

Comparing Ordering Strategies For Process Discovery Using Synthesis Rules

Tsung-Hao Huang and Wil M. P. van der Aalst

Process and Data Science (PADS), RWTH Aachen University, Aachen, Germany
{tsunghao.huang, wvdaalst}@pads.rwth-aachen.de

Abstract. Process discovery aims to learn process models from observed behaviors, i.e., event logs, in the information systems. The discovered models serve as the starting point for process mining techniques that are used to address performance and compliance problems. Compared to the state-of-the-art Inductive Miner, the algorithm applying synthesis rules from the free-choice net theory discovers process models with more flexible (non-block) structures while ensuring the same desirable soundness and free-choiceness properties. Moreover, recent development in this line of work shows that the discovered models have compatible quality. Following the synthesis rules, the algorithm incrementally modifies an existing process model by adding the activities in the event log one at a time. As the applications of rules are highly dependent on the existing model structure, the model quality and computation time are significantly influenced by the order of adding activities. In this paper, we investigate the effect of different ordering strategies on the discovered models (w.r.t. fitness and precision) and the computation time using real-life event data. The results show that the proposed ordering strategy can improve the quality of the resulting process models while requiring less time compared to the ordering strategy solely based on the frequency of activities.

Keywords: Process discovery · Synthesis rules · Ordering strategy.

1 Introduction

Process mining, a discipline bridging the gap between process science and data science [2], offers techniques and tools to analyze event data, i.e., event logs, generated during the process execution. The analysis generated by process mining techniques provides valuable data-driven insights for the stakeholders.

Process discovery is one of the three main research fields in process mining among conformance checking and process enhancement. Process discovery techniques aim to learn end-to-end process models from the event data. With the discovered models, knowledge workers can apply other process mining techniques to generate further insights for optimization.

While various algorithms have been proposed, only a few ensure desirable properties such as soundness and free-choiceness. On the one hand, the soundness property guarantees that (1) it is always possible to finish the process (2)

a process can be properly completed (3) no inexecutable transitions exist in the model [1]. On the other hand, the free-choice property separates the choice and synchronization constructs of a process model (Petri net). Such property is desirable as it allows easy conversions from the discovered model to widely-used notations such as BPMN [3]. Moreover, free-choice nets are supported by an abundance of analysis techniques developed from the theory [5].

State-of-the-art techniques, such as the Inductive Miner (IM) [9] family, discover process models guaranteed to be sound and free-choice. IM can provide such guarantees by exploiting its internal process representation - the *process tree*. However, such representation can also be a double-edged sword. Due to the representational bias, the discovered models by IM are doomed to be block-structured, i.e., the model must compose of parts that have a single entry and exit [9]. This implies that only a subset of sound free-choice workflow nets can be discovered by IM.

To provide a more flexible process representation while keeping the same guarantees, we proposed a novel discovery algorithm, the so-called Synthesis Miner in [8]. The Synthesis Miner utilizes the synthesis rules from the free-choice net theory [5]. Activities in the event log are gradually added to a model under construction using predefined patterns. Following the rules ensures that the discovered process models are always sound and free-choice. Moreover, it is shown that the discovered models have compatible quality compared to the ones from Inductive Miner. Nevertheless, the possible applications of synthesis rules are highly dependent on the existing model structure. Different orders of adding activities can result in different models. Therefore, an open research question is the influence of the order in which the activities are added to an existing model on the final process model quality. In this paper, we address the research question by comparing the ordering strategies for the Synthesis Miner and taking a deeper look into the impacts of the activity adding order to the model quality and computation time. The experiment using four publicly available real-life event logs shows that advanced ordering strategies can significantly improve the model quality and the computation time.

The remainder of the paper is structured as follows. Related work is presented in Sect.2. We introduce the necessary notations and concepts used throughout the paper in Sect. 3. Then, the proposed ordering strategies are introduced in Sect. 4. The evaluation using publicly available real-life event logs is presented in 5. Finally, Sect. 6 concludes this paper.

2 Related Work

For a general introduction to process mining, we refer to [2]. Additionally, a review and benchmark of the recent development in process discovery can be found in [4]. In this paper, we focus on process discovery techniques that incrementally modify a model under construction to derive the final process.

Incremental process mining allows users to learn a process model from event logs by gradually integrating different traces into an existing model [14]. As the ordering strategy has a significant impact on the model quality, a study [13] is

conducted to investigate the interplay. Nevertheless, it is the trace that is added to the algorithm iteratively rather than the activity. Therefore, it is less relevant to this paper.

Dixit et al. [6] were among the first to use synthesis rules from free-choice net theory [5] to discover process models. Inspired by [6], [8] introduces the Synthesis Miner that automates the discovery by introducing predefined patterns and a search space pruning mechanism. Both [6] and [8] introduce a few ordering strategies for their approaches. However, the choice of ordering is left to the user as an input parameter. The impact of the ordering strategies on the model quality and computation time is not thoroughly investigated. Furthermore, the interplay between the ordering strategies and the search space pruning has not been explained. Last but not least, a comparison between different ordering strategies is needed. In this paper, we aim to address the open research question and provide users with a rule of thumb.

3 Preliminaries

In this section, we introduce the necessary concepts and notations that are used throughout the paper.

For an arbitrary set A , we denote the set of all possible sequences as A^* and the set of all multi-sets over A as $\mathcal{B}(A)$. Given $\sigma_1, \sigma_2 \in A^*$, $\sigma_1 \cdot \sigma_2$ denotes the concatenation of the two sequences. Let A be a set and $X \subseteq A$ be a subset of A . For $\sigma \in A^*$ and $a \in A$, we define $\upharpoonright_{X \in A^* \rightarrow X^*}$ as a projection function recursively with $\langle \rangle \upharpoonright_X = \langle \rangle$, $\langle \langle a \rangle \cdot \sigma \rangle \upharpoonright_X = \langle a \rangle \cdot \sigma \upharpoonright_X$ if $a \in X$ and $\langle \langle a \rangle \cdot \sigma \rangle \upharpoonright_X = \sigma \upharpoonright_X$ if $a \notin X$. For example, $\langle x, y, x \rangle \upharpoonright_{\{x, z\}} = \langle x, x \rangle$. The projection function can also be applied to a multi-set of sequences. For example, $[\langle x, y, x \rangle^4, \langle x, y \rangle^2, \langle y, x, z \rangle^6] \upharpoonright_{\{y, z\}} = [\langle y \rangle^6, \langle y, z \rangle^6]$. We denote \mathcal{U}_A as the universe of activity labels.

Definition 1 (Trace & Log). *A trace $\sigma \in \mathcal{U}_A^*$ is a sequence of activity labels. A log is a multi-set of traces, i.e., $L \in \mathcal{B}(\mathcal{U}_A^*)$.*

Definition 2 (Log Properties [8]). *Let $L \in \mathcal{B}(\mathcal{U}_A^*)$ and $a, b \in \mathcal{U}_A$ be two activity labels. We define the following log properties:*

- $\#(a, L) = \sum_{\sigma \in L} |\{i \in \{1, 2, \dots, |\sigma|\} \mid \sigma(i) = a\}|$ is the times a occurred in L .
- $\#(a, b, L) = \sum_{\sigma \in L} |\{i \in \{1, 2, \dots, |\sigma| - 1\} \mid \sigma(i) = a \wedge \sigma(i + 1) = b\}|$ is the number of direct successions from a to b in L .
- $\text{caus}(a, b, L) = \begin{cases} \frac{\#(a, b, L) - \#(b, a, L)}{\#(a, b, L) + \#(b, a, L) + 1} & \text{if } a \neq b \\ \frac{\#(a, b, L)}{\#(a, b, L) + 1} & \text{if } a = b \end{cases}$ is the strength of causal relation (a, b) .
- $A_c^{\text{pre}}(a, L) = \{a_{\text{pre}} \in \mathcal{U}_A \mid \text{caus}(a_{\text{pre}}, a, L) \geq c\}$ is the set of a 's preceding activities, determined by threshold c .
- $A_c^{\text{fol}}(a, L) = \{a_{\text{fol}} \in \mathcal{U}_A \mid \text{caus}(a, a_{\text{fol}}, L) \geq c\}$ is the set of a 's following activities, determined by threshold c .

Definition 3 (Petri Net). Let $N = (P, T, F, l)$ be a Petri net, where P is the set of places, T is the set of transitions, $P \cap T = \emptyset$. $F \subseteq (P \times T) \cup (T \times P)$ is the set of arcs, and $l \in T \rightarrow \mathcal{U}_A \cup \{\tau\}$ is a labeling function that assigns activity labels to transitions. A transition $t \in T$ is invisible (or silent) if $l(t) = \tau$.

Definition 4 (Path & Elementary Path). A path of a Petri net $N = (P, T, F)$ is a non-empty sequence of nodes $\rho = \langle x_1, x_2, \dots, x_n \rangle$ such that $(x_i, x_{i+1}) \in F$ for $1 \leq i < n$. ρ is an elementary path if $x_i \neq x_j$ for $1 \leq i < j \leq n$. For $X, X' \in P \cup T$, $\text{elemPaths}(X, X', N) \subseteq (P \cup T)^*$ is the set of all elementary paths from some $x \in X$ to some $x' \in X'$.

Definition 5 (Workflow Net (WF-net) [1]). Let $N = (P, T, F, l)$ be a Petri net. $W = (P, T, F, l, i, o, \top, \perp)$ is a WF-net iff (1) it has a dedicated source place $i \in P$: $\bullet i = \emptyset$ and a dedicated sink place $o \in P$: $o \bullet = \emptyset$ (2) $\top \in T$: $\bullet \top = \{i\} \wedge i \bullet = \{\top\}$ and $\perp \in T$: $\perp \bullet = \{o\} \wedge o \bullet = \{\perp\}$ (3) every node x is on some path from i to o , i.e., $\forall x \in P \cup T, (i, x) \in F^* \wedge (x, o) \in F^*$, where F^* is the reflexive transitive closure of F .

Definition 6 (Activity Order). Let $L \in \mathcal{B}(\mathcal{U}_A^*)$ and $A = \bigcup_{\sigma \in L} \{a \in \sigma\}$. $\gamma \in A^*$ is an activity order for L if $\{a \in \gamma\} = A$ and $|\gamma| = |A|$.

Synthesis Miner: Process Discovery Using Synthesis Rules In previous work [8], we introduced the Synthesis Miner that guarantees to discover sound and free-choice workflow nets by applying the synthesis rules defined in [5] with an additional dual abstraction rule [8].

Given a workflow net W , the abstraction rule (ψ_A) allows to add a place p and a transition t between a set of transitions $R \subseteq T$ and a set of places $S \subseteq P$ if they are fully connected, i.e., $(R \times S \subseteq F) \wedge (R \times S \neq \emptyset)$. The linear transition/place rule (ψ_T/ψ_P) allows to add a transition t /place p if it is linearly dependent on the other transitions/places in the corresponding incidence matrix. The dual abstraction rule (ψ_D) can add a transition t and a place p between a set of places S and a set of transitions R if $(S \times R \subseteq F) \wedge (S \times R \neq \emptyset)$. All four rules¹ preserve sound and free-choice properties [5,8]. Fig. 1 shows a few examples of rules applications.

Given a log L , the Synthesis Miner first determines an activity order γ . Then, the iteration is initiated. In iteration i (where $1 \leq i \leq |\gamma|$), activity $\gamma(i)$ is added to an existing net² from the $i - 1$ iteration. The procedure for every iteration is as follows: (1) use heuristics from the projected log $L_i = L \upharpoonright_{\{\gamma(1), \gamma(2), \dots, \gamma(i)\}}$ to find the most likely position for the to-be-added activity $\gamma(i)$ on the existing WF-net (W_i), (2) apply predefined patterns (derived from synthesis rules) to get the set of candidate nets, and (3) select the best net (w.r.t. fitness and precision) from the set of candidates for the next iteration.

¹ For the formal definitions of the rules, we refer to [5,8].

² The existing net in the first iteration is initiated by the initial net, as shown in the example for the abstraction rule in Fig.1.

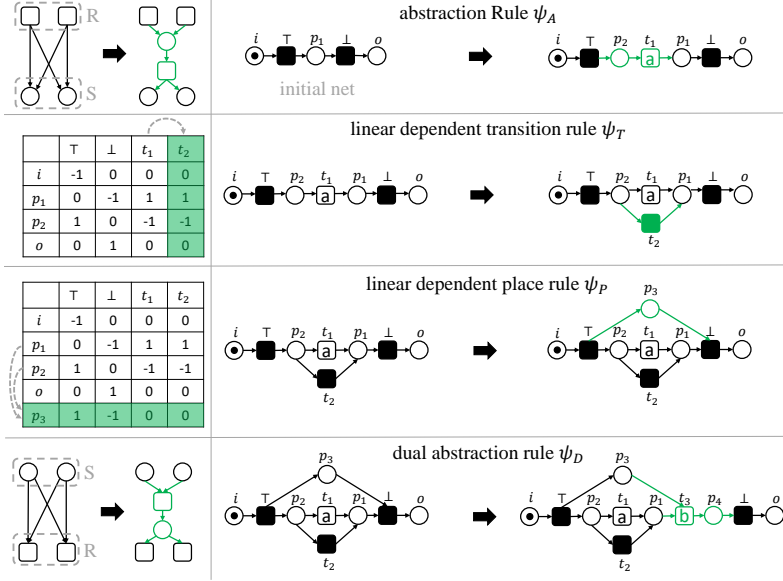


Fig. 1: Some examples of the synthesis rules applications. ψ_A allows to add p_2 and t_1 by $R = \{\top\}$ and $S = \{p_1\}$. t_2 is added by ψ_T as it is linearly dependent on t_1 . p_3 is added by ψ_P as it is a linear combination of p_1 and p_2 . ψ_D allows to add t_3 and p_4 with $S = \{p_1, p_3\}$ and $R = \{\perp\}$.

As step (1) is directly affected by the ordering strategy, we formally define³ how the search space is limited to only a subset of the nodes on a workflow net using log heuristics.

Definition 7 (Reduced Search Space). Let $a \in \mathcal{U}_A^*$ be an activity, $L \in \mathcal{B}(\mathcal{U}_A^*)$ be a log, $W = (P, T, F, l, i, o, \top, \perp)$ be a WF-net, and $0 \leq c \leq 1$. T^{pre} is the set of transitions labeled by the preceding activities of a in log L . $T^{pre} = \{t \in T \mid l(t) \in A_c^{pre}(a, L)\}$ if $A_c^{pre}(a, L) \neq \emptyset$, otherwise $T^{pre} = \{\top\}$. T^{fol} is the set of transitions labeled by the following activities of a in log L . $T^{fol} = \{t \in T \mid l(t) \in A_c^{fol}(a, L)\}$ if $A_c^{fol}(a, L) \neq \emptyset$, otherwise $T^{fol} = \{\perp\}$. The reduced search space is $reduce(a, L, W, c) = \{x \in \rho \mid \rho \in elemPath(T^{pre}, T^{fol}, W)\}$.

The function *reduce* first finds the preceding and following activities and the corresponding sets of labeled transitions for the to-be-added activity $\gamma(i)$. Then, it returns the set of nodes, denoted as V_i , that are on the path between the preceding and following transitions. V_i is used to confine the application of synthesis rules. To be more precise, the set of transitions R and the set of places S used as the preconditions for applying rules ψ_A and ψ_D need to be a subset of V_i , i.e., $S \subseteq V \wedge R \subseteq V$. As for rule ψ_T/ψ_P , the new transition/place (t'/p') cannot have arcs connected to any node other than V_i . This step helps us to limit the search space to the most likely nodes on a workflow net to add activity $\gamma(i)$. Fig. 2 shows an example for reducing the search space.

³ As the formal definitions of steps (2) and (3) are out of scope, we refer to [8].

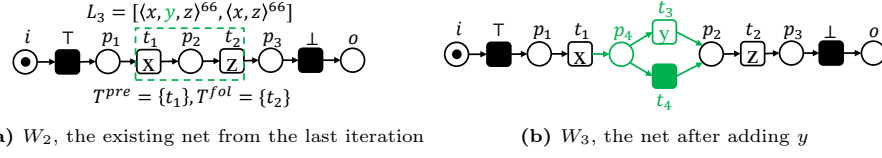


Fig. 2: An example showing how the search space is reduced. Consider the log $L_3 = [\langle x, y, z \rangle^{66}, \langle x, z \rangle^{66}]$. y is the activity which we want to add to the net W_2 . Using $c = 0.9$, we get $T^{pre} = \{t_1\}$ and $T^{fol} = \{t_2\}$. Therefore, the function *reduce* would return the set of nodes between t_1 and t_2 , which means $V_3 = \{t_1, p_2, t_2\}$ as highlighted by the green dashed line in (a). The application of synthesis rules would then only consider these three nodes. Finally, the best net is selected as W_3 from the candidates and is visualized in (b).

4 Ordering Strategies

In this section, we introduce different ordering strategies. To illustrate the ordering strategy, consider the following log $L_s = [\langle b, c, d, e, f, g \rangle, \langle b, e, c, d, f, g \rangle, \langle b, e, c, f, g, d \rangle, \langle b, e, c, f, d, g \rangle, \langle b, c, e, d, f, g \rangle, \langle b, c, e, f, g, d \rangle, \langle b, c, e, f, d, g \rangle, \langle e, b, c, f, g, d \rangle, \langle e, b, c, f, d, g \rangle]$.

The corresponding directly follows graph (DFG) is shown in Fig. 3.

The first ordering strategy is frequency-based and it is relatively straightforward. The activities are simply ordered by their frequency in the log.

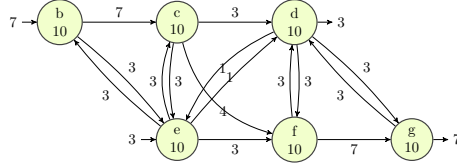


Fig. 3: The DFG for log L_s .

Definition 8 (Frequency-Based Ordering). Let $L \in \mathcal{B}(\mathcal{U}_A^*)$. Frequency-based ordering function is $order_{freq}(L) = \gamma$ such that γ is an activity order and $\forall_{1 \leq i < j \leq |\gamma|} \#(\gamma(i), L) \geq \#(\gamma(j), L)$.

If activities have the same frequency, we order them alphabetically. Using the example log L_s for illustration, the order would be $order_{freq}(L_s) = \langle b, c, d, e, f, g \rangle$.

The other ordering strategies are more involved as they consider not only the frequency of activities but also the connections between them. Before introducing the other ordering strategies, we first define a helper function that ranks the directly-follow activities based on the strength of connections.

Definition 9 (Directly-Follow Activities Sorting). Let $L \in \mathcal{B}(\mathcal{U}_A^*)$ and $a \in \mathcal{U}_A$. $A = \{b \in \mathcal{U}_A \mid \#(a, b, L) > 0\}$ is the set of activities directly-follow a in L at least once and $\sigma \in A^*$. Directly-follow activities sorting is $sortDFA(a, L) = \sigma$ such that $\{b \in \sigma\} = A$ and $|\sigma| = |A|$ and $\forall_{1 \leq i < j \leq |\sigma|} \#(a, \sigma(i), L) \geq \#(a, \sigma(j), L)$.

For example, $sortDFA(b, L_s) = \langle c, e \rangle$. This is because activities c and e have incoming arcs from b and the strength $\#(b, c, L_s) \geq \#(b, e, L_s)$. With the function for sorting directly-follow activities defined, we are now ready to define the Breadth-First-Search-Based ordering strategy in Algo. 1.

Algorithm 1: Breadth-First-Search-Based Ordering, $order_{BFS}$

```

Input   : A log  $L \in \mathcal{B}(\mathcal{U}_A^*)$ 
Output : An activity order  $\gamma$  for  $L$ 
 $A \leftarrow \bigcup_{\sigma \in L} \{a \in \sigma\}$ ; // the set of activities in  $L$ 
 $A^s \leftarrow \{\sigma(1) \mid \sigma \in L \wedge \sigma \neq \langle \rangle\}$ ; // the set of start activities in  $L$ 
 $\sigma \leftarrow order_{freq}(L) \upharpoonright_{A^s}$ ; // the sequence of start activities ordered by frequency
 $i \leftarrow 1$ ;
while  $|\sigma| \neq |A|$  :
   $A' \leftarrow A \setminus \{a \in \sigma\}$ ; // the set of activities that are not in  $\sigma$ 
   $\sigma' \leftarrow sortDFA(\sigma(i), L) \upharpoonright_{A'}$ ; // sort  $\sigma(i)$ 's following activities & project on  $A'$ 
   $\sigma \leftarrow \sigma \cdot \sigma'$ ; // update  $\sigma$ 
   $i \leftarrow i + 1$ ;
 $\gamma \leftarrow \sigma$ ;
return  $\gamma$ ;

```

BFS-based ordering strategy starts by building a sequence of start activities in a log and iteratively append the sequence of directly-follow activities using the function in Def. 9. Applying the function to the example log L_s , we get $order_{BFS}(L_s) = \langle b, e \rangle \cdot \langle c \rangle \cdot \langle f, d \rangle \cdot \langle \rangle \cdot \langle g \rangle = \langle b, e, c, f, d, g \rangle$. σ is initiated with $\langle b, e \rangle$. Then, in iteration i , σ is appended by the sequence of $\sigma(i)$'s directly-follow activities sorted by $sortDFA(\sigma(i), L_s)$ with the set of activities already in σ filtered out. The loop continues until σ includes every activity in the log. As its name suggests, the ordering prioritizes the exploration of the directly-follow activities.

Next, we introduce another ordering strategy in Algo. 2 that is Depth-First-Search-based. While also considering the connection between the activities as BFS-based ordering strategy, DFS-based ordering prioritizes depth over breadth. That is, the directly-follow activities are not explored thoroughly until activities with higher depth have been explored. Applying DFS-based ordering to log L_s , we get $order_{DFS}(L_s) = \langle b, c, f, g, d, e \rangle$.

Note that although we define the BFS- and DFS-based ordering strategies to start from the start activities, one can also initiate the exploration from another direction, i.e., from the end activities and subsequently explore the directly-precede activities for ordering. Using L_s as an example, if starting from the set of end activities, we would get $\langle g, f, c, b, e, d \rangle$ with DFS-based ordering on log L_s and $\langle g, d, f, c, e, b \rangle$ with BFS-based ordering.

To explain how the progression of the process discovery influenced by the different ordering strategies, Fig. 4 shows all the intermediate nets when applying Synthesis Miner to log L_s using the three different ordering strategies. DFS-based ordering tends to build the process from start to end at the beginning before adding the activities in the parallel/choice branches. On the contrary, BFS-based ordering prioritizes the construction of local control flows. For example, the difference is observable from iteration 1 to 2. While all the ordering strategies produce the same net in iteration 1, BFS-based ordering suggests to add the concurrent activity e for b in iteration 2 and DFS-based ordering adds

Algorithm 2: Depth-First-Search-Based Ordering $order_{DFS}$

Input : A log $L \in \mathcal{B}(\mathcal{U}_A^*)$
Output : An activity order γ for L

$A \leftarrow \bigcup_{\sigma \in L} \{a \in \sigma\}$; // the set of activities in L
 $A^s \leftarrow \{\sigma(1) \mid \sigma \in L \wedge |\sigma| \neq 0\}$; // the set of start activities in L
 $\sigma^s \leftarrow order_{freq}(L) \upharpoonright_{A^s}$; // the sequence of start activities ordered by frequency
 $\sigma \leftarrow \langle \sigma^s(1) \rangle$; // initiate the sequence with the most frequent start activity
 $\sigma^s \leftarrow \sigma^s \upharpoonright_{\{A^s \setminus \{\sigma^s(1)\}\}}$; // update σ^s to be the stack

while $|\sigma| \neq |A|$:

$A' \leftarrow A \setminus \{a \in \sigma\}$; // set of activities that are not in σ
 $\sigma^f \leftarrow sortDFA(\sigma(\sigma), L) \upharpoonright_{A'}$; // sort $\sigma(\sigma)$'s following activities

if $|\sigma^f| = 0$:

$\sigma \leftarrow \sigma \cdot \langle \sigma^s(1) \rangle$; // append the 1st element from the stack σ^s to σ

else :

$\sigma \leftarrow \sigma \cdot \langle \sigma^f(1) \rangle$; // append the 1st element from σ^f to σ
 $\sigma^s \leftarrow (\sigma^f \upharpoonright_{A \setminus \{a \in \sigma \vee a \in \sigma^s\}}) \cdot (\sigma^s \upharpoonright_{A \setminus \{a \in \sigma\}})$; // update the stack σ^s

$\gamma \leftarrow \sigma$;
return γ ;

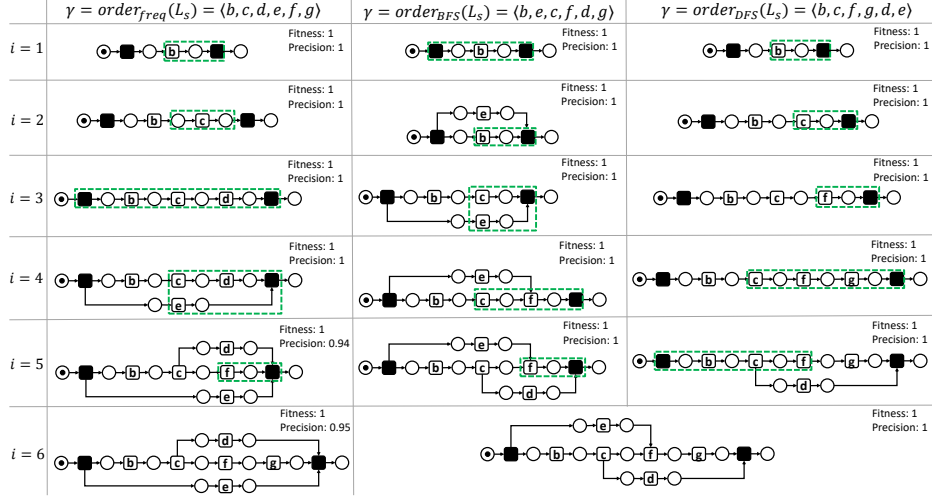


Fig. 4: A comparison of different ordering strategies for log L_s . Each column represents an ordering strategy and each row corresponds to the intermediate workflow net in iteration i after adding $\gamma(i)$. The green dashed lines highlight the nodes representing the reduced search space. The metrics fitness and precision are measured using the corresponding projected log $L_i = L \upharpoonright_{\{\gamma(1), \gamma(2), \dots, \gamma(i)\}}$. Note that the final model discovered by the BFS- and DFS-based ordering strategies are the same in this example.

the directly-follow activity c of b first. The frequency ordering doesn't seem to have clear patterns for the discovery.

We expect that the choice of ordering can significantly influence the computation time of discovery. The main difference stems from the time required to check

the feasibility of the linear dependency rules. As the WF-net grows, it becomes more expensive (w.r.t. time) to check if a candidate place/transition is linear dependent. Thus, it is preferable to limit the search space as small as possible, especially in the later iterations. Recall that the reduced search space (Def. 7) is a set of nodes confining the application of synthesis rules. The green dashed lines in Fig. 4 highlight the reduced search space V_i in iteration i . As shown in Fig. 4, generally, BFS-based ordering can keep the search space smaller than the other strategies because it prioritizes the connected activities. In contrast, the search space of DFS-based ordering is more likely to be large in the later iterations. As the parallel/alternative activities are added later, the preceding and following activities of the to-be-added activity $\gamma(i)$ is highly likely to be spread across the existing net. Together with the effect of search space reduction, it results in a relatively large search space, which indicates more nodes to be considered. Examples can be seen in iterations 4 and 5 for the DFS-based ordering in Fig. 4.

Although it is assumed that BFS-based ordering would have relatively lower computation time, search space reduction might introduce trade-offs between the optimal solution and time. In the following section, we aim to investigate the impact of the ordering strategy on both model quality and the time to discover the process model in the experiment.

5 Evaluation

In this section, we present the experiment used to evaluate the ordering strategies including the setup and a discussion of the result⁴.

5.1 Experimental Setup

For the experiment, we use four publicly available real-life event logs [7,10,11,12]. The logs are filtered to focus on the mainstream behaviors (at least 95% of the traces) where the most frequent trace variants are used. For the BPI2017 log [7], we split it into three logs using the activity prefix (A, W, O). This results in six logs in total.

For every event log, we apply different ordering strategies for the Synthesis Miner [8] with default values for the other parameters. For the BFS- and DFS-based ordering strategies, we apply the ordering from both directions (start and end activities). Therefore, we evaluate five ordering strategies. To measure the effect of ordering strategies on search space pruning, we keep track of the ratio of reduced search space. This is evaluated by $\frac{|V_i|}{|P_i \cup T_i| - 2}$, where V_i is the set of reduced nodes, P_i and T_i are the set of places and transitions in the existing WF-net W_i . The -2 in the denominator is there to exclude the two places (source and sink) that can never be connected by new nodes by Def. 5. Using Fig. 4 as an example, the value of $\frac{|V_3|}{|P_3 \cup T_3| - 2}$ for the frequency ordering strategy would be $\frac{9}{11-2} = 1$ in iteration 3. This indicates that all the possible nodes are considered for the application of synthesis rules to add the next activity. Furthermore, we evaluate the final model in terms of fitness, precision, and F1 score (the harmonic mean of fitness and precision).

⁴ <https://github.com/tsunghao-huang/synthesisRulesMiner>

5.2 Results and Discussion

Search Space Reduction and Computation Time Fig. 5 shows the result of the comparison among the five ordering strategies regarding their effects on the search space reduction. The value in the y-axis $\frac{|V_i|}{|P_i \cup T_i| - 2}$ is the average across six event logs. As indicated, the metric keeps track of the reduced search space ratio for adding the next activity, which indicates the number of possible synthesis rule applications. In general, we can observe from the figure that the

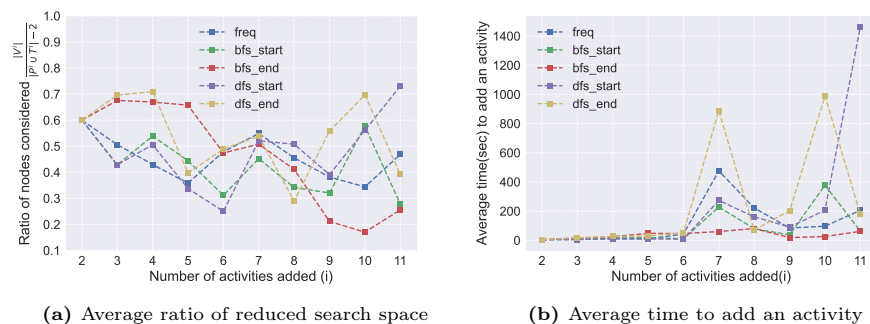


Fig. 5: Comparisons of ordering strategies on the effects of search space reduction as well as the computation time for each step. Note that it is preferable to have a lower value for $\frac{|V_i|}{|P_i \cup T_i| - 2}$.

ordering strategies behaved as expected. As shown in Fig. 5a, in the later stage of the discovery ($i \geq 8$), the BFS-ordering strategies (bfs_start, bfs_end) keep the ratio of reduced search space at a low level while the value for frequency and DFS-based ordering strategies show that they are more likely to include a large portion of the nodes in the search space.

Fig. 5b shows the average time to add an activity to the existing WF-net for each step of six logs. Comparing the two figures, one can see the effect of search space reduction on the computation time. As shown in Fig. 5b, the bfs_end strategy keeps the average computation time for each step at a fairly low level. This is also the case for the bfs_start strategy despite the two peaks when adding the 7th and 10th activity. The two peaks in the 7th and 10th steps are especially severe for the dfs_end strategy. Both took more than 10 minutes to add a single activity to the existing model. Also, the longest duration to add an activity also happens in the 11th step of the dfs_start strategy.

In short, due to its interplay with the search space reduction, the BFS-based ordering strategies have significant advantage in terms of computation time.

Model Quality Table 1⁵ shows the result of the model quality using the five different ordering strategies. As expected, we observe that the BFS-based ordering strategies have the lowest computation time in all six event logs. This

⁵ To provide a reference to the state of the art, we also present the results from IMf (marked by gray color). The best model generated by IMf (w.r.t. F1 score) is selected from a set of nets using five different values ([0.1, 0.2, 0.3, 0.4, 0.5]) for the filter.

Table 1: Quality of the models discovered by different ordering strategies.

Log	Ordering Strategy & IMf	Fitness	Precision	F1	time(sec)
BPI2017A	frequency	0.971	0.947	0.958	685
	BFS_start	0.973	1.000	0.986	893
	BFS_end	0.990	0.935	0.961	334
	DFS_start	0.963	0.868	0.913	1850
	DFS_end	0.999	0.986	0.993	1248
	IMf(0.2)	0.999	0.936	0.967	10
BPI2017O	frequency	0.993	0.962	0.978	537
	BFS_start	0.985	0.963	0.974	165
	BFS_end	0.989	1.000	0.995	231
	DFS_start	0.996	1.000	0.998	498
	DFS_end	0.993	0.962	0.978	360
	IMf(0.2)	0.997	0.907	0.950	7
BPI2017W	frequency	0.993	0.726	0.838	3617
	BFS_start	0.974	0.864	0.914	1626
	BFS_end	0.993	0.888	0.936	579
	DFS_start	0.974	0.864	0.914	1732
	DFS_end	0.993	0.901	0.944	5397
	IMf(0.2)	0.923	0.897	0.910	14
helpdesk	frequency	0.974	0.984	0.978	51
	BFS_start	0.974	0.984	0.978	52
	BFS_end	0.983	0.976	0.979	43
	DFS_start	0.974	0.984	0.978	49
	DFS_end	0.989	0.963	0.976	64
	IMf(0.2)	0.967	0.950	0.958	1
hospital billing	frequency	0.945	0.810	0.879	509
	BFS_start	0.931	0.922	0.936	314
	BFS_end	0.988	0.935	0.961	383
	DFS_start	0.931	0.970	0.961	2154
	DFS_end	0.943	0.883	0.920	2359
	IMf(0.2)	0.982	0.906	0.943	45
traffic	frequency	0.967	0.930	0.945	274
	BFS_start	0.967	0.930	0.945	202
	BFS_end	0.972	0.720	0.825	388
	DFS_start	0.991	0.933	0.960	366
	DFS_end	0.942	0.858	0.903	443
	IMf(0.4)	0.904	0.720	0.801	28

corresponds to the findings in the previous section. Moreover, despite the search space being considerably reduced, the models discovered using BFS-ordering strategies have the highest F1 score in two out of the six logs.

As for the DFS-based ordering strategies, they have an apparent disadvantage for computation time but get the highest F1 score in the other four event logs. The result matches our assumption as search space reduction introduces a trade-off between the optimal solution and time. Lastly, the frequency ordering strategy has no significant advantage in model quality and computation time. The results show that the ordering strategies that take the connections between activities into consideration can improve the Synthesis Miner than the frequency-based ordering strategy.

6 Conclusion

In this paper, we introduced five ordering strategies for the process discovery algorithm using synthesis rules [8]. We investigated the impact of ordering strategies on model quality and computation time. The results show that compared to the ordering strategy solely based on the frequency of activities, the proposed ordering strategies considered the connection between activities (Breadth-First-

Search-based and Depth-First-Search-based) have superior performance w.r.t. time and model quality respectively. It is shown in the result that the introduced BFS-based ordering strategies can speed up the computation. Nevertheless, the overall discovery time of the Synthesis Miner is still not comparable to the state of the art despite being able to discover models with better quality. Therefore, for future work, we plan to speed up the Synthesis Miner by further exploiting the log heuristics and investigating more sophisticated ordering strategies. Another direction for improvement is the ability to cope with infrequent behaviors as we use the most frequent trace variants to capture the mainstream process. It would be valuable to introduce a filtering mechanism to the Synthesis Miner so that it can directly work on the original log without depending on pre-filtering the log.

Acknowledgements. We thank the Alexander von Humboldt (AvH) Stiftung for supporting our research.

References

1. van der Aalst, W.M.P.: The application of Petri nets to workflow management. *J. Circuits Syst. Comput.* **8**(1), 21–66 (1998)
2. van der Aalst, W.M.P.: *Process Mining - Data Science in Action*, Second Edition. Springer (2016)
3. van der Aalst, W.M.P.: Using free-choice nets for process mining and business process management. In: *FedCSIS 2021*. vol. 25, pp. 9–15 (2021)
4. Augusto, A., Conforti, R., Dumas, M., Rosa, M.L., Maggi, F.M., Marrella, A., Mecella, M., Soo, A.: Automated discovery of process models from event logs: Review and benchmark. *IEEE Trans. Knowl. Data Eng.* **31**(4), 686–705 (2019)
5. Desel, J., Esparza, J.: *Free Choice Petri Nets*. No. 40, Cambridge university press (1995)
6. Dixit, P.M., Buijs, J.C.A.M., van der Aalst, W.M.P.: Prodigy : Human-in-the-loop process discovery. In: *RCIS 2018*. pp. 1–12. IEEE (2018)
7. van Dongen, B.: *BPI Challenge 2017* (2017). <https://doi.org/10.4121/uuid:5f3067df-f10b-45da-b98b-86ae4c7a310b>
8. Huang, T., van der Aalst, W.M.P.: Discovering sound free-choice workflow nets with non-block structures. In: *EDOC 2022*. vol. 13585, pp. 200–216. Springer (2022). https://doi.org/10.1007/978-3-031-17604-3_12
9. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Scalable process discovery and conformance checking. *Softw. Syst. Model.* **17**(2), 599–631 (2018)
10. de Leoni, M.M., Mannhardt, F.: *Road Traffic Fine Management Process* (2015). <https://doi.org/10.4121/uuid:270fd440-1057-4fb9-89a9-b699b47990f5>
11. Mannhardt, F.: *Hospital Billing - Event Log* (2017). <https://doi.org/10.4121/uuid:76c46b83-c930-4798-a1c9-4be94dfb741>
12. Polato, M.: *Dataset belonging to the help desk log of an Italian Company* (2017). <https://doi.org/10.4121/uuid:0c60edf1-6f83-4e75-9367-4c63b3e9d5bb>
13. Schuster, D., Domnitsch, E., van Zelst, S.J., van der Aalst, W.M.P.: A generic trace ordering framework for incremental process discovery. In: *IDA 2022*. vol. 13205, pp. 264–277. Springer (2022)
14. Schuster, D., van Zelst, S.J., van der Aalst, W.M.P.: Incremental discovery of hierarchical process models. In: *RCIS 2020*. vol. 385, pp. 417–433. Springer (2020)