Recognizing Relationships: Detecting the 4C Spectrum in $O(P^2 + T^2)$ for Acyclic Sound Process Models

Thomas M. Prinz[1], Torsten Welsch[2], and N. Long Ha[3]

[1] Course Evaluation Service, Friedrich Schiller University Jena, Jena, Germany
Thomas.Prinz@uni-jena.de
[2] Department Information Technologies, HTBLA Grieskirchen, Grieskirchen, Austria
t.welsch@htl-grieskirchen.at
[3] Faculty of Economic Information Systems, University of Economics, Hue University, Vietnam, hnlong@hueuni.edu.vn

Abstract. The identification of business process models within large model collections poses a significant challenge. Process querying offers a solution by selecting models that meet specific characteristics, utilizing queries based on behavioral relations. These relations, which include conflict, co-occurrence, causality, and concurrency (collectively known as the 4C Spectrum), describe the potential interactions between tasks within process models during execution. However, existing approaches to compute these behavioral relations are inefficient for models with numerous execution traces, often requiring extensive time. This paper introduces a set of algorithms, termed "Behavioral Relation Computations" (in short BeRelCo), capable of identifying all 4C Spectrum behavioral relations within acyclic sound free-choice workflow nets with quadratic time complexity $O(P^2 + T^2)$. Our experiments demonstrate significant benefits, particularly for process models characterized by many execution traces.

Keywords: Business process models · Behavioral relations · Acyclic · Soundness

1 Introduction

Business process management (BPM) is an interdisciplinary domain that integrates business and computer science principles, focusing on the analysis and improvement of *business process models*. These models act as blueprints, describing task sequences, dependencies, and decision points to achieve business objectives [6]. Established modeling languages, such as the *Business Process Model and Notation* (BPMN), enable organizations to articulate their processes comprehensively. Typically stored in extensive repositories, identifying process models presents a significant challenge when they are to fulfill certain characteristics. For instance, a business analyst searching for a model that concurrently executes payments and deliveries, subsequent to an order, would face the task of manually reviewing each model in the absence of IT support.

Process queries enable businesses to systematically search through their process models in repositories, identifying models that meet specific criteria [14]. The core of process queries lies in the exploration of *behavioral relations* [15]. These relations reveal how tasks within a process model interact, indicating whether tasks are mutually

exclusive, co-occur, cause one another, or can be executed concurrently. Polyvyanyy et al. [15] introduced a comprehensive set of these behavioral relations, known as the *4C Spectrum*. This spectrum uncovers the fundamental relations of *conflict*, *co-occurrence*, *causality*, and *concurrency*, showcasing their various manifestations across different execution traces of process models. The *4C Spectrum* is aligned with other established behavioral relation frameworks in literature, such as the *(causal) behavioral profile* [21, 22], enhancing its validity and application in the field of BPM.

The behavioral relations within the *4C Spectrum* are binary, indicating that the size of each relation scales quadratically with the number of nodes in a process model. Current detection techniques for these relations, however, tend to require exponential time in the worst case. Although it seems acceptable to derive the behavioral relations for a single process model in seconds, indexing and querying process models from huge repositories is difficult to accomplish with such computationally expensive algorithms [10], especially from an ecological perspective. Polyvyanyy et al. [15] laid the computational groundwork for some of these relations, leveraging concepts such as *reachability* and the *covering problem*. Similarly, Wolf [23] linked most behavioral relations back to the reachability problem. Yet, the general reachability problem for Petri nets falls within the `NONELEMENTARY` complexity class [3], posing significant computational challenges. Ha and Prinz [10] explored these relations for acyclic sound workflow graphs, utilizing a *Single-Entry Single-Exit* (SESE) decomposition into fragments [19] (similarly to Weidlich et al. [22]). This approach uses transitive rules but struggles with unstructured process model fragments, known as "rigids" [19], where state-space exploration — a process with exponential time complexity — becomes necessary. *The current gap is the absence of an algorithm capable of efficiently computing all behavioral relations for unstructured fragments in low polynomial (i. e., quadratic to bi-quadratic) time.* Despite these challenges, understanding behavioral relations is also crucial for analyzing process similarity [11] and checking compliance with business rules [13].

This paper introduces a set of algorithms, termed *Behavioral Relation Computations* (in short *BeRelCo*), which are designed for process models that can be represented as *acyclic sound free-choice workflow nets*. Soundness is a minimal quality correctness criterion [5], whereas free-choiceness increases the alignment of workflow nets to industrial process languages [8]. *BeRelCo* is capable of computing all behavioral relations within the *4C Spectrum* with a *quadratic time complexity*, $O(N^2)$, where N represents the number of nodes. Experimental results from two datasets demonstrate the computational advantages of *BeRelCo*, particularly when numerous different execution traces are possible. For both datasets, all pairwise relations were computed in less than 1 second, with some instances experiencing a speed-up factor exceeding 1000. The approach is also effective for models exhibiting inclusive behavior. While the current focus is on acyclic process models, we are convinced that extending the methodology to cyclic models through *loop decomposition* [16] — a method that converts a cyclic model into a set of acyclic models with equivalent behavior — is feasible.

This paper is organized as follows: Section 2 introduces basic concepts necessary for understanding the rest of the work. This is followed by a description of the behavioral relations of the *4C Spectrum* in acyclic nets in Sect. 3. We then derive algorithms for these relations, showcasing our methodological contributions in Sect. 4. Section 5

briefly discusses how these algorithms can be extended to inclusive behavior. Related work is investigated in Sect. 6. In Sect. 7, an evaluation of the algorithms demonstrates their effectiveness and computational benefits. The paper concludes in Sect. 8 with a reflection on the implications of our work.

2 Preliminaries

This paper builds upon well-established definitions of Petri and workflow nets.

Definition 1 (Petri net). A *(Petri) net* is a triple (P,T,F) with P and T are finite, disjoint sets of *places* and *transitions* and $F \subseteq (P \times T) \cup (T \times P)$ is the *flow* relation. ⌟

The union $P \cup T$ of a net $N = (P,T,F)$ can be interpreted as *nodes* and F as *edges* between those nodes. For $x \in P \cup T$, $\bullet x = \{p \mid (p,x) \in F\}$ is the *preset* of x (all directly preceding nodes) and $x\bullet = \{s \mid (x,s) \in F\}$ is the *postset* of x (all directly succeeding nodes). Each node in $\bullet x$ is an *input* of x and each node in $x\bullet$ is an *output* of x. The preset and postset of a set of nodes $X \subseteq P \cup T$ is defined as $\bullet X = \bigcup_{x \in X} \bullet x$ and $X\bullet = \bigcup_{x \in X} x\bullet$, respectively. A *path* $(n_1,\ldots,n_m)$ is a sequence of nodes $n_1,\ldots,n_m \in P \cup T$ with $m \geq 1$ and $\forall i \in \{1,\ldots,m-1\}$: $n_i \in \bullet n_{i+1}$. Note that places and transitions alternate on paths. If all nodes of a path are pairwise different, the path is *acyclic*; otherwise, it is *cyclic*. $Paths_N(x,y)$ denotes the set of all paths between nodes x and y in N, where $x,y \in P \cup T$. N is *acyclic* if all its paths are acyclic. Each net in this paper is restricted to be *simple free-choice*: $\forall p \in P, |p\bullet| > 1$: $\bullet(p\bullet) = \{p\}$ [8]. In the nets shown here, circles represent places, rectangles transitions and directed edges represent flows (see Fig. 1 as an example).

Definition 2 (Workflow and AFW-net). A *workflow net* $WN = (P,T,F,s,f)$ is a net (P,T,F) with $s,f \in P$, $\bullet s = \emptyset$, and $f\bullet = \emptyset$. s is the *source* and f is the *sink* of WN. All nodes are on a path from s to f. If WN is acyclic and free-choice, we call it *AFW-net*. ⌟

Figure 1 visualizes a workflow net. *Markings* of workflow nets describe states, which specify the number of *tokens* at each place:

Definition 3 (Marking). A *marking* of a workflow net $WN = (P,T,F,s,f)$ is a total mapping M: $P \mapsto \mathbb{N}_0$ that assigns a natural number of *tokens* to each place of P. $M(p) = 1$ means that place $p \in P$ carries one token in marking M. ⌟

The *initial* marking M_s is a marking where only the source s has a token. The *terminal* marking M_f is a marking where only the sink f has a token. Transitions whose input places all have tokens are *enabled* in a marking and can be fired, leading to the workflow net's semantics:

Definition 4 (Semantics). Let $WN = (P,T,F,s,f)$ be a workflow net with a marking M. A transition $t \in T$ is *enabled* in M iff every place $p \in \bullet t$ contains at least one token in M, $\forall p \in \bullet t$: $M(p) \geq 1$. If t is enabled in M, then t can *occur* ("fire"), which leads to a *step* from M to M' via t, denoted as $M \xrightarrow{t} M'$, with

$$M'(p) = M(p) \; - \; \begin{cases} 1, & p \in \bullet t \\ 0, & \text{else} \end{cases} \; + \; \begin{cases} 1, & p \in t\bullet \\ 0, & \text{else.} \end{cases}$$ ⌟

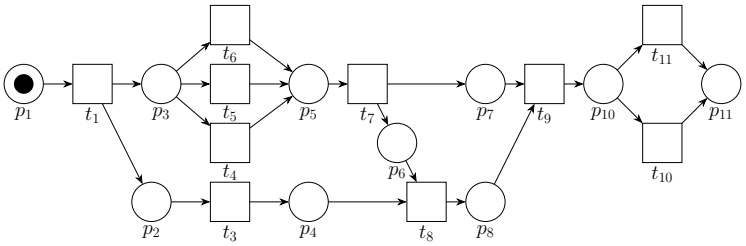

Fig. 1. A graphical example of a Petri net.

I. e., in a step via t, t "consumes" one token from all its input places and "produces" one token for all its output places. Stepwise firings of transitions lead to chains of fired transitions, which describe the behavior of a net as occurrence sequences:

Definition 5 (Occurrence Sequences and Runs). Let $WN = (P,T,F,s,f)$ be a workflow net with a marking M_0. A sequence of transitions $\sigma = \langle t_1,\ldots,t_n\rangle$, $n \in \mathbb{N}_0$, $t_1,\ldots,t_n \in T$, is an *occurrence sequence* of M_0 iff there is a sequence of markings $M_0, M_1, \ldots, M_n$ such that $M_{i-1} \xrightarrow{t_i} M_i$ holds for each $i \in \{1,\ldots,n\}$. It can be said that σ *leads* from M_0 to M_n. A place $p \in P$ occurs in σ, depicted as $p \in \sigma$, iff the steps $M_0 \xrightarrow{t_1} M_1 \xrightarrow{t_2} \ldots \xrightarrow{t_n} M_n$ contain a marking M_i, $i \in \{1,\ldots,n\}$, with $M_i(p) \geq 1$. σ is a *run* iff σ leads from the initial marking M_s to the terminal marking M_f of WN. ⌟

A marking M' is *reachable* from a marking M (denoted $M \rightarrow^* M'$) iff there is an occurrence sequence σ of M that leads to M'.

Definition 6 (Soundness). A workflow net $WN = (P,T,F,s,f)$ with its initial marking $M_s = \{s\}$ and its terminal marking $M_f = \{f\}$ is *sound* iff

(1) $\forall M,\ M_s \rightarrow^* M\colon M \rightarrow^* M_f$,
(2) $\forall M,\ M_s \rightarrow^* M\colon (M(f) \geq 1 \implies M = M_f)$, and
(3) there is no *dead* transition in WN: $\quad \forall t \in T\ \exists M,M'\colon M_s \rightarrow^* M \xrightarrow{t} M'$. [1] ⌟

This paper focuses on sound AFW-nets.

Definition 7 (Run Net). A net $\pi = (P_R, T_R, F_R)$ is a *run net* of a sound AFW-net $N = (P,T,F,s,f)$ and a run R iff

$$P_R = \{p \in P\colon p \in R\} \wedge T_R = \{t \in T\colon t \in R\} \wedge F_R = \{(x,y) \in F\colon x \in (P_R \cup T_R) \wedge y \in (P_R \cup T_R)\}$$

$n \in R$ *occurs* in π, depicted as $n \in \pi$. $\Pi(N)$ depicts the set of all run nets of N. $\Pi(x) = \{\pi \in \Pi(N)\colon x \in \pi\}$ is the set of all run nets of N, in which the node x occurs. ⌟

Figure 2 illustrates one possible run net of the net in Fig. 1. The definition of run nets in this work deviates from that of occurrence nets proposed by Polyvyanyy et al. [15], as it is simplified for the context of sound AFW-nets, wherein each place and transition occurs no more than once. Such run nets are analogous to *instance subgraphs* as defined in the workflow graph theory [18].

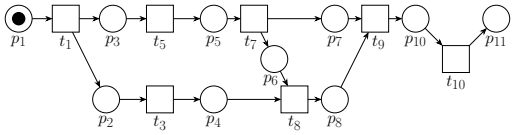

Fig. 2. A run net of the net in Fig. 1.

3 Behavioral Relations

Investigating the behavior of an AFW-net follows two perspectives: (1) The consideration of a single run net π, or (2) the consideration of all its run nets Π. For (1), it can be examined (a) if a node occurs in π and (b) if two nodes that occur in π are *causal* or *concurrent* to each other. In Fig. 1, t_3 can be *concurrent* to t_4 and t_1 is *causal* for t_8. For (2), the consideration of all run nets contains two sub-perspectives: *Existential* and *total* behavior between two nodes. Examples of *existential* behavior are *can co-occur* (there is a run net, in which two nodes occur) and *can conflict* (there is a run net, in which one node occurs but the other does not). In Fig. 1, t_3 *can conflict* with t_6 as there is a run net, in which t_3 occurs but not t_6. t_1 and t_6 *can co-occur* as there is a run net, in which both occur. Examples of *total* behavior are *total co-occur* (two nodes occur always together) and *total conflict* (two nodes never occur together). In Fig. 1, t_5 and t_6 are in *total conflict*, and t_1 and t_3 are in *total co-occur* relation.

Polyvyanyy et al. [15] introduced the *4C Spectrum* to describe all nuances of *conflict*, *co-occurrence*, *causality*, and *concurrency* within process models. Originally, these behavioral relations were defined solely in terms of transitions. However, analyzing the interactions between places — specifically, whether two places can simultaneously contain tokens in the same marking — is also crucial for understanding the full dynamics of process models. Therefore, this work extends the examination of behavioral relations to include all node pairs, encompassing both places and transitions.

If two nodes occur in a run net, they are either in a *causal* or *concurrency* relation [15]. These basic relations are only defined over a single run net (perspective (1)):

Definition 8 (Concurrency and Causality). Let $N = (P, T, F, s, f)$ be a sound AFW-net with one of its run nets $\pi \in \Pi(N)$.
Two nodes $x, y \in P \cup T$ are *concurrent* in π (depicted as $x \parallel_\pi y$) iff

$$x, y \in \pi \quad \wedge \quad Paths_\pi(x, y) = \varnothing \quad \wedge \quad Paths_\pi(y, x) = \varnothing.$$

x is *causal* for y in π, depicted as $x\, causal_\pi\, y$, iff

$$x, y \in \pi \quad \wedge \quad Paths_\pi(x, y) \neq \varnothing \quad [15].$$

Considering the set of all run nets of a net (perspective (2)), *causality* and *concurrency* between two nodes can manifest in many variants within the *4C Spectrum*. In sound AFW-nets, each node can occur at most once within a run. Consequently, these many variants are simplified to *existential* and *total causality/concurrency*. *Existential causality/concurrency* implies that there is at least one run net in which the

two nodes occur and are in a *causal* or *concurrency* relation, respectively. *Total causality/concurrency* indicates that, in every run net where the two nodes occur, they maintain a *causal* or *concurrency* relation. *Can co-occur* and *can conflict* are defined over the existence of at least one run net. In sound AFW-nets, every pair of two nodes x and y is exactly in one of the following four relations for all run nets [15]: (1) *total conflict*; (2) *total co-occur*; (3) *requires* (each run net that contains x also contains y, however, there are run nets with y but not with x); and (4) *independent* (there are run nets containing either x or y, but also run nets, which contain both). In Fig. 1, t_6 *requires* t_3 and t_6 is *independent* from t_{11}.

Based on *occurrence*, *concurrency* and *causality*, the following definition summarizes all behavioral relations resulting from sound AFW-nets [10]. *Please be aware that existential/total concurrency/causality is only defined for run nets, in which two nodes in relation occur, thus "total" implies "existential" [15]:*

Definition 9 (Behavioral Relations). Let $N = (P, T, F, s, f)$ be a sound AFW-net.

The behavioral relations between two different nodes $x, y \in P \cup T$ are [10]:

$$
\begin{aligned}
x \mathit{\ canConflict\ } y &\iff \Pi(x) \setminus \Pi(y) \neq \varnothing \\
x \mathit{\ canCooccur\ } y &\iff \Pi(x) \cap \Pi(y) \neq \varnothing \\
x \mathit{\ totalConflict\ } y &\iff x \mathit{\ canConflict\ } y \wedge y \mathit{\ canConflict\ } x \wedge \overline{x \mathit{\ canCooccur\ } y} \\
x \mathit{\ totalCooccur\ } y &\iff \overline{x \mathit{\ canConflict\ } y} \wedge \overline{y \mathit{\ canConflict\ } x} \wedge x \mathit{\ canCooccur\ } y \\
x \mathit{\ requires\ } y &\iff \overline{x \mathit{\ canConflict\ } y} \wedge y \mathit{\ canConflict\ } x \wedge x \mathit{\ canCooccur\ } y \\
x \mathit{\ independent\ } y &\iff x \mathit{\ canConflict\ } y \wedge y \mathit{\ canConflict\ } x \wedge x \mathit{\ canCooccur\ } y \\
x \mathit{\ concurrent}^{\exists}\ y &\iff \exists \pi \in \Pi(x) \cap \Pi(y) \colon x \parallel_{\pi} y \\
x \mathit{\ concurrent}^{\forall}\ y &\iff \forall \pi \in \Pi(x) \cap \Pi(y) \colon x \parallel_{\pi} y \wedge x \mathit{\ concurrent}^{\exists}\ y \\
x \mathit{\ causal}^{\exists}\ y &\iff \exists \pi \in \Pi(x) \cap \Pi(y) \colon x \mathit{\ causal}_{\pi}\ y \\
x \mathit{\ causal}^{\forall}\ y &\iff \forall \pi \in \Pi(x) \cap \Pi(y) \colon x \mathit{\ causal}_{\pi}\ y \wedge x \mathit{\ causal}^{\exists}\ y
\end{aligned}
$$

4 Structural Computation of the Behavioral Relations

The aim of this paper is to define algorithms that avoid the need for discovery-like computation, which currently exhibits exponential runtime in analyzing AFW-nets. This section examines the structural properties of sound AFW-nets to ultimately compute all behavioral relations between pairs of nodes within a quadratic time complexity.

4.1 Existential Concurrency and Total Concurrency

Prinz et al. [17] have introduced an algorithm with quadratic time complexity for detecting the *concurrency* relation in sound acyclic free-choice workflow nets. Instead of defining concurrency over run nets as in this work, their definition utilizes reachability: Two places are concurrent if there is a reachable marking from the initial marking, in which both places carry tokens. Concurrency between a place and a transition or

two transitions is defined similarly. In Theorem 3 on page 140 of [17], they show that concurrency in sound AFW-nets requires the absence of paths between the nodes in relation. As a consequence, two nodes can only be in a concurrency relation if they do not have a path to each other in a run net as well. Therefore, their definition of concurrency aligns with what is termed *existential concurrency* in this work, thereby enabling the relation's determination between all nodes in a quadratic time:

Proposition 1 (Existential Concurrency). Let $N = (P, T, F, s, f)$ be a sound AFW-net. $concurrent^{\exists}$ can be computed in a quadratic time complexity $O(|P|^2 + |T|^2)$ [17]. □

Concurrency in Def. 8 is defined over the absence of paths between two nodes x and y in at least one *run net*. Fortunately, as mentioned before, soundness restricts and simplifies concurrency: If x and y are in an *existential concurrency* relation, then there cannot be a path between x and y and vice versa in the *AFW-net*:

Theorem 1. Let $N = (P, T, F, s, f)$ be a sound AFW-net with two nodes $x, y \in P \cup T$, $x \neq y$.

$$x\ concurrent^{\exists}\ y \quad \Longrightarrow \quad Paths_N(x,y) = \varnothing \wedge Paths_N(y,x) = \varnothing \tag{1}$$

$$Paths_N(x,y) \neq \varnothing \vee Paths_N(y,x) \neq \varnothing \quad \Longrightarrow \quad \overline{x\ concurrent^{\exists}\ y} \tag{2}$$

Proof. See Prinz et al. [17] (Cor. 4 (p. 141) for (1) and Theorem 3 (pp. 140–141) for (2)). □

Please note that Theorem 1 argues over paths in an AFW-net (N) and *not* over paths in a run net (π), as it is done in Def. 8 of *concurrency*. Of course, an AFW-net without paths between two nodes x and y cannot have any paths between x and y in a run net. As a consequence, each run net, in which such x and y occur, cannot have a path between x and y and vice versa. Therefore, x and y are always *concurrent* if they occur (Def. 8). This corresponds to the definition of *total concurrency* in Def. 9. In summary, *existential concurrency* is equal to *total concurrency*:

Theorem 2 (Existential is Total Concurrency). Let $N = (P, T, F, s, f)$ be a sound AFW-net and $x, y \in P \cup T$ two of its nodes.

$$x\ concurrent^{\exists}\ y \quad \Longleftrightarrow \quad x\ concurrent^{\forall}\ y$$

Proof. Of course, $x\ concurrent^{\forall}\ y \implies x\ concurrent^{\exists}\ y$ directly follows from Def. 9. For this reason, this proof focuses on $x\ concurrent^{\exists}\ y \implies x\ concurrent^{\forall}\ y$.

The preconditions by Theorem 2 are a sound AFW-net $N = (P, T, F, s, f)$ and two nodes $x, y \in P \cup T$. The theorem demands for the validity of $x\ concurrent^{\exists}\ y$. By $x\ concurrent^{\exists}\ y$ and Theorem 1, there is no path between x and y as well as between y and x in the AFW-net N:

$$Paths_N(x,y) = \varnothing \quad \wedge \quad Paths_N(y,x) = \varnothing$$

From this, it follows:

$$\forall \pi \in \Pi(x) \cap \Pi(y)\colon Paths_\pi(x,y) = \varnothing \wedge Paths_\pi(y,x) = \varnothing \wedge x\ concurrent^{\exists}\ y$$

$$\overset{\text{Def. 8}}{\Longleftrightarrow} \quad \forall \pi \in \Pi(x) \cap \Pi(y)\colon x \parallel_\pi y \wedge x\ concurrent^{\exists}\ y$$

Following this and Def. 9: $x\ concurrent^{\forall}\ y$ ✓ □

4.2 Existential Causality and Total Causality

Existential causality requires a path in the AFW-net between two nodes in relation, as Def. 8 needs a path in the run net [9,10]. Since the AFW-nets considered here are sound and free-choice, the existence of a path between x and y implies the existence of a run net, in which x has a path to y, i. e., x and y are in an *existential causality* relation:

Theorem 3 (Existential Causality). Let $N = (P,T,F,s,f)$ be a sound AFW-net and $x,y \in P \cup T$ two of its nodes.

$$x \; causal^{\exists} \; y \quad \Longleftrightarrow \quad Paths_N(x,y) \neq \varnothing$$

Proof. See the contraposition of Lemma 4.2 in Ha and Prinz [10]. □

A node in an acyclic graph has (1) a path to itself and (2) only paths to nodes to which some outputs has a path too. A reverse topological order ensures that a node is only processed after all its output nodes have been addressed (the reverse topological order of the net in Fig. 1 is $(p_{11}, t_{11}, t_{10}, p_{10}, t_9, p_8, p_7, t_8, p_4, t_3, p_2, p_6, t_7, p_5, t_4, t_5, t_6, p_3, t_1, p_1)$). For this reason, a check for all nodes, if there is a path between them in an ASW-net, can be performed in $O(|P|+|T|+|F|)$ [2]. In the worst case, $|F|$ can be quadratic to $|P|+|T|$ according to Def. 1. Thus, finding all pairs in *existential causality* relation can be achieved in a quadratic time complexity of $O\big((|P|+|T|+(|P|+|T|)^2\big) = O(|P|^2+|T|^2)$.

If a node x is *existential causal* to a node y, then there is a path between x and y in the AFW-net by Theorem 3. Following Theorem 1 of *existential concurrency*, x and y can *never* be in a *concurrency* relation as there is a path between x and y in the AFW-net. As a consequence, if x and y are *existential causal*, then x and y are not *existential concurrent*. Polyvyanyy et al. [15] state that if two nodes occur in a run net, they are either in a *concurrency* or *causality* relation. Therefore, x and y must be in a *causality* relation for all run nets, in which both occur. This fits the definition of *total causality* in Def. 9. In summary, *existential causality* implies *total causality*:

Theorem 4 (Existential is Total Causality). Let $N = (P,T,F,s,f)$ be a sound AFW-net and $x,y \in P \cup T$ two of its nodes.

$$x \; causal^{\exists} \; y \quad \Longleftrightarrow \quad x \; causal^{\forall} \; y$$

Proof. Of course, $x \; causal^{\forall} \; y \implies x \; causal^{\exists} \; y$ directly follows from Def. 9. For this reason, this proof focuses on $x \; causal^{\exists} \; y \implies x \; causal^{\forall} \; y$.

By Theorem 4, we have a sound AFW-net $N = (P,T,F,s,f)$ with two nodes $x,y \in P \cup T$. The theorem requires $x \; causal^{\exists} \; y$. Following from $x \; causal^{\exists} \; y$ and Theorem 3, there is a path between x and y in the AFW-net N:

$$Paths_N(x,y) \neq \varnothing \tag{3}$$

Thus, x and y cannot be in an *existential concurrency* relation according to Theorem 1:

$$\begin{aligned} \overline{x \; concurrent^{\exists} \; y} \quad &\overset{\text{Def. 9}}{\Longleftrightarrow} \quad \forall \pi \in \Pi(x) \cap \Pi(y) \colon x \nparallel_\pi y \\ &\overset{\text{Def. 8}}{\Longleftrightarrow} \quad \forall \pi \in \Pi(x) \cap \Pi(y) \colon Paths_\pi(x,y) \neq \varnothing \vee Paths_\pi(y,x) \neq \varnothing \end{aligned}$$

As N is acyclic and given (3), there cannot be a path between y and x in N. Thus, $Paths_\pi(y,x) = \varnothing$ must hold, resulting in $Paths_\pi(y,x) \neq \varnothing$ to be invalid. Finally, this leads to $\forall \pi \in \Pi(x) \cap \Pi(y) \colon Paths_\pi(x,y) \neq \varnothing$ and $x \; causal^{\exists} \; y$. So for all run nets π, in which x and y occur, $x \; causal_\pi \; y$ by Def. 8 and $x \; causal^{\exists} \; y$. This meets Def. 9 of $x \; causal^{\forall} \; y$. ✓ □

4.3 Can Co-occur and Can Conflict

In a sound AFW-net, if two nodes occur in a run net, then always in a *causal* or *concurrency* relation [15], i. e., if two nodes are neither *causal* nor *concurrent*, they *can never occur* together in a run net. This leads to *can co-occur*:

Proposition 2 (Can Co-occur). Let $N = (P,T,F,s,f)$ be a sound AFW-net and two nodes $x,y \in P \cup T$. Following Polyvyanyy et al. [15], it holds:

$$x \; canCooccur \; y \quad \Longleftrightarrow \quad x \; causal^{\exists} \; y \;\vee\; y \; causal^{\exists} \; x \;\vee\; x \; concurrent^{\exists} \; y. \qquad \square$$

The derivation of the *can conflict* relation is not obvious. Instead of deriving it directly, its negation is derived: $\overline{canConflict}$. By Def. 9, $\overline{x \; canConflict \; y}$ is defined as $\Pi(x) \setminus \Pi(y) = \varnothing$ for a sound AFW-net N, which is equal to $\Pi(x) \subseteq \Pi(y)$. Therefore, if there is a run net π with $x \in \pi$, then $y \in \pi$; i. e., x occurs always with y. Soundness of N restricts $\overline{x \; canConflict \; y}$ because there is always a run net, which contains x:

Proposition 3. Let $N = (P,T,F,s,f)$ be an AFW-net.

$$N \text{ sound} \quad \overset{\text{Def. 6}}{\Longrightarrow} \quad \forall x \in P \cup T\colon \Pi(x) \neq \varnothing \qquad \square$$

Resulting from Prop. 3, $\overline{x \; canConflict \; y}$ implies $x \; canCooccur \; y$:

Proposition 4. Let $N = (P,T,F,s,f)$ be a sound AFW-net and $x,y \in P \cup T$ are two of its nodes. It follows from Prop. 3 :

$$\overline{x \; canConflict \; y} \quad \Longrightarrow \quad x \; canCooccur \; y \qquad \square$$

From Prop. 3, it holds that $\Pi(x) \neq \varnothing$ and $\Pi(y) \neq \varnothing$ and $\overline{x \; canConflict \; y}$ is equal to $\Pi(x) \subseteq \Pi(y)$. For this reason, a node x can only be in $\overline{canConflict}$ relation with nodes, which occur together with x in at least one run net. In Fig. 1, p_7 *cannot conflict with* t_8 but not with t_6.

Again, if a node x *cannot conflict* with a node y, then in each run net, in which x occurs, y occurs as well. y may appear before, after, without or concurrent to x. As a first step of the derivation, a special case of *cannot conflict* is considered: All run nets, in which x and y occur *and* x is causal to y, i. e., $\overline{x \; canConflict \; y}$ and $x \; causal^{\exists} \; y$. This special case is called the *trigger* relation:

Definition 10 (Trigger). Let $N = (P,T,F,s,f)$ be a sound AFW-net and two nodes $x,y \in P \cup T$. x *triggers* y iff $x = y \;\vee\; (\overline{x \; canConflict \; y} \wedge x \; causal^{\exists} \; y)$. ⌟

In Fig. 1, t_7 *triggers* t_9 but not t_3. By $x = y$, the *trigger* relation is reflexive. From *causality* and Theorem 3 follows that between two nodes x and y in *trigger* relation, there is a path from x to y in the AFW-net. Furthermore, in each run net where x occurs, there must be a path from x to y. However, it is not necessary for this to be the same path across different run nets. Whether there is always a path between two nodes x and y in every run net in which they occur can be derived by three transitive rules for a sound AFW-net $N = (P,T,F,s,f)$:

Algorithm 1 Determination of the *triggers* relation as an adjacency list R for all nodes of a sound AFW-net $N = (P, T, F, s, f)$.

```
1:  function COMPUTETRIGGER(N = (P, T, F, s, f))
2:      // Initialize R and a list L of nodes to compute.
3:      R ← ∅
4:      L ← P ∪ T in reverse topological order starting from f
5:      for all x ∈ L do
6:          // All nodes in x's postset were processed.
7:          if x ∈ T then
8:              R(x) ← {x} ∪ ⋃_{o∈x•} R(o) // R(x) represents all nodes triggered by x
9:          else
10:             R(x) ← {x} ∪ ⋂_{o∈x•} R(o)
11:     return R
```

(1) $x = y$: Follows directly from Def. 10.
(2) $x \in T$: If one of x's outputs o *triggers* y, then in each π, where o occurs, y occurs as well and there is a path from o to y in π. If x occurs in a π, then $o \in \pi$ (since $x \in T$) and there is the path (x, o) in π. Thus, if x occurs in a π, then $y \in \pi$ with a path from x via o to y in π. In summary, if x *triggers* y, then there must be at least one output o of x, which *triggers* y.
(3) $x \in P$: Since the net is free-choice, x represents the sink ($|x\bullet| = 0$), a simple sequence ($|x\bullet| = 1$) or a decision ($|x\bullet| \geq 2$). The sink is not of interest, as it can only *trigger* itself. Thus, we consider the case $|x\bullet| \geq 1$. If there would be a transition $o \in x\bullet$, which does not *trigger* y, there would be a run net with o but not with y or without a path from o to y. If x occurs in a π and o is fired, then π must not contain y or there is no path from x to y in π, i. e., x does not *trigger* y. Therefore, only if each $o \in x\bullet$ *triggers* y, x *triggers* y as well.

The next equation summarizes the transitive rules:

$$\begin{aligned} x \text{ triggers } y \quad \Longleftrightarrow \quad & x = y \quad \vee \\ & x \in T \ \wedge \ \exists o \in x\bullet\colon o \text{ triggers } y \quad \vee \\ & x \in P \ \wedge \ \forall o \in x\bullet\colon o \text{ triggers } y \end{aligned} \tag{4}$$

These three transitive rules are utilized to formulate Alg. 1, which is designed for detecting the *trigger* relation. The algorithm organizes this relation into an adjacency list R and iteratively processes the net in reverse topological order, beginning with the net's sink (line 4). This sequential processing in reverse order ensures that a node is only processed after all its output nodes have been addressed. If a node x is processed, either it is handled as a transition (line 8) or place (line 10). In both cases, x is added to $R(x)$ following the transitive rule $x = x$. If $x \in T$, x *triggers* all nodes that are *triggered* from x's output places, following the transitive rule $x \in T \ \wedge \ \exists o \in x\bullet$: o *triggers* y. Otherwise, if $x \in P$, x *triggers* all nodes that are *triggered* from all x's output transitions, following the transitive rule $x \in P \ \wedge \ \forall o \in x\bullet$: o *triggers* y.

Time complexity: The topological ordering can be achieved in $O(|P| + |T| + |F|)$ [2]. Given that each node, along with all its postset nodes, is considered exactly once, the

algorithm is guaranteed to terminate. According to Def. 1, the worst-case scenario for the number of flows is quadratic regarding the number of nodes. Consequently, this establishes that the algorithm operates with a quadratic time complexity of $O(|P|^2 + |T|^2)$.

As mentioned before, the *trigger* relation is a special case of $\overline{\textit{canConflict}}$, in which the nodes in relation require *existential/total causality*. This special case helps to derive the general case of $\overline{\textit{canConflict}}$, i. e., in which the nodes in relation do not require *causality*: If a node x *cannot conflict* with a node y in general, there is not the necessity for a path from x to y. However, if x and y occur in a run net, then always with a path from the source s of the AFW-net N to x and y, respectively.

For an imaginative visualization, consider the following metaphor: Imagine a mountain summit representing a node x. From your starting point, the source s, several paths W lead to this summit. If, on each of these paths, there is a trigger z capable of causing an avalanche in the valley (a node y), then reaching the summit (x) invariably results in an avalanche in the valley (y). Conversely, if at least one path exists without such a trigger, it is possible to reach the summit without causing an avalanche in the valley (y). The following theorem uses a similar approach to derive $\overline{\textit{canConflict}}$:

Theorem 5 (Cannot Conflict). Let $N = (P,T,F,s,f)$ be a sound AFW-net with two nodes $x,y \in P \cup T$, $x \neq y$.

$$\overline{x \textit{ canConflict } y} \quad \Longleftrightarrow \quad \forall W \in \textit{Paths}(s,x)\ \exists z \in W\colon z \textit{ triggers } y$$ ⌟

Proof (Theorem 5). The theorem requires a sound AWF-net $N = (P,T,F,s,f)$ with two nodes $x,y \in P \cup T$, $x \neq y$. The proof considers both directions:

$\forall W \in \textit{Paths}(s,x)\ \exists z \in W\colon z \textit{ triggers } y \implies \overline{x \textit{ canConflict } y}$. By Prop. 3, $\Pi(x) \neq \varnothing$. Let $\pi \in \Pi(x)$ be such a run net. Since $x \in \pi$, there is a path $W \in \textit{Paths}_\pi(s,x)$ from the source s to x in π. Furthermore, for all such paths W, there exists a $z \in W$, z *triggers* y. As a consequence, $y \in \pi$ by Def. 10. For this reason:

$$\Pi(x) \subseteq \Pi(y) \quad \Longleftrightarrow \quad \overline{x \textit{ canConflict } y}\ \checkmark$$

$\overline{x \textit{ canConflict } y} \implies \forall W \in \textit{Paths}(s,x)\ \exists z \in W\colon z \textit{ triggers } y$. Proof by contradiction:

$$\overline{x \textit{ canConflict } y} \quad \wedge \quad \exists W \in \textit{Paths}(s,x)\ \forall z \in W\colon \overline{z \textit{ triggers } y} \tag{5}$$

Let W be such a path from the source s to x, on which no node triggers y. Therefore, $\overline{x \textit{ triggers } y}$ and $y \notin W$. Since $\overline{x \textit{ triggers } y}$, there is a path $W' \in \textit{Paths}(s,f)$ from s to the sink f with $\forall z \in W'\colon \overline{z \textit{ triggers } y}$. We construct a run net π that is based on the path W'. If further places and transitions must be added to construct π, which are *not* on path W', we add only places $p \in P$ with $\overline{p \textit{ triggers } y}$ following the transitive rules (4) of *triggers*. Since no node is added to π, which *triggers* y, $y \notin \pi$. Since $x \in W'$, $x \in \pi$. It holds by Def. 9:

$$\Pi(x) \setminus \Pi(y) \neq \varnothing \quad \Longleftrightarrow \quad x \textit{ canConflict } y$$

This contradicts with (5). ↯ Therefore, the original statement must hold. ✓ □

Following the last Theorem 5, $\overline{\textit{canConflict}}$ can be derived from the *trigger* relation. Algorithm 2 accounts for all paths from the source to each node indirectly. The nodes are processed in topological order (line 4), ensuring that a node is not addressed until all its inputs have been processed. This method guarantees that all paths leading from the source to these inputs are considered before the node itself is processed (the topological order of the nodes in Fig. 1 is (p_1, t_1, p_3, t_4, t_5, t_6, p_5, t_7, p_7, p_6, p_2, t_3, p_4, t_8, p_8, t_9, p_{10},

$t_{10}, t_{11}, p_{11})\big)$. A node x *cannot conflict* with all nodes that are *triggered* by x (except itself) and with all nodes that are *triggered* on all paths to x's preset nodes (line 7). Since $\overline{canConflict}$ is irreflexive, x is removed from $R(x)$ (lines 8–9).

Time complexity: Algorithm 2 can compute $\overline{canConflict}$ in $O(2|P|+2|T|+|F|)$, using a linear topological ordering algorithm [2]. Since $|F|$ is quadratic regarding the number of nodes in the worst case by Def. 1, $\overline{canConflict}$ can be computed in $O(|P|^2+|T|^2)$. Note that the negation of $\overline{canConflict}$ leads to the *can conflict* relation. Therefore, *can conflict* can also be computed in $O(|P|^2+|T|^2)$:

Algorithm 2 Determination of the $\overline{canConflict}$ relation as an adjacency list R for all nodes of a sound AFW-net $N = (P,T,F,s,f)$.

```
1:  function COMPUTECANNOTCONFLICT(N = (P,T,F,s,f))
2:      R_trigger ←COMPUTETRIGGER(N) // R_trigger is an adjacency list
3:      // Initialize R and a list L of nodes to compute.
4:      R ← ∅
5:      L ← P ∪ T in a topological order starting from s
6:      for all x ∈ L do
7:          // All nodes in x's preset were already processed.
8:          R(x) ← R_trigger(x) ∪ ⋂_{z∈•x} R(z)
9:      for all x ∈ L do
10:         R(x) ← R(x) ∖ {x}
11:     return R
```

Proposition 5 (Can Conflict). Let $N = (P,T,F,s,f)$ be a sound AFW-net. *canConflict* can be computed in $O(|P|^2+|T|^2)$. ⌟

Proof. The proposition requires a sound AFW-net $N = (P,T,F,s,f)$. $\overline{canConflict}$ can be computed in $O(|P|^2+|T|^2)$ for each pair of nodes with Alg. 2. For this reason, *canConflict* can also be computed in $O(|P|^2+|T|^2)$ after pairwise checking the $\overline{canConflict}$ relation for all nodes. □

4.4 Total Co-Occur, Total Conflict, Requirement, and Independence

Proposition 6 (Total Co-Occur, Total Conflict, Requires, and Independent). Let $N = (P,T,F,s,f)$ be a sound AFW-net. Since *canCooccur* can be computed in $O(|P|^2+|T|^2)$ by Prop. 1 and Prop. 2, and *canConflict* can be computed in $O(|P|^2+|T|^2)$ by Prop. 5, *totalCooccur*, *totalConflict*, *requires*, and *independent* can be computed in a quadratic time complexity $O(|P|^2+|T|^2)$ by applying Def. 9 on each pair of nodes. □

In summary, *it is possible to derive the complete 4C Spectrum for sound AFW-nets in a quadratic time complexity,* $O(|P|^2+|T|^2)$. We have named the set of derivation algorithms *Behavioral Relation Computations*, or *BeRelCo* for short.

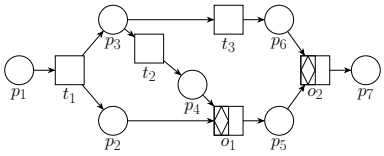

Fig. 3. An example of a net with ORs. Def. 8 of concurrency by Polyvyanyy et al. [15] is counter-intuitive in terms of time as ...

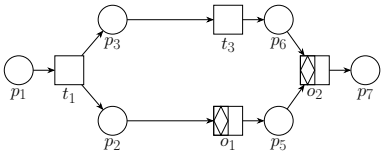

Fig. 4. ... one of its run nets has no path between p_3 and p_5 but p_3 loses its token always before p_5 gets one.

5 Inclusive Behavior in Process Models

Inclusive behavior, a concept in BPM, serves as a middle ground between exclusive behavior — achievable through places with at least two output transitions — and concurrent behavior — characterized by transitions with at least two output places. Nodes exhibiting this inclusive behavior are typically referred to as *OR* nodes. On one hand, an OR node produces tokens for a non-empty subset of its output places. On the other hand, not all inputs of an OR node need to carry a token for the OR to be activated. It is a widely accepted principle that OR nodes, especially those with multiple inputs, "wait for all possible tokens" that might arrive, particularly in acyclic process models [20].

OR nodes can significantly increase the number of potential run nets [10], making the management of ORs crucial in the detection of behavioral relations. Despite their frequent occurrence in process models, OR nodes are not directly representable in Petri nets. One approach to accommodate ORs, as done in this paper, is to extend the Petri net concept: Nets get a further set for inclusive nodes (represented as rectangles with diamonds in figures in the following). We took the accepted semantics of Völzer [20] for acyclic nets (OR-joins "wait for all possible tokens"). Alternatively, ORs can be translated into a Petri net by substituting them with combinations of transitions and places. However, these substitutions can result in a *non*-free-choice net or expand the net to a free-choice variant where the number of nodes exponentially increases compared to the original model [8]. In essence, translating ORs can lead to either a non-free-choice or an exponentially larger net, both posing significant computational challenges due to a combinatorial increase in run nets.

Another challenge arises from the definition of concurrency (as per Def. 8, based on the work of Polyvyanyy et al. [15]), which defines concurrency on the absence of a direct path between two nodes in at least one run net. With the introduction of OR constructs, situations may arise where two nodes appear to be concurrent under Def. 8 but cannot actually be enabled simultaneously or share tokens in the same marking. This discrepancy makes the standard definition of concurrency (e. g., [15, 21]) counter-intuitive in scenarios involving ORs, where parallelism does not align with the expected behavior. For instance, consider nodes p_3 and p_5 in Fig. 3, where transitions marked with diamonds signify ORs. Here, p_3 invariably loses a token *before* p_5 receives one, avoiding premature firing of o_1. Despite this sequential token transfer, these nodes are deemed concurrent by Def. 8 in the run net depicted in Fig. 4.

To address this imprecision in acyclic AFW-nets, soundness offers a solution. We recommend to refine the concurrency definition regarding the results of this paper: *Two*

nodes x and y are concurrent iff there is no path between x and y in the AFW-net *(instead of a run net) and x and y co-occur.*

The *BeRelCo* algorithms are effectively designed to accommodate OR constructs within Petri nets, employing this refined definition of concurrency (as per Definition 8) to seamlessly integrate these elements: (1) The concurrency detection by [17] starts with transitions, as transitions with multiple outputs cause concurrency. Instead of only starting with transitions, the algorithm can simply be extended so that it also starts with ORs, as, in the "worst case", the ORs also cause concurrency for each pair of outputs. (2) Alg. 1 determines the *trigger* relation and distinguishes between places and transitions as the only other algorithm of *BeRelCo*. Line 7 checks whether the current node x is a transition, or not. If x is an OR, then it can represent a simple decision like a place with multiple outputs. Therefore, ORs should be treated as place-like in line 10.

6 Related Work

In the following, we investigate the related work in terms of computational complexity and limitations regarding the investigated behavioral relations defined in Def. 9.

The (causal) behavioral profile of Weidlich et al. [21, 22] was intensively investigated in research. It consists of four relations: *strict order*, *exclusiveness*, *interleaving order*, and *co-occurrence*. The challenge is to align these relations with those in Def. 9. The *strict order* and *exclusiveness* relations are equal to $causal^{\forall}$ and *totalConflict*, respectively. The *co-occurrence relation* is similar to *requires* since it is *not* defined as a symmetric relation [22]. The *interleaving order* relation is a mix of $concurrent^{\exists}$ and *independent*. For sound free-choice workflow nets being decomposable into structured SESE fragments [19], the computational complexity of the four relations is linear for a single pair of nodes and, therefore, cubic for all pairs. In the more general case of unstructured SESE fragments, this complexity increases to $O(|P|^5 + |T|^5)$.

Polyvyanyy et al. [15] and Wolf [23] mapped most of the 4C Spectrum relations to the reachability problem. For acyclic sound free-choice workflow nets, the reachability problem has a polynomial computational complexity [24], however, the concrete polynomial is unknown. Ha and Prinz [10] have investigated the set of behavioral relations of Def. 9 for acyclic sound *workflow graphs*. Their approach has a complexity of $O(|F|^3)$ for a single pair of nodes and, therefore, $O\big(|F|^3 \dot{(}|P|^2 + |T|^2)\big)$ for all pairs.

In summary, the computational complexity for acyclic sound free-choice nets is $O(|P|^5 + |T|^5)$ for $causal^{\forall}$, *totalConflict*, and *requires*; and $O\big(|F|^3 \dot{(}|P|^2 + |T|^2)\big)$ for the other relations. Furthermore, except for the algorithm in [10], the other approaches are only applicable to workflow nets *without* inclusive behavior. *To the best knowledge of the authors, the approach of this paper has the best computational worst-case complexity to compute the behavioral relations of Def. 9 and is the only approach to compute these relations for inclusive behavior in a polynomial complexity.*

7 Evaluation

The *BeRelCo* algorithms, detailed in Sect. 4, have been implemented using a simple script-based approach in PHP for evaluation purposes. This implementation is open-

source and accessible on GitHub[4]. Our experiments were conducted on a machine outfitted with an Intel® Core™ i7 CPU, featuring 14 cores and 64 GB of main memory, running Microsoft Windows 11 Professional. PHP version 8 was utilized for the execution of the scripts. To guarantee the reliability of our results, each runtime measurement was performed ten times. We excluded the fastest and slowest runs from this set and calculated the mean values from the remaining measurements. We then compared the performance of the BeRelCo algorithms against a brute-force approach for deriving all run nets according to Def. 7, hereinafter referred to as 'RunNets'. The brute-force approach was chosen since all approaches in the literature finally depends on a similar strategy for inherently unstructured fragments (e. g., for [22] and [10]).

The evaluation of the algorithms was conducted using two well-known datasets: the IBM Websphere Business Modeler dataset [7, 12], henceforth referred to as the *IBM* dataset, which includes 1,386 files, and the SAP Reference Models dataset [4], hereafter referred to as the *SAP* dataset, containing 604 files. The data sets were selected because they represent real process models for which numerous analyses can already be found in the literature. Given the algorithms' prerequisites for analyzing sound AFW-nets, a subset of the datasets was selected for investigation — specifically, 604 nets from the IBM collection and 414 from the SAP collection are sound acyclic free-choice workflow nets (all nets of IBM are free-choice, 178 nets are cyclic, and, from the remaining 1208 acyclic nets, 604 are unsound; all nets of SAP are free-choice, 31 nets are cyclic, and, from the remaining 569 acyclic nets, 152 are unsound). The nets within the IBM dataset were provided in PNML format, facilitating direct analysis. Conversely, the SAP dataset's nets, present in a simplified JSON format that describes BPMN-like models with AND, XOR, and OR nodes, required conversion into Petri nets for compatibility. This conversion was guided by the extended Petri net concept discussed in Sect. 5.

Subsequently, runtime measurements of both datasets are presented in the form x_I | y_S with measures x_I for *IBM* and y_S for *SAP*. The size of the nets under investigation varies, with 75% of the nets having 57_I | 36_S nodes or fewer. The largest nets contain a maximum of 546_I | 133_S nodes. Places are more frequent than transitions ($61\%\pm4\%_I$ | $56\%\pm6\%_S$ of all nodes are places). The SAP nets contain $7\%\pm7\%$ OR nodes.

Upon application to *IBM* and *SAP*, both algorithms successfully identified the same node relations across all eligible nets. The main objective was to establish the *BeRelCo* algorithm as a more efficient alternative to *RunNets* or similar exploratory algorithms. Consequently, we assessed the computational time required by both algorithms to derive all relations within a net. The findings indicate that the *BeRelCo* algorithms outperform *RunNets* in processing speed across both datasets: It needs just 0.655_I | 0.328_S [s] to compute $>6.3\text{M}_I$ | $>1.2\text{M}_S$ pairs of nodes being in relation. In contrast, the *RunNets* algorithm requires 4.038_I | 435.066_S [s] for doing the same job, i. e., the *BeRelCo* algorithm is approx. 6_I | 1326_S times faster. Notably, *RunNets* was unable to process 2 nets from *SAP* because the number of run nets exceeded 500k. We have introduced this limit artificially after some experiments, as otherwise the allocated main memory would be exceeded and lead to an uncatchable error in PHP. Figure 5 illustrates the computation times of some of the nets in relation to their size (number of nodes). It shows the computation times for three sets: (1) For nets requiring more than 0.01 [s] for *RunNets*

[4] `https://github.com/guybrushPrince/berelco`

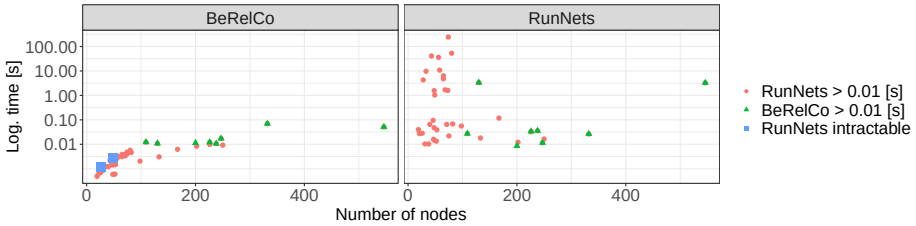

Fig. 5. Comparison between the number of nodes and the logarithmic time to compute all behavioral relations for three different subsets of all nets regarding their computation times.

(illustrated as red dots); (2) for nets requiring more than 0.01 [s] for *BeRelCo* (illustrated with green triangles) and (3) for intractable nets with *RunNets*. In summary, the *BeRelCo* algorithm demonstrates superior efficiency, processing all nets that took more than 0.01 [s] with *RunNets* in less than 0.01 [s]. Furthermore, any net that required more than 0.01 [s] with *BeRelCo* was processed in a similar or longer duration by *RunNets*. Notably, intractable nets with *RunNets* were processed by *BeRelCo* in under 0.01 [s]. *BeRelCo* consistently achieves tractable processing times for each net, with a maximum duration of up to 0.1 [s], showcasing a significant improvement.

8 Conclusion

Systematically identifying business process models within extensive collections poses a considerable challenge. Process queries facilitate the identification of models that meet specific characteristics, relying on behavioral relations, to illustrate how tasks in a process model interact during execution. The *4C Spectrum* — comprising *conflict*, *co-occurrence*, *causality*, and *concurrency* — provides a comprehensive framework for these behavioral relations. However, current computational methods for analyzing these relations can be time-consuming for various process models. This paper introduces a suite of algorithms, named *Behavioral Relation Computations* (*BeRelCo*), designed to efficiently detect all behavioral relations within the *4C Spectrum* in quadratic time complexity, $O(|P|^2 + |T|^2)$, for models that can be represented as acyclic sound free-choice workflow nets. Our experiments validate the *BeRelCo* algorithms' effectiveness, notably in models with numerous execution traces.

Industrial process modeling languages, often aligned with workflow graphs, can be depicted as free-choice workflow nets [8]. *Soundness* was identified as a critical requirement [5]. Therefore, the algorithms we have introduced hold substantial value for the BPM community, not only facilitating process queries but also underpinning process similarity analysis, compliance checking, and other key BPM activities.

Currently, our approach is limited to acyclic nets. Future efforts will focus on extending these algorithms to handle cyclic nets through loop decomposition [16] by devising combinatorial rules for behavioral relations for the resulting acyclic nets. Moreover, we aim to adapt our algorithms for models that feature duplicated labels, where tasks may occur multiple times, broadening their applicability and utility in complex process model analyses.

References

1. van der Aalst, W.M.P.: Verification of workflow nets. In: Azéma, P., Balbo, G. (eds.) Application and Theory of Petri Nets 1997, 18th International Conference, ICATPN '97, Toulouse, France, June 23-27, 1997, Proceedings. Lecture Notes in Computer Science, vol. 1248, pp. 407–426. Springer (1997). https://doi.org/10.1007/3-540-63139-9_48
2. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 3rd Edition. MIT Press (2009)
3. Czerwinski, W., Lasota, S., Lazic, R., Leroux, J., Mazowiecki, F.: The reachability problem for Petri nets is not elementary. J. ACM **68**(1), 7:1–7:28 (2021)
4. van Dongen, B.F., Jansen-Vullers, M.H., Verbeek, H.M.W., van der Aalst, W.M.P.: Verification of the SAP reference models using EPC reduction, state-space analysis, and invariants. Comput. Ind. **58**(6), 578–601 (2007)
5. van Dongen, B.F., Mendling, J., van der Aalst, W.M.P.: Structural patterns for soundness of business process models. In: Tenth IEEE International Enterprise Distributed Object Computing Conference (EDOC 2006), 16-20 October 2006, Hong Kong, China. pp. 116–128. IEEE Computer Society (2006). https://doi.org/10.1109/EDOC.2006.56
6. Dumas, M., Rosa, M.L., Mendling, J., Reijers, H.A.: Fundamentals of Business Process Management, Second Edition. Springer (2018)
7. Fahland, D., Favre, C., Koehler, J., Lohmann, N., Völzer, H., Wolf, K.: Analysis on demand: Instantaneous soundness checking of industrial business process models. Data Knowl. Eng. **70**(5), 448–466 (2011), https://web.archive.org/web/20131208132841/http://service-technology.org/publications/fahlandfjklvw_2009_bpm, (Mar. 2024)
8. Favre, C., Fahland, D., Völzer, H.: The relationship between workflow graphs and free-choice workflow nets. Inf. Syst. **47**, 197–219 (2015)
9. García-Bañuelos, L., van Beest, N., Dumas, M., Rosa, M.L., Mertens, W.: Complete and interpretable conformance checking of business processes. IEEE Trans. Software Eng. **44**(3), 262–290 (2018). https://doi.org/10.1109/TSE.2017.2668418
10. Ha, N.L., Prinz, T.M.: Partitioning behavioral retrieval: An efficient computational approach with transitive rules. IEEE Access **9**, 112043–112056 (2021)
11. Kunze, M., Weidlich, M., Weske, M.: Behavioral similarity - A proper metric. In: Rinderle-Ma, S., Toumani, F., Wolf, K. (eds.) Business Process Management - 9th International Conference, BPM 2011, Clermont-Ferrand, France, August 30 - September 2, 2011. Proceedings. Lecture Notes in Computer Science, vol. 6896, pp. 166–181. Springer (2011)
12. Mendling, J., Verbeek, H.M.W., van Dongen, B.F., van der Aalst, W.M.P., Neumann, G.: Detection and prediction of errors in epcs of the SAP reference model. Data Knowl. Eng. **64**(1), 312–329 (2008). https://doi.org/10.1016/j.datak.2007.06.019
13. Polyvyanyy, A., Pika, A., ter Hofstede, A.H.M.: Scenario-based process querying for compliance, reuse, and standardization. Inf. Syst. **93**, 101563 (2020)
14. Polyvyanyy, A., ter Hofstede, A.H., La Rosa, M., Ouyang, C., Pika, A.: Process Query Language: Design, Implementation, and Evaluation. Information Systems **122**, 102337 (2024)
15. Polyvyanyy, A., Weidlich, M., Conforti, R., Rosa, M.L., ter Hofstede, A.H.M.: The 4C Spectrum of fundamental behavioral relations for concurrent systems. In: Ciardo, G., Kindler, E. (eds.) Application and Theory of Petri Nets and Concurrency - 35th International Conference, PETRI NETS 2014, Tunis, Tunisia, June 23-27, 2014. Proceedings. Lecture Notes in Computer Science, vol. 8489, pp. 210–232. Springer (2014)
16. Prinz, T.M., Choi, Y., Ha, N.L.: Understanding and decomposing control-flow loops in business process models. In: Ciccio, C.D., Dijkman, R.M., del-Río-Ortega, A., Rinderle-Ma, S. (eds.) Business Process Management - 20th International Conference, BPM 2022, Münster, Germany, September 11-16, 2022, Proceedings. Lecture Notes in Computer Science, vol. 13420, pp. 307–323. Springer (2022)

17. Prinz, T.M., Klaus, J., van Beest, N.R.T.P.: Pushing the limits: Concurrency detection in acyclic sound free-choice workflow nets in $O(P^2 + T^2)$. In: Köhler-Bussmeier, M., Moldt, D., Rölke, H. (eds.) Proceedings of the International Workshop on Petri Nets and Software Engineering 2024 co-located with the 45th International Conference on Application and Theory of Petri Nets and Concurrency (PETRI NETS 2024), June 24 - 25, 2024, Geneva, Switzerland. CEUR Workshop Proceedings, vol. 3730, pp. 132–154. CEUR-WS.org (2024), https://ceur-ws.org/Vol-3730/paper08.pdf
18. Sadiq, W., Orlowska, M.E.: Analyzing process models using graph reduction techniques. Inf. Syst. **25**(2), 117–134 (2000). https://doi.org/10.1016/S0306-4379(00)00012-0
19. Vanhatalo, J., Völzer, H., Koehler, J.: The refined process structure tree. Data Knowl. Eng. **68**(9), 793–818 (2009). https://doi.org/10.1016/j.datak.2009.02.015
20. Völzer, H.: A new semantics for the inclusive converging gateway in safe processes. In: Hull, R., Mendling, J., Tai, S. (eds.) Business Process Management - 8th International Conference, BPM 2010, Hoboken, NJ, USA, September 13-16, 2010. Proceedings. vol. 6336, Lecture Notes in Computer Science, pp. 294–309. Springer (2010)
21. Weidlich, M., Mendling, J., Weske, M.: Efficient consistency measurement based on behavioral profiles of process models. IEEE Trans. Software Eng. **37**(3), 410–429 (2011)
22. Weidlich, M., Polyvyanyy, A., Mendling, J., Weske, M.: Causal behavioural profiles - efficient computation, applications, and evaluation. Fundam. Informaticae **113**(3-4), 399–435 (2011). https://doi.org/10.3233/FI-2011-614
23. Wolf, K.: Interleaving based model checking of concurrency and causality. Fundam. Informaticae **161**(4), 423–445 (2018). https://doi.org/10.3233/FI-2018-1709
24. Yamaguchi, S.: Polynomial time verification of reachability in sound extended free-choice workflow nets. IEICE Trans. Fundam. Electron. Commun. Comput. Sci. **97-A**(2), 468–475 (2014). https://doi.org/10.1587/TRANSFUN.E97.A.468