We define the skeleton of a process graph as the set of all *node sequences*. A node sequence is a new and continuous sequence, as discovered while processing the variations in the global ranking computation (see Algorithm 1 line 4), containing at least one node, see Figure 5C. A *sequence edge* is an edge part of a node sequence. Note that the sequence edges have been split up into chains of virtual nodes and edges. The order of all nodes in the skeleton is constrained by the global order. Edges that are not part of the skeleton, however, are unconstrained. This allows us to minimize edge crossings (R2) and improve overall layout quality (R3) while maintaining stability in the main structure of the process (R5).

The global order defines the order in which the node sequences should be placed for every global rank separately. To this end, based on the global ranking and the skeleton of the process (graph), we define the hierarchical, undirected, node sequence graph $NSG = (V_{nsg}, E_{nsg})$. The set of nodes V_{nsg} contains *real sequence nodes*, representing the nodes in the skeleton (*i.e.*, all nodes in $\bar{V}$), and *virtual sequence nodes*, representing the virtual nodes in the skeleton. The set of edges E_{nsg} consists of the virtual edges that represent the sequence edges in the skeleton. Since every rank in the NSG maps to a global rank, the real and virtual sequence nodes represent the presence of a node sequence on a certain global rank. Consequently, any order permutation of the sequence nodes $n \in V_{nsg}$ results in a valid global order. The graph layout in Figure 5C is computed using the global order and the corresponding node sequence graph is shown in Figure 5B, where squares and circles represent real and virtual sequence nodes, respectively. Note that the number in each node corresponds to the sequence number in Figure 5C.

We consider the most frequent path as the most important behavior (R1b.). We call this path the *backbone* of the graph. Since this path may not cover all ranks, we define the backbone as the set of real or virtual sequence nodes, such that, for every rank r in the node sequence graph, the sequence node that is discovered the earliest while building the global ranking belongs to the backbone. For example, in Figures 5A and 5B, the nodes with a red border belong to the backbone. The backbone then forms the center of the graph layout and all other nodes should be positioned around this backbone (R1b.).

Let $nodes(s)$ be the set of nodes part of a sequence s. We then define the *connectedness* of two sequences s_i and s_j as $conn(s_i, s_j) = \sum_{e \in \gamma(s_i, s_j)} weight(e)$ where $\gamma(s_i, s_j)$ is the set of edges that start/end in $nodes(s_i)$ and start/end in $nodes(s_j)$.

Based on our requirements and the concepts of the backbone and connectedness, we define the following requirements for our global order:

RGO1 **Connectedness:** Sequences that have a high connectedness should be placed next to or close to each other. This especially holds for node sequences that have a high connectedness to the backbone. By doing this, edges between adjacent node sequences are shorter (R3) and are less likely to cross (R2).

RGO2 **Crossings:** We do not want unnecessary sequence edge intersections (R2). Since these sequence edges are part of the skeleton, they are considered more important for the user to understand the process. For example, the node sequence graph as illustrated in Figure 5B satisfies this property.

RGO3 **Balance:** Given the backbone, for every rank, we wish to balance the sequence nodes around the backbone. By doing this, the backbone ends up in the center of the node sequence graph (R1, R1b.).

Algorithm 3 Compute Global Order

```
1: procedure ComputeGlobalOrder(NSG(V_nsg, E_nsg))
2:     for each rank r in ranking(NSG) do
3:         ConnectednessSort(r)
4:     NSG' ← NSG \ backbone
5:     comp ← FindComponents(NSG')
6:     Balance(comp)
```

Our global order algorithm is shown in Algorithm 3. ConnectednessSort(r) on line 3 sorts the sequence nodes n on rank r based on their connectedness to the backbone as shown in Figure 5A. Then, to both minimize crossings in the skeleton and to balance the graph (R2 and R1b.), we first find a set of connected components (see line 5). Since we want to balance around the backbone, we find components in NSG', which is the NSG without the backbone nodes (see line 4). These components can be placed on either side of the backbone without causing extra crossings. Then, $Balance(comp)$ on line 6 considers the components from large to small, where the size of a component is defined by the number of nodes in the component. For every component, $Balance(comp)$ moves the component to the left of the backbone if doing so improves the balance, *i.e.*, if the ratio between the number of nodes left and right of the backbone improves. Additionally, when moving a component to the left of the backbone, the connectedness sort order is preserved, *i.e.*, sequence nodes that are closer to the backbone than other sequence nodes (on the same rank) always remain closer. Figure 5B shows the NSG after balancing the sequence nodes. The global order can now be obtained directly from the node order in the NSG.

Sorting by connectedness takes $O(|V_{nsg}|\log|V_{nsg}|)$, since we sort all sequence nodes. Finding all connected components can be done by traversing the NSG, which takes $O(|V_{nsg}|+|E_{nsg}|)$ time. Balancing takes $O(|V_{nsg}|)$ since we have at most $|V_{nsg}|$ components. Therefore, the worst case running time of the global order computation is $O(|V_{nsg}|\log|V_{nsg}|+|E_{nsg}|)$. The NSG contains all nodes in $\bar{V}$ and contains at most $\bar{E}$ edges. Hence, the storage complexity of the NSG is $O(|\bar{V}|+k|\bar{E}|)$ where k is the average number of virtual nodes per edge. Note that we can have at most $|\bar{V}|$ ranks and therefore k is at most $|\bar{V}|$. In practice, however, k is much smaller. The global order stores the order for all nodes and sequence edges, resulting in a storage complexity of $O(|\bar{V}|+|\bar{E}|)$.

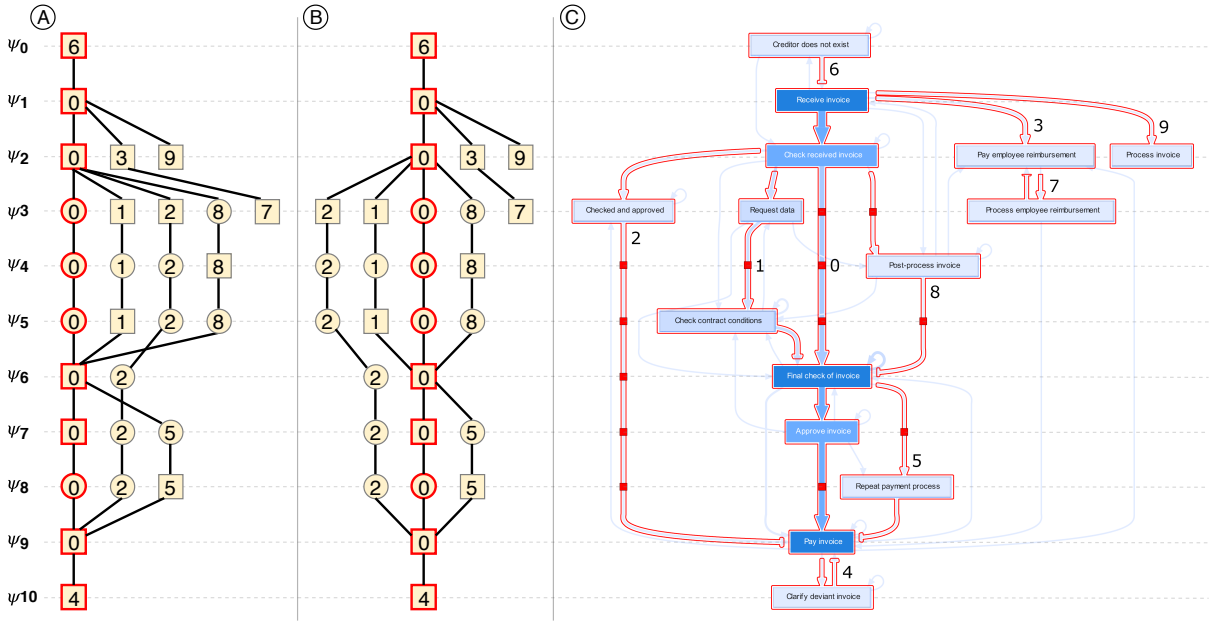

Figure 5: *Every ψ_i represents a global rank. In A and B, squares and circles represent real and virtual sequence nodes, respectively. The numbers match the numbering in C and represent the order in which the node sequences are discovered by the global ranking algorithm. Sequence nodes with a red border belong to the backbone. A) The node sequence graph after sorting each rank based on backbone connectedness. B) The final node sequence graph. C) Illustration of the order in which the node sequences (outlined in red) are discovered by the global ranking algorithm. The number next to a red outline indicates the order of discovery. Red squares represent virtual sequence nodes.*

3.4. Crossing Minimization

During the node ordering step for some graph $G_i \subseteq \overline{G}$, the global order defines order constraints for both real and virtual sequence nodes for every rank ψ_i. Note that non-sequence virtual nodes are unconstrained and can, therefore, be placed in any order. Ideally, we want to compute an order permutation for every rank such that we both satisfy the order constraints and minimize the number of edge crossings (R2). Unfortunately, edge crossing minimization is NP-Complete [EMW86]. Therefore, in practice, algorithms reduce edge crossing minimization to a sequence of one-sided two-level [GKNV93, For04] crossing minimization problems. In this work, we present a novel crossing minimization algorithm where we have a sequence of constrained one-sided two-level crossing minimization problems [For04]. More specifically, the computed order permutation should adhere to the global order.

Algorithm 4 Crossing Minimization

```
1: procedure CROSSING MINIMIZATION(G)
2:     order ← InitOrder()
3:     best ← order
4:     for i ← 0 to MaxIterations do
5:         wMedian(order, i)
6:         Transpose(order)
7:         if crossing(order) < crossing(best) then
8:             best ← order
9:     return best
```

Our algorithm, RELMINCROSS (see Algorithm 4), follows the same steps as the crossing minimization algorithm by Gansner *et al.* [GKNV93] and therefore has a similar running time. In our algorithm, INITORDER is replaced and WMEDIAN and TRANSPOSE are modified. More specifically, the sort and exchange operations in WMEDIAN and TRANSPOSE respectively are modified such that swapping nodes that are constrained by the global order is disallowed. A difference in running time is that Gansner *et al.* [GKNV93] run Algorithm 4 twice – with different initial orders – to obtain the best results. Since INITORDER for RELMINCROSS always computes the same initial node order, this is not necessary, improving the overall running time.

Given some graph $G_i \subseteq \overline{G}$, our INITORDER computes $init(n) \in [-1,1]$ values for all nodes $n \in V_i$ and then determines the initial order by sorting the nodes on $init(n)$. Negative and positive *init* values represent nodes left and right of the backbone, respectively, and backbone nodes have an *init* value of 0. Intuitively, initialization is done such that the skeleton of the process graph is initialized based on the global order and remaining edges are placed based on the skeleton.

Using the fixed order given by the global order, INITORDER first computes the *init* values for all sequence nodes for each rank separately. First, the backbone node on every rank is identified and assigned value 0. Following this, the sequence nodes left and right of the backbone node are uniformly assigned *init* values in the ranges $[-1,0)$ and $(0,1]$, respectively, based on the global order. In Figure 6, we show a graph layout that was computed using RELMIN-

CROSS. As we can see, all backbone nodes have *init* value 0, and nodes further to the left and right, with respect to the backbone, have lower and higher *init* values, respectively. Following this, INITORDER computes the *init* values for all non-sequence virtual nodes. Let (x,y) be a non-sequence edge. The $init(n)$ values of the virtual nodes belonging to (x,y) are then computed as follows:

$$init(n) = \begin{cases} init(x) & |init(x)| > |init(y)| \\ init(y) & |init(x)| \leq |init(y)|,\ init(y) \neq 0, \\ & init(x) \neq 0 \\ \varepsilon & init(x) = init(y) = 0, \\ & n \text{ is part of a back edge} \\ -\varepsilon & init(x) = init(y) = 0, \\ & n \text{ is part of a forward edge} \end{cases}$$

Each non-sequence virtual node n is given the *init* value of its start or end node furthest away from the backbone to ensure edges are as straight and short as possible (R1c. and R3). For example, in Figure 6, the non-sequence virtual nodes and their computed $init(n)$ values are shown in green. If both $init(x)$ and $init(y)$ are equal to zero, the edge starts and ends at the backbone. If we were to set $init(n)$ to zero for these nodes, then it is ambiguous on which *side* of the backbone n should be initially placed. Therefore, based on the direction of the edge, we set $init(n)$ to a small number ε such that, during the crossing minimization itself, edges will not arbitrarily cross the backbone (R2).

As discussed in Section 3.1.2, in rare cases we have to add rank(s) to fix horizontal edges that may occur due to filtering. Adding such a rank results in longer edges and therefore in extra virtual nodes (see the circles in Figure 4). These virtual nodes do not have a defined global order and are therefore always unconstrained.

Once all *init* values have been computed, every rank is sorted by these values. Note that the *init* values of sequence nodes on the same rank are always unique. The *init* values of non-sequence virtual nodes, however, can be the same. When this happens, the nodes are sorted based on the length of their edges where we place the longer edge further away from the backbone to reduce edge crossings (R2). If the edge lengths are the same, the most important edge, *i.e.*, with the highest weight, is placed closest to the backbone (R1b.). Observe that the global order constraints are satisfied after sorting.

After initializing the order, edge crossings can still be present. Therefore, the remainder of the algorithm iteratively uses WMEDIAN and TRANSPOSE to reduce the remaining edge crossings (R2) without breaking any order constraints (R5).

4. Evaluation

We evaluate our approach both quantitatively and qualitatively. For both evaluations, we compare our layouts to the output of *dot* [GKN15], which is part of the Graphviz software [2] and is based on the work of Gansner *et al.* [GNV88, GKNV93]. Because *dot* provides high quality graph drawings for a variety of cases and since it can be easily integrated, it is the current industry standard used to draw hierarchical directed graphs that represent a process.

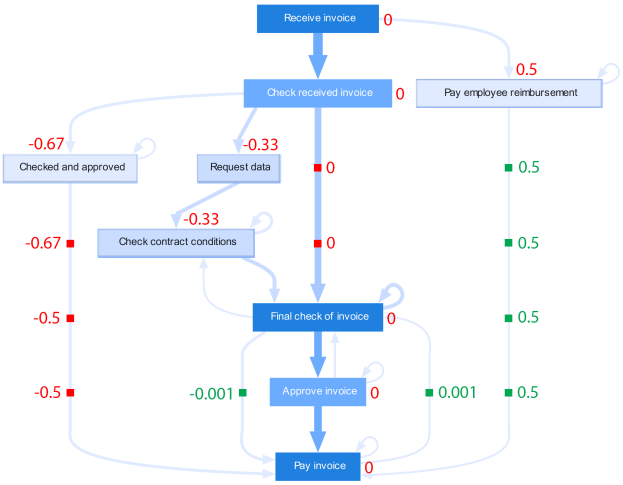

Figure 6: *A graph layout computed using* RELMINCROSS *with* $\varepsilon = 0.001$. *Red and green numbers indicate the* $init(n)$ *values for the sequence and non-sequence nodes respectively. Red squares represent virtual sequence nodes while green squares represent non-sequence virtual nodes.*

We first introduce quality and stability metrics [DG02] that 'measure' aesthetic criteria and the amount of change, respectively.

Definition 4.1 (Quality Metric) Given a graph $G = (V,E)$ for which a layout has been computed. A quality metric is a function $QM_\lambda : G \to R_0^+$ that quantifies a certain graph layout aesthetic. A unique name representing the metric is λ. A lower value for a quality metric implies that G adheres more to the aesthetic criterion. For example, for a graph layout G that has no edge crossings, the number of edge crossings is denoted by $QM_{crossings}(G) = 0$.

Definition 4.2 (Stability Metric) Given two graphs $G = (V,E)$ and $G' = (V',E')$ for which layouts have been computed. A stability metric is a function $SM_\lambda : (G,G') \to R_0^+$ that quantifies the amount of change between G and G' depending on what we measure. A unique name representing the metric is λ. When $SM_\lambda = 0$, there is no difference between G and G' for metric λ.

Based on a literature investigation and given our requirements, we selected the following metrics (the exact specifications are provided in the supplementary material):

- **Running Time** $QM_{time}(G)$: time in milliseconds required to compute a layout.
- **Edge Crossings** $QM_{crossings}(G)$: the sum of multiplied edge weights of edge pairs that cross.
- **Average Edge Length** $QM_{avg\_length}(G)$: the weighted average edge length.
- **Edge Bends** $QM_{bends}(G)$: the sum of the number of edge bends in an edge multiplied by the respective edge weight.
- **Back Edges** $QM_{back\_edges}(G)$: the sum of the edge weights of the back edges in the layout.
- **Flow Direction** $QM_{flow}(G)$: the ratio between edge segments

that are (approximately) horizontal and edge segments that are (approximately) vertical.
- **Area** $QM_{area}(G)$: the area of the layout.
- **Relative Euclidean** $SM_{rel\_eucl}(G,G')$: the sum of the change in relative distance between node pairs that are in both layouts.
- **Hausdorff** $SM_{hausdorff}(G,G')$: the maximum of the distances moved by nodes that are in both layouts.
- **Orthogonal** $SM_{orthogonal}(G,G')$: the sum of the change in direction (angle) between node pairs that are in both layouts.
- **Epsilon-Cluster** $SM_{cluster}(G,G')$: the epsilon-cluster change between the two layouts.
- **Edge Shape** $SM_{edge\_shape}(G,G')$: the change in the shape of the edges.

4.1. Quantitative Evaluation

The quality and stability metrics allow us to quantitatively compare our algorithm and *dot*. We use 13 datasets (see supplementary material), which represent real-world processes. For every dataset, we ran 500 tests on both methods. Each test involved a pair of randomly generated graphs $G_1, G_2 \subseteq \overline{G}$, which were obtained by randomly removing a subset of the edges $\overline{E}$ and then removing all nodes $n \in \overline{V}$ that became disconnected. Consequently, G_1 and G_2 are random sub-graphs of $\overline{G}$. Since all tests are independent and the sample sizes are large, we can assume that the test results are normally distributed (per algorithm, per dataset). To determine which algorithm performs statistically better, we applied a one-way t-test (significance level $\alpha = 0.05$) for every metric (per dataset).

The aggregated results are shown in Figure 7. The two algorithms are on the rows, and the columns (except the last three) represent the metrics. A cell value is a count that expresses for how many of the datasets the related algorithm performed significantly better. For example, our algorithm is significantly faster (see column *(QM) Time*) for all 13 datasets. The last three columns present aggregated values: *qSum* the sum of all quality metric counts, *sSum* the sum of all stability metric counts, and *Sum* the total sum of all counts.

The results show that our algorithm outperforms *dot* significantly. For the stability metrics, our algorithm performs significantly better on all datasets. This was expected because we specifically designed our algorithm to compute stable layouts. Interestingly, our algorithm also outperforms *dot* for most of the quality metrics, despite the trade-off between layout stability and layout quality. The only two metrics where *dot* performs better (or equally well) are *Average Edge Length* and *Area*. This reflects the design of *dot*, which tries to keep edges short and produce compact layouts [GKNV93].

The global ranking and global order are reused during every graph layout computation and therefore only need to be computed once for a given dataset. Therefore, the total running time needed to compute a single graph layout is the sum of running time complexities of each of the algorithmic steps listed in Table 1. Assigning ranks to nodes can be done directly based on the global ranking and therefore takes $O(|\overline{V}|)$ time. In contrast, *dot* needs to run both a cycle removal and a rank assignment algorithm [GKNV93]. As discussed in Section 3.4, our crossing minimization algorithm has a

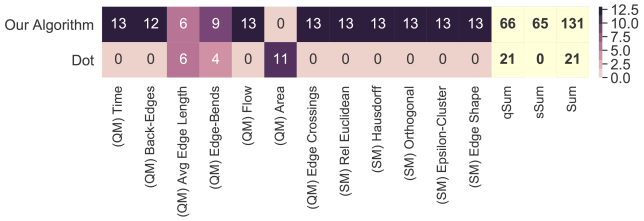

Figure 7: *The combined statistical test results.*

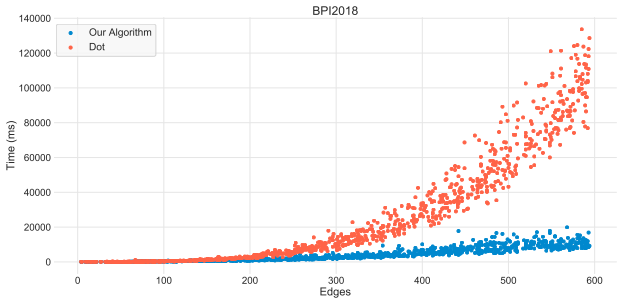

Figure 8: *The running time results for the BPI2018 dataset*

similar running time as the algorithm of Gansner *et al.* [GKNV93]. Overall, compared to *dot*, we only use the same algorithm for the node positioning step. The running time results for one of the datasets (BPI2018) are shown in Figure 8. The plot shows that our algorithm is substantially faster than *dot* for larger graphs. Running time results for the remaining datasets are provided in the supplementary material.

4.2. Qualitative Evaluation

To determine which algorithm is preferred by actual users, we also performed a user study in which 14 process mining experts participated. Participants used both our algorithm and *dot* and answered questions about their experience. The results show that participants preferred our algorithm both for its layout quality and stability. Participants found our graph layouts more readable because the main path is positioned in the center and edges are straighter, which makes them easier to follow. Additionally, participants stated that the stability and animated transitions made it easier to follow changes. We provide more details of the experimental setup and the results in the supplementary material.

5. Discussion

The global ranking and global order constraints achieve the goal of an intuitive representation of the semantics of the process (R1), as demonstrated in Figures 1C and 1D. There are also downsides to constraining the layout by the global ranking and global order, however. One aspect is that the ranking and order are computed for the unfiltered process data, which may result in suboptimal layouts

for certain filters. Another issue is that the strict ranking may result in unnecessary extra ranks, which take up vertical space but do not improve the layout or the interpretation of the process. An example can be seen in Figures 1C and 1D. The global ranking puts *Post-process invoice* below the rank of *Checked and approved* and above the rank of *Final check of invoice*. However, this is not actually necessary as *Post-process invoice* and *Checked and approved* could be on the same rank in this graph. An issue of the global order is that it can result in unbalanced layouts. In Figures 1C and 1D, for example, the global order enforces *Repeat payment process* right of the edge (*Approve invoice*, *Pay invoice*). The layout would be more balanced if this node were placed left of this edge.

The global ranking and global order help to preserve the mental map of the user. The global ranking constrains the vertical order of the nodes, and the global order constrains the horizontal movement of nodes. Together, these constraints reduce changes in the layout and keep the layout stable (R5). A disadvantage is that these constraints can cause extra edge crossings. We partially solve this problem by only constraining sequence nodes and edges. Non-sequence edges can be positioned freely, allowing the graph layout algorithm to reduce edge crossings.

6. Conclusion

We have proposed a novel layout algorithm for processes, which is based on the Sugiyama framework. Our approach makes use of process data in the event log and provides three main contributions. The first is a novel ranking algorithm that computes a global ranking based on the process data. By using the global ranking during graph layout computation, we obtain layouts that intuitively represent the actual process. Also, the global ranking stabilizes the layout by constraining the vertical order of nodes. The second contribution is a constraint computation algorithm that computes a global order based on the process data. The global order constrains the horizontal movement of nodes and is used in our third contribution: a crossing minimization algorithm that maintains the global order constraints. Finally, to make it easier for a user to follow the changes between two graph layouts, we use phased animation to further help preserve the user's mental map.

We evaluated our approach by a quantitative and qualitative evaluation. The quantitative evaluation shows that our novel algorithm produces graph layouts of higher quality and higher stability than the industry standard. Additionally, our algorithm is significantly faster, especially for larger process graphs. In the qualitative evaluation, we found similarly favorable results for our approach.

As part of future work, we plan to extend our approach to processes with multiple backbones. In this paper, we have implicitly assumed that there is only a single backbone by just considering the most frequent path. While this is often the case in practice, it does not hold for all processes. This would require a method to mine multiple subprocesses. Furthermore, the global ranking may create undesirable extra ranks. It would be interesting to investigate techniques in which node constraints (in our case, the global ranking) are 'relaxed' in some scenarios (similar to Görg *et al.* [GBPD04]). This may prevent the creation of these extra ranks but would also reduce layout stability.

Acknowledgments

This research work was carried out at ProcessGold in cooperation with the Eindhoven University of Technology.

References

[1] ProcessGold - business intelligence and process mining platform. https://processgold.com/en/. Accessed: 2019-23-02. 2

[2] Graphviz - graph visualization software. http://www.graphviz.org/. Accessed: 2018-05-12. 9

[AAA*07] ALVES A., ARKIN A., ASKARY S., BARETTO C., BLOCH B., CUBERA F., FORD M., GOLAND Y., GUIZAR A., KARTHA N., LIU C., KHALAF R., KONIG D., MARIN M., MEHTA V., THATTE S., VAN DER RIJN D., YENDLURI P., YIU A.: Web Services Business Process Execution Language, Apr. 2007. 3

[Aal16] AALST W. M. P. V. D.: *Process Mining: Data Science in Action*. Springer, Apr. 2016. 1, 2

[AEHK10] ALBRECHT B., EFFINGER P., HELD M., KAUFMANN M.: An automatic layout algorithm for BPEL processes. In *Proceedings of the 5th International Symposium on Software Visualization* (New York, NY, USA, 2010), SOFTVIS '10, ACM, pp. 173–182. 1, 3

[AP16] ARCHAMBAULT D., PURCHASE H. C.: Can animation support the visualisation of dynamic graphs? *Information Sciences 330* (Feb. 2016), 495–509. 4

[BB99] BEDERSON B. B., BOLTMAN A.: Does animation help users build mental maps of spatial information? In *Proc. IEEE Symp. Information Visualization (InfoVis '99)* (1999), pp. 28–35. 4

[BBDW17] BECK F., BURCH M., DIEHL S., WEISKOPF D.: A taxonomy and survey of dynamic graph visualization. *Computer Graphics Forum 36*, 1 (Jan. 2017), 133–159. 1, 3

[BP90] BÖHRINGER K.-F., PAULISCH F. N.: Using constraints to achieve stability in automatic graph layout algorithms. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (New York, NY, USA, 1990), CHI '90, ACM, pp. 43–51. 3

[BPF14] BACH B., PIETRIGA E., FEKETE J. D.: GraphDiaries: Animated transitions and temporal navigation for dynamic networks. *IEEE Transactions on Visualization and Computer Graphics 20*, 5 (May 2014), 740–754. 4

[BS15] BERNSTEIN V., SOFFER P.: Identifying and quantifying visual layout features of business process models. In *Enterprise, Business-Process and Information Systems Modeling*, Lecture Notes in Business Information Processing. Springer, Cham, June 2015, pp. 200–213. 3

[BW97] BRANDES U., WAGNER D.: A bayesian paradigm for dynamic graph layout. In *Graph Drawing* (Sept. 1997), Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, pp. 236–247. 3, 4

[CDBTT95] COHEN R., DI BATTISTA G., TAMASSIA R., TOLLIS I.: Dynamic graph drawings: Trees, series-parallel digraphs, and planar ST-digraphs. *SIAM Journal on Computing 24*, 5 (Oct. 1995), 970–1001. 3

[CLRS09] CORMEN T. H., LEISERSON C. E., RIVEST R. L., STEIN C.: *Introduction to Algorithms*, 3rd edition ed. The MIT Press, Cambridge, Mass, July 2009. 6

[CP96] COLEMAN M. K., PARKER D. S.: Aesthetics-based graph layout for human consumption. *Softw. Pract. Exper. 26*, 12 (Dec. 1996), 1415–1438. 3

[CT12] CHINOSI M., TROMBETTA A.: BPMN: An introduction to the standard. *Computer Standards & Interfaces 34*, 1 (Jan. 2012), 124–134. 4

[DDK*02] DIGUGLIELMO G., DUROCHER E., KAPLAN P., SANDER G., VASILIU A.: Graph layout for workflow applications with ILOG JViews. In *Graph Drawing* (Aug. 2002), Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, pp. 362–363. 3, 4

[DG02] DIEHL S., GÖRG C.: Graphs, they are changing. In *Graph Drawing* (Aug. 2002), Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, pp. 23–31. 9

[DGK01] DIEHL S., GÖRG C., KERREN A.: Preserving the mental map using foresighted layout. In *In Proc. Joint Eurographics–IEEE TCVG Symp. Visualization (VisSym'01)* (2001), Springer Verlag, pp. 175–184. 3

[EHK*03] ERTEN C., HARDING P. J., KOBOUROV S. G., WAMPLER K., YEE G.: GraphAEL: Graph animations with evolving layouts. In *Graph Drawing* (Sept. 2003), Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, pp. 98–110. 3

[EMW86] EADES P., MCKAY B., WORMALD N. C.: On an edge crossing problem. In *Proc. 9th Australian Computer Science Conf.* (1986), pp. 327–334. 8

[ESK09] EFFINGER P., SIEBENHALLER M., KAUFMANN M.: An interactive layout tool for BPMN. In *2009 IEEE Conference on Commerce and Enterprise Computing* (July 2009), pp. 399–406. 3, 4

[FE02] FRIEDRICH C., EADES P.: Graph drawing in motion. *Journal of Graph Algorithms and Applications, Volume 6* (2002). 4

[For04] FORSTER M.: A fast and simple heuristic for constrained two-level crossing reduction. In *Graph Drawing* (Sept. 2004), Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, pp. 206–216. 8

[FT08] FRISHMAN Y., TAL A.: Online dynamic graph drawing. *IEEE Transactions on Visualization and Computer Graphics 14*, 4 (July 2008), 727–740. 3, 4

[FWSL12] FENG K. C., WANG C., SHEN H. W., LEE T. Y.: Coherent time-varying graph drawing with multifocus+context interaction. *IEEE Transactions on Visualization and Computer Graphics 18*, 8 (Aug. 2012), 1330–1342. 3

[GBPD04] GÖRG C., BIRKE P., POHL M., DIEHL S.: Dynamic graph drawing of sequences of orthogonal and hierarchical graphs. In *Graph Drawing* (Sept. 2004), Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, pp. 228–238. 3, 11

[GKN15] GANSNER E. R., KOUTSOFIOS E., NORTH S.: *Drawing graphs with dot*. Tech. rep., Graphviz, Jan. 2015. 1, 2, 4, 9

[GKNV93] GANSNER E. R., KOUTSOFIOS E., NORTH S. C., VO K. P.: A technique for drawing directed graphs. *IEEE Transactions on Software Engineering 19*, 3 (Mar. 1993), 214–230. 4, 6, 7, 8, 9, 10

[GNV88] GANSNER E. R., NORTH S. C., VO K. P.: DAG–a program that draws directed graphs. *Software: Practice and Experience 18*, 11 (Nov. 1988), 1047–1062. 9

[GPZ*14] GSCHWIND T., PINGGERA J., ZUGAL S., REIJERS H. A., WEBER B.: A linear time layout algorithm for business process models. *Journal of Visual Languages & Computing 25*, 2 (Apr. 2014), 117–132. 3

[HEHL13] HUANG W., EADES P., HONG S.-H., LIN C.-C.: Improving multiple aesthetics produces better graph drawings. *Journal of Visual Languages & Computing 24*, 4 (Aug. 2013), 262–272. 3

[HM98] HE W., MARRIOTT K.: Constrained Graph Layout. *Constraints 3*, 4 (Oct. 1998), 289–314. 3

[JMM*16] JABRAYILOV A., MALLACH S., MUTZEL P., RÜEGG U., VON HANXLEDEN R.: Compact Layered Drawings of General Directed Graphs. *arXiv:1609.01755 [cs]* (Aug. 2016). arXiv: 1609.01755. 3

[JMS18] JÜNGER M., MUTZEL P., SPISLA C.: A Flow Formulation for Horizontal Coordinate Assignment with Prescribed Width. *arXiv:1806.06617 [cs]* (June 2018). arXiv: 1806.06617. 3

[LLY06] LEE Y.-Y., LIN C.-C., YEN H.-C.: Mental map preserving graph drawing using simulated annealing. In *Proc. 2006 Asia-Pacific Symp. Information Visualisation (APVis'06)* (Darlinghurst, Australia, 2006), pp. 179–188. 3

[MELS95] MISUE K., EADES P., LAI W., SUGIYAMA K.: Layout adjustment and the mental map. *Journal of Visual Languages & Computing 6*, 2 (June 1995), 183–210. 1, 3, 4

[Men17] MENNENS R.: *The implementation of a Sugiyama Layout algorithm*. Tech. rep., Eindhoven University of Technology, 2017. 4

[Men18] MENNENS R.: *Graph layout stability in process mining*. Master's thesis, Eindhoven University of Technology, Eindhoven, The Netherlands, Sept. 2018. 5

[Nor95] NORTH S. C.: Incremental layout in DynaDAG. In *Graph Drawing* (Sept. 1995), Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, pp. 409–418. 3

[PCA02] PURCHASE H. C., CARRINGTON D., ALLDER J.-A.: Empirical evaluation of aesthetics-based graph layout. *Empirical Software Engineering 7*, 3 (Sept. 2002), 233–255. 3

[PCJ97] PURCHASE H. C., COHEN R. F., JAMES M. I.: An Experimental Study of the Basis for Graph Drawing Algorithms. *J. Exp. Algorithmics 2* (Jan. 1997). 3

[PHG06] PURCHASE H. C., HOGGAN E., GÖRG C.: How important is the "mental map"? – an empirical investigation of a dynamic graph layout algorithm. In *Graph Drawing* (Sept. 2006), Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, pp. 184–195. 1

[PKL04] PINAUD B., KUNTZ P., LEHN R.: Dynamic graph drawing with a hybridized genetic algorithm. In *Adaptive Computing in Design and Manufacture VI*. Springer, London, 2004, pp. 365–375. 3

[Pur00] PURCHASE H. C.: Effective information visualisation: a study of graph drawing aesthetics and algorithms. *Interacting with Computers 13*, 2 (Dec. 2000), 147–162. 3

[Pur02] PURCHASE H. C.: Metrics for graph drawing aesthetics. *Journal of Visual Languages & Computing 13*, 5 (Oct. 2002), 501–516. 3

[RBRB06] RINDERLE S. B., BOBRIK R., REICHERT M. U., BAUER T.: Business process visualization – use cases, challenges, solutions. In *Proceedings of the Eighth International Conference on Enterprise Information Systems (ICEIS'06): Information System Analysis and Specification* (May 2006), INSTICC PRESS. 1, 3

[RPD09] REITZ F., POHL M., DIEHL S.: Focused animation of dynamic compound graphs. In *13th Int. Conf. Information Visualisation* (July 2009), pp. 679–684. 3, 4

[San95] SANDER G.: A fast heuristic for hierarchical Manhattan layout. In *Graph Drawing* (Sept. 1995), Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, pp. 447–458. 3

[SI08] SHANMUGASUNDARAM M., IRANI P.: The effect of animated transitions in zooming interfaces. In *Proceedings of the Working Conference on Advanced Visual Interfaces* (New York, NY, USA, 2008), AVI '08, ACM, pp. 396–399. 4

[SIG07] SHANMUGASUNDARAM M., IRANI P., GUTWIN C.: Can smooth view transitions facilitate perceptual constancy in node-link diagrams? In *Proceedings of Graphics Interface 2007* (New York, NY, USA, 2007), GI '07, ACM, pp. 71–78. 4

[SL97] SIMONS D. J., LEVIN D. T.: Change blindness. *Trends in Cognitive Sciences 1*, 7 (Oct. 1997), 261–267. 4

[SP08] SAFFREY P., PURCHASE H.: The "mental map" versus "static aesthetic" compromise in dynamic graphs: A user study. In *Proc. 9th Conf. Australasian User Interface* (Darlinghurst, Australia, 2008), AUIC '08, pp. 85–93. 3

[ST01] SIX J. M., TOLLIS I. G.: Automated visualization of process diagrams. In *Graph Drawing* (Sept. 2001), Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, pp. 45–59. 3

[STT81] SUGIYAMA K., TAGAWA S., TODA M.: Methods for visual understanding of hierarchical system structures. *IEEE Transactions on Systems, Man, and Cybernetics 11*, 2 (1981), 109–125. 2, 3, 4, 6

[VdA09] VAN DER AALST W. M.: Process-aware information systems: lessons to be learned from process mining. In *Transactions on Petri Nets and Other Models of Concurrency II*. Springer, 2009, pp. 1–26. 1

[Wad00] WADDLE V.: Graph layout for displaying data structures. In *Graph Drawing* (Sept. 2000), Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, pp. 241–252. 3

[YLS*04] YANG Y., LAI W., SHEN J., HUANG X., YAN J., SETIAWAN L.: Effective visualisation of workflow enactment. In *Advanced Web Technologies and Applications* (Apr. 2004), Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, pp. 794–803. 3, 4

[ZKS11] ZAMAN L., KALRA A., STUERZLINGER W.: The effect of animation, dual view, difference layers, and relative re-layout in hierarchical diagram differencing. In *Proceedings of Graphics Interface 2011 (GI'11)* (2011), Canadian Human-Computer Communications Society, pp. 183–190. 1, 4