Aggregating Causal Runs into Workflow Nets

B.F. van Dongen, J. Desel, and W.M.P. van der Aalst

b.f.v.dongen@tm.tue.nl, joerg.desel@fernuni-hagen.de,
w.m.p.v.d.aalst@tm.tue.nl

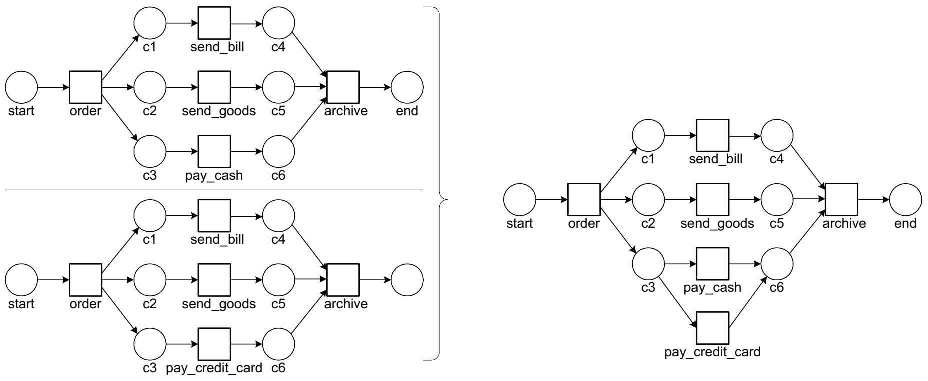
Abstract. This paper provides three algorithms for constructing system nets from sets of partially-ordered causal runs. The three aggregation algorithms differ with respect to the assumptions about the information contained in the causal runs. Specifically, we look at the situations where labels of conditions (i.e. references to places) or events (i.e. references to transitions) are unknown. Since the paper focusses on aggregation in the context of process mining, we solely look at workflow nets, i.e. the class of Petri nets with unique start and end places. The difference of the work presented here and most work on process mining is the assumption that events are logged as partial orders instead of linear traces. Although the work is inspired by applications in the process mining and workflow domains, the results are generic and can be applied in other application domains.

1 Introduction

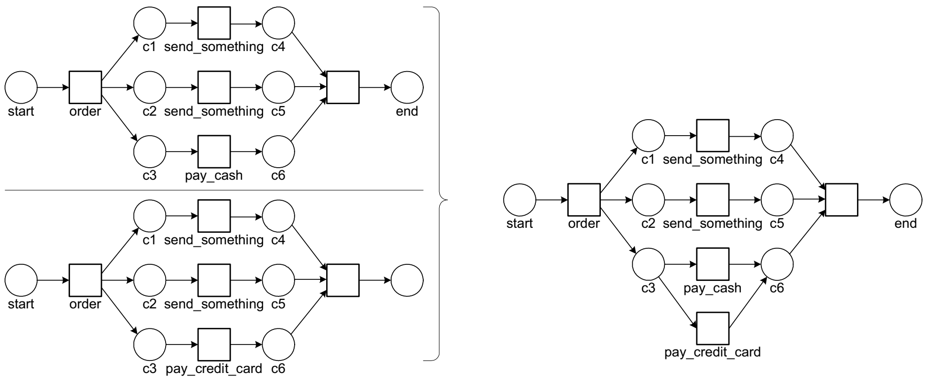
This paper proposes different approaches to "discover" process models from observing runs, i.e., runs (also known as causal nets or occurrence nets, see e.g. [1]) are aggregated into a single Petri net that captures the observed behaviour. Runs provide information about events together with pre- and post-conditions which constitute a (partial) order between these events.

There are many techniques to discover sequential process models based on event logs (also known as transaction logs, audit trails, etc). People working on process mining techniques [2] also tackle situations where processes may be concurrent and the set of observations is incomplete. The set of possible sequences is typically larger than the number of process instances thus making it unrealistic to assume that all possible combinations of behaviour have been observed.

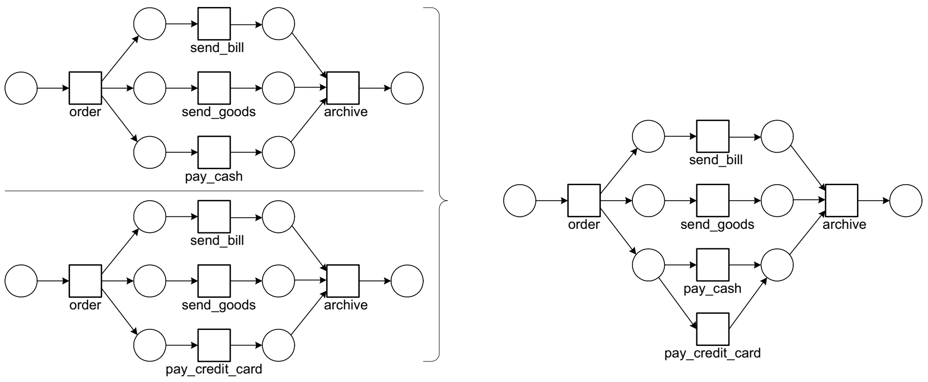
In many applications, the event log is linear, e.g., sorted based on timestamps, and an approach based on runs is not applicable. However, there are processes where it is possible to monitor causal dependencies (e.g., by analyzing the dataflows). We also encountered some systems that actually log behaviour using a representation similar to runs. The ad-hoc workflow management system InConcert of Tibco (formerly Xerox) allows end users to define and modify process instances (e.g., customer orders) while capturing the causal dependencies between the various activities. The representation used directly corresponds to runs. The analysis tool ARIS PPM (Process Performance Monitor) of IDS Scheer can extract runs represented as so-called *instance EPCs* (Event-driven Process Chains) from systems such as SAP R/3 and Staffware. These examples show that in real-life systems and processes runs can be recorded or already are being recorded, thus motivating the work presented in this contribution.



(a) Known event and condition labels.



(b) Known condition labels and unknown or non-unique event labels.



(c) Known event labels and unknown condition labels.

Fig. 1. Example describing the three problems tackled in this paper.

To introduce the main topic of this paper we use the examples shown in Figure 1. The left-hand side of this figure shows several runs. These are the behaviours that have been observed by extracting information from e.g. some enterprise information system. The right-hand side shows the models that we would like to discover by aggregating the runs shown on the left-hand side. Runs are represented by acyclic Petri nets without any choices. Figure 1 (a) shows the easiest situation. Here we assume that in the run all event and condition labels have been recorded in some event log. There are two runs and it is easy to see that the aggregated model can indeed reproduce these two runs, and no other runs are possible. However, in general not all possible runs need to be present. In most application domains the number of possible runs is larger than the actual number of process instances. Figure 1 (b) describes a more complex problem where not all event labels are recorded or where the same label may refer to different transitions. For example, in Figure 1 (b) the archive event is no longer visible and the two send events (`send_goods` and `send_bill`) cannot be distinguished, since both of them are recorded as `send_something`. Figure 1 (c) illustrates the most challenging problem, i.e., the event labels are given but the condition labels are not recorded at all. Nevertheless, the Petri net on the right is the most likely candidate process to exhibit such behaviour. In this paper we will tackle the problem of aggregating runs into a Petri net that can generate these runs. We will show that it is possible to do this for the three situations depicted in Figure 1.

The generation of system nets from their causal runs has been investigated before. The first publication on this topic is [3]. Here the basis is assumed to be the set of all runs. These runs are folded, i.e., events representing the occurrence of the same transition are identified, and so are conditions representing a token on the same place. In [4] a similar folding approach is taken, but there the authors start with a set of causal runs, as we do in the present paper. [4] does not present algorithms in details for the aggregation of runs but rather concentrates on correctness criteria for the derived system net. [5] extracts Petri nets from models which are based on Message Sequence Charts (MSCs), a concept quite similar to causal runs. Less related is the work presented in [6], where a special variant of MSCs is used to generate a system implementation.

In more recent papers [7], so-called regions are defined for partial orders of events representing runs. These regions correspond to anonymous places of a synthesized Place/Transition net, which can generate these partial orders. In contrast to our work, the considered partial orders are any linearizations of causal orders, i.e., two ordered events can either occur in a sequence (then there is a causal run with a condition "between" the events) or they can occur concurrently. Consequently, conditions representing tokens on places are not considered in these partial orders whereas our approach heavily depends on these conditions.

The goal of process mining is to extract information about processes from event logs. One of its aspects focusses on finding a process specification, based on a set of executions of that process, i.e. a process log is taken as a starting point. A variety of algorithms have been proposed to generate a process model based on this log. Typically, such a log is considered to consist of cases (i.e. process instances, for example one insurance claim in a process dealing with insurance claims) and all tasks in each case are totally ordered (typically based on the timestamps). In this paper, we take a different approach. We start by looking at so-called runs. These runs are a *partial* ordering on the tasks within each case. However, in addition to the partial ordering of tasks, we may have information about the local states of the system from which the logs originated, i.e. for each event the pre- and post-conditions are known. This closely relates to the process mining algorithms presented in [8] and [9]. However, also in these papers only causal dependencies between events and no state information is assumed to be known.

In this paper, we provide three algorithms for the aggregation of runs. First, we assume we indeed have full knowledge of each event, its preconditions and its postconditions. Then, we assume that we cannot uniquely identify events, i.e. the label of an event may refer to multiple transitions. Finally, we provide an algorithm that assumes less knowledge about pre- and post-conditions. Before we elaborate on our results, we first provide some preliminary definitions that we use throughout the paper.

2 Preliminaries

Let $G = (N, E)$ be a directed graph, i.e. N is the set of nodes and $E \subseteq N \times N$ is the set of edges. If $N' \subseteq N$, we say that $G' = (N', E \cap (N' \times N'))$ is a subgraph of G. $N' \subseteq N$ generates a *maximal connected subgraph* if it is a maximal set of vertices generating a connected subgraph. For $n \in N$, we define $\overset{G}{\bullet} n = \{m \in N \mid (m, n) \in E\}$ as the pre-set and $n \overset{G}{\bullet} = \{m \in N \mid (n, m) \in E\}$ as the post-set of n with respect to the graph G. If the context is clear, the superscript G is omitted.

Let $G = (N, E)$ be a graph. Let μ be a set of colors. A function $f : N \to \mu$ is a coloring function if, for all $(n_1, n_2) \in E$, either $n_1 = n_2$ or $f(n_1) \neq f(n_2)$.

As stated in the introduction, our starting point is not only a partial order of events within a case, but also information about the state of a case. Since we want to be able to represent both events and states, Petri nets provide a natural basis for our approach. In this paper, we use the standard definition of finite marked Place/Transition (P/T-nets) nets $N = (P, T, F, M_0)$. We restrict ourselves to P/T nets where for all transitions t holds that $\bullet t \neq \emptyset$ and $t\bullet \neq \emptyset$.

We use square brackets for the enumeration of the elements of a bag representing a marking of a P/T-net, e.g. $[2a, b, 3c]$ denotes the bag with two a's, one b, and three c's. The sum of two bags ($X \uplus Y$), the presence of an element in a bag ($a \in X$), and the notion of subbags ($X \leq Y$) are defined in a straightforward way, and they can handle a mixture of sets and bags. Furthermore, $\biguplus_{a \in A} \big(f(a)\big)$ denotes the sum over the bags that are results of function f applied to the elements a of a bag A.

Petri nets specify processes. The behaviour of a Petri net is given in terms of causal nets, representing process instances (i.e. cases). Therefore, we introduce some concepts (notation taken from [1]). First, we introduce the notion of a causal net, this is a specification of one process instance of some process specification.

Definition 2.1. (Causal net)
A P/T net (C, E, K, S_0) is called a causal net if:

- for every place $c \in C$ holds that $|\bullet c| \leq 1$ and $|c\bullet| \leq 1$,
- the transitive closure of K is irreflexive, i.e. it is a partial order on $C \cup E$,
- for each place $c \in C$ holds that $S_0(c) = 1$ if $\bullet c = \emptyset$ and $S_0(c) = 0$ if $\bullet c \neq \emptyset$.

In causal nets, we refer to places as *conditions* and to transitions as *events*.

Each event of a causal net should refer to a transition of a corresponding P/T-net and each condition should refer to a token on some place of the P/T-net. These references are made by mapping the conditions and the events of a causal net onto places and transitions, respectively, of a Petri net. We call the combination of a causal net and such a mapping a *run*.

Definition 2.2. (Run)
A run (N, α, β) of a P/T-net (P, T, F, M_0) is a causal net $N = (C, E, K, S_0)$, together with two mappings $\alpha : C \rightarrow P$ and $\beta : E \rightarrow T$, such that:

- For each event (transition) $e \in E$, the mapping α induces a bijection from $\bullet e$ to $\bullet\beta(e)$ and a bijection from $e\bullet$ to $\beta(e)\bullet$,
- $\alpha(S_0) = M_0$ where α is generalized to markings by $\alpha : (C \rightarrow \mathbb{N}) \rightarrow (P \rightarrow \mathbb{N})$, such that $\alpha(S_0)(p) = \sum_{c|\alpha(c)=p} S_0(c)$.

The causal behaviour of the P/T-net (P, T, F, M_0) is defined as its set of runs. To avoid confusion, the P/T-net (P, T, F, M_0) is called *system net* in the sequel.

3 Aggregation of Runs

In this section, we will introduce an approach that takes a set of runs as a starting point. From this set of runs, a system net is constructed. Moreover, we need to find a mapping from all the events and conditions in the causal nets to the transitions and places in the system net. From Definition 2.2, we know that there should exist a bijection between all places in the pre- or post-set of an event in some causal net and the pre- or post-set of a transition in a system net. *Therefore, two conditions belonging to the pre- or post-set of an event should not be mapped onto the same label.* This restriction is in fact merely another way to express the fact that our P/T-nets do not allow for more than one edge between a place and a transition or vice versa. More generally, we define a labelling function on the nodes of a graph as a function that does not give the same label to two nodes that have a common element in their pre-sets or a common element in their post-sets.

Definition 3.1. (Labelling function)
Let μ be a set of labels. Let $G = (N, E)$ be a graph. Let $R = \{(n_1, n_2) \subseteq N \times N \mid n_1 \overset{G}{\bullet} \cap n_2 \overset{G}{\bullet} \neq \emptyset \vee \overset{G}{\bullet} n_1 \cap \overset{G}{\bullet} n_2 \neq \emptyset\}$. We define $f : N \rightarrow \mu$ to be a *labelling function* if f is a coloring function on the graph (N, R).

We focus on the aggregation of runs that originate from a Petri net with clearly defined starting state and completion state, i.e. processes that describe a lifespan of some case. This assumption is very natural in the context of workflow management systems. However, it applies to many other domains where processes are instantiated for specific cases. Hence, we will limit ourselves to a special class of Petri nets, namely workflow nets.

Definition 3.2. (Workflow nets)
A P/T-net $N = (P, T, F, M_0)$ is a *workflow net* (WF-net) if:

1. *object creation*: P contains an input place p_{ini} such that $\bullet p_{ini} = \emptyset$,
2. *object completion*: P contains an output place p_{out} such that $p_{out}\bullet = \emptyset$,
3. *connectedness*: there is a path from p_{ini} to every node and from every node to p_{out},
4. *initial marking*: $M_0 = [p_{ini}]$, i.e. the initial marking marks only p_{ini}.

As a consequence, a WF-net has exactly one one input place. When looking at a run of a WF-net, we can therefore conclude that there is exactly one condition containing a token initially and all other conditions do not contain tokens. A set of causal nets fulfilling this condition and some structural consequences is called a *causal set*.

Definition 3.3. (Causal set)
Let $n \in \mathbb{N}$ and let $\Phi = \{(C_i, E_i, K_i, S_i) \mid 0 \leq i < n\}$ be a set of causal nets. We call this set a *causal set* if and only if, for $0 \leq i < n$ holds:

- all sets C_i, E_i, K_i are disjoint,
- for $0 \leq i \leq n$, $\sum_{c \in C_i} S_i(c) = 1$, i.e. exactly one condition has an empty pre-set,
- for $e \in E_i$, if $S_i(c) = 1$ for some $c \in \bullet e$ then $\{c\} = \bullet e$,
- for $e \in E_i$, if $c\bullet = \emptyset$ for some $c \in e\bullet$, then $\{c\} = e\bullet$.

The concept of constructing a system net from a causal set is called *aggregation*. This concept can be applied if we assume that each causal net in the given set can be called a run of some system net. From Definition 2.2 we know that we need two mappings α and β satisfying the two properties mentioned. Using the definition of a system net and the relation between system nets and runs, we can conclude that any aggregation algorithm should have the following functionality:

- it should provide the set of places P of the system net,
- it should provide the set of transitions T of the system net,
- it should provide the flow relation F of the system net,
- it should provide the initial marking M_0 of the system net,
- for each causal net in the causal set, it should provide the mappings $\alpha_i : C_i \rightarrow P$ and $\beta_i : E_i \rightarrow T$, in such a way that for all causal nets, $\alpha_i(S_i)$ is the same (i.e. they have the same initial marking) and they induce bijections between pre- and post-sets of events and their corresponding transitions.

Each event that appears in a causal net has a corresponding transition in the original system net. Moreover, bijections exist between the pre- and post-sets of this event and the corresponding transitions. In order to express this in terms of labelling functions of causal nets, we formalize this concept using the notion of *transition equivalence*.

Definition 3.4. (Transition equivalence)
Let μ, ν be two disjoint sets of labels. Let $\Phi = \{N_i = (C_i, E_i, K_i, S_i) \mid 0 \leq i < n\}$ be a causal set, and let $\Psi = \{(\alpha_i : C_i \rightarrow \mu, \beta_i : E_i \rightarrow \nu) \mid 0 \leq i < n\}$ be a set of labelling functions of (C_i, E_i, K_i, S_i). We define (Φ, Ψ) to *respect transition equivalence* if and only if for each $e_i \in E_i$ and $e_j \in E_j$ with $\beta_i(e_i) = \beta_j(e_j)$ the following holds:

- for each $(c_i, e_i) \in K_i$ we have a $(c_j, e_j) \in K_j$ such that $\alpha_i(c_i) = \alpha_j(c_j)$,
- for each $(e_i, c_i) \in K_i$ we have a $(e_j, c_j) \in K_j$ such that $\alpha_i(c_i) = \alpha_j(c_j)$.

4 Aggregation with Known Labels

In this section, we present an aggregation algorithm that assumes that we know all mapping functions, and that these mapping functions adhere to the definition of a run. To illustrate the aggregation process, we make use of a running example. Consider Figure 2 where four part of runs are shown. We assume that the events A,B,C,D,E,F and G do not appear in any other part of each run. Our first aggregation algorithm is

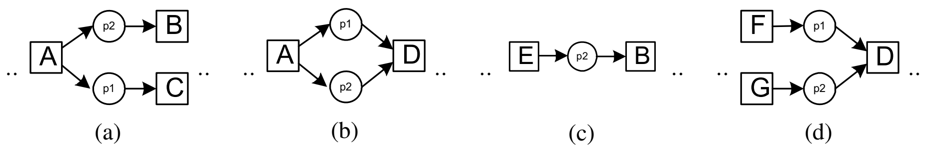


Fig. 2. Four examples of parts of runs.

called the ALK aggregation algorithm (short for "All Labels Known"). This algorithm assumes all information to be present, such as in Figure 2, i.e. it assumes known labels for events and known labels for conditions. These labels refer to concrete transitions and places in the aggregated system net.

Definition 4.1. (ALK aggregation algorithm)
Let μ, ν be two disjoint sets of labels. Let Φ be a causal set of size n with causal nets (C_i, E_i, K_i, S_i) $(0 \leq i < n)$.

Furthermore, let $\{(\alpha_i : C_i \to \mu, \beta_i : E_i \to \nu) \mid 0 \leq i < n\}$ be a set of labelling functions respecting transition equivalence, such that for all causal nets $\alpha_i(S_i)$ is the same. We construct the system net (P, T, F, M_0) belonging to these runs as follows:

- $P = \bigcup_{0 \leq i < n} \text{rng}(\alpha_i)$ is the set of places (note that $P \subseteq \mu$)[1],
- $T = \bigcup_{0 \leq i < n} \text{rng}(\beta_i)$ is the set of transitions (note that $T \subseteq \nu$),
- $F = \bigcup_{0 \leq i < n} \{(\alpha_i(c), \beta_i(e)) \in P \times T \mid (c, e) \in K_i \cap (C_i \times E_i)\} \cup$
  $\bigcup_{0 \leq i < n} \{(\beta_i(e), \alpha_i(c)) \in T \times P \mid (e, c) \in K_i \cap (E_i \times C_i)\}$
  is the flow relation,
- $M_0 = \alpha_0(S_0)$ is the initial marking.

The result of the ALK aggregation algorithm from Definition 4.1 for the parts presented in Figure 2 is shown in Figure 3. Another example is given in Figure 1(a).

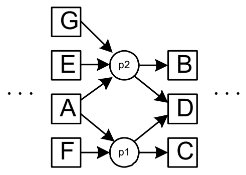


Fig. 3. The aggregated Petri net.

The aggregated net shown in Figure 3 can actually generate the runs of Figure 2. This is always the case after applying the ALK aggregation algorithm:

[1] With rng we denote the range of a function, i.e. rng$(f) = \{f(x) \mid x \in \text{dom}(f)\}$

Property 4.2. (The ALK algorithm is correct)
For $0 \leq i < n, N_i = (C_i, E_i, K_i, S_i), (N_i, \alpha_i, \beta_i)$ is indeed a run of $\sigma = (P, T, F, M_0)$ (i.e., the requirements stated in Definition 2.2 are fulfilled).

The ALK algorithm is a rather trivial aggregation over a set of runs. However, it is assumed that the mapping functions α_i and β_i are known for each causal net. Furthermore, we assume two sets of labels μ and ν to be known. However, when applying these techniques in the context of process mining, it is often not realistic to assume that all of these are present. Therefore, in the remainder of this paper, we relax some of these assumptions to obtain more usable process mining algorithms.

5 Aggregation with Duplicate or Missing Transition Labels

In this section, we will assume that the causal set used to generate the system net and the labelling functions do not respect transition equivalence. We introduce an algorithm to change the labelling function for events in such a way that this property holds again. In the domain of process mining, the problem of so-called "duplicate transitions" (i.e. several transitions with the same label) is well-known (cf. [10–12]). Therefore, there is a need for algorithms to find out which events actually belong to which transition. We assume that we have causal nets with labelling functions, where some events have the same label, even though they may refer to different transitions (see Figure 4).

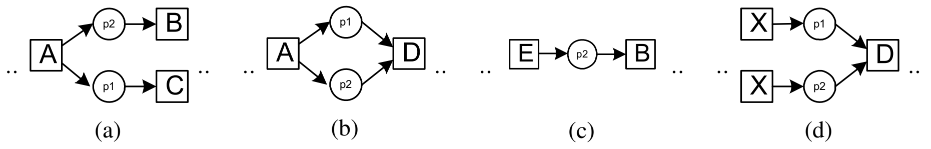


Fig. 4. Four examples of parts of runs.

In terms of an aggregation algorithm, the problem of duplicate labels translates to the situation where the property of transition equivalence is not satisfied. Since the aggregation algorithm presented in the previous section only works if this property holds, we provide an algorithm to redefine the labelling functions for events.

Definition 5.1. (Relabelling algorithm)
Let μ, ν be two disjoint sets of labels. Let $\Phi = \{N_i \mid 0 \leq i < n \wedge N_i = (C_i, E_i, K_i, S_i)\}$ be a causal set and let $\Psi = \{(\alpha_i : C_i \to \mu, \beta_i : E_i \to \nu) \mid 0 \leq i < n\}$ be a set of labelling functions in (C_i, E_i, K_i, S_i) such that $\alpha_i(S_i)$ is the same for all causal nets. Furthermore, assume that μ and ν are minimal, i.e. $\bigcup_{0 \leq i < n} \text{rng}(\alpha_i) = \mu$ and $\bigcup_{0 \leq i < n} \text{rng}(\beta_i) = \nu$. Let $E^\star = \bigcup_{0 \leq i < n} E_i$ be the set of all events in the causal set.

We define the relabelling algorithm as follows:

1. Define $\bowtie \subseteq E^\star \times E^\star$ as an equivalence relation on the elements of $E^\star$ in such a way that $e_i \bowtie e_j$ with $e_i \in E_i$ and $e_j \in E_j$ if and only if $\beta_i(e_i) = \beta_j(e_j)$, $\alpha_i(\overset{N_i}{\bullet} e_i) = \alpha_j(\overset{N_i}{\bullet} e_j)$, and $\alpha_i(e_i \overset{N_i}{\bullet}) = \alpha_j(e_j \overset{N_i}{\bullet})$.
2. For each $e \in E^\star$, we say $\text{eqvl}(e) = \{e' \in E^\star \mid e \bowtie e'\}$.
3. Let ν' be the set of equivalence classes of $\bowtie$, i.e. $\nu' = \{\text{eqvl}(e) \mid e \in E^\star\}$.

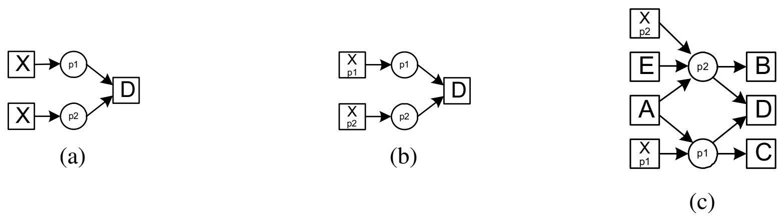


Fig. 5. The original and relabelled part of Figure 4(d) and a part of the aggregated net.

4. For all causal nets (C_i, E_i, K_i, S_i) and labelling functions α_i, define a labelling function $\beta'_i : E_i \to \nu'$ such that for an event e_i, $\beta'_i(e_i) = \text{eqvl}(e_i)$, i.e. it returns the equivalence class of $\bowtie$ containing e_i.

After re-labelling the events, the part of the run shown in Figure 4(d) is relabelled to include the pre- and post-conditions. Figure 5(a) shows the fragment before relabelling, whereas Figure 5(b) shows the fragment after relabelling. (We only show the relabelling with respect to the post-conditions.) Applying the ALK algorithm of Definition 4.1 to the relabelled runs yields the result as shown in Figure 5(c). Note that we do not show the ν' labels explicitly, i.e. B refers to the equivalence class of events labelled B.

What remains to be shown is that our algorithm does not only work for our small running example, but also in the general case. The only difference between the assumptions in Definition 4.1 and Definition 5.1 is the requirement with respect to transition equivalence. Therefore, if suffices to show that after applying the relabelling algorithm on a causal set, we can establish transition equivalence.

Property 5.2. (Transition equivalence holds after relabelling)
Let μ, ν be two disjoint sets of labels. Let $\Phi = \{(C_i, E_i, K_i, S_i) \mid 0 \leq i < n\}$ be a causal set, and let $\Psi = \{(\alpha_i : C_i \to \mu, \beta_i : E_i \to \nu) \mid 0 \leq i < n\}$ be a set of labelling functions in (C_i, E_i, K_i, S_i), such that $\alpha_i(S_i)$ is the same for all causal nets. After applying the relabelling algorithm, the property of transition equivalence holds for (Φ, Ψ'), with $\Psi' = \{(\alpha_i : C_i \to \mu, \beta'_i : E_i \to \nu') \mid 0 \leq i < n\}$, and β'_i as defined in Definition 5.1.

The algorithm presented above is capable of finding events that have the same label, but correspond to different transitions in the system net. When *no transition labels are known at all*, it can be applied to *find all transition labels*, by using an initial $\nu = \{\tau\}$ and initial mapping functions β_i, mapping everything onto τ. However, in that case, no distinction can be made between events that have the same pre- and post-set, but should have different labels. After applying this relabelling algorithm, the ALK algorithm of Section 4 can be used to find the system net belonging to the given causal nets.

6 Aggregation with Unknown Place Labels

In Section 5, we have shown a way to identify the transitions in a system net, based on the labels of events in causal nets. However, what if condition labels are not known? Notice that the difference to other approaches based on partial orders is that here we do

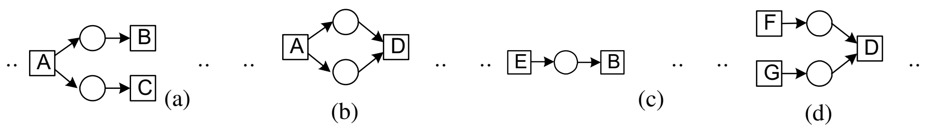


Fig. 6. Four examples of parts of runs.

know the conditions constituting the order between events but do *not* know which two conditions refer to a token in the same place of the P/T-net representing the process.

So, in this section, we take one step back. We assume all events to refer to the correct transition and try to identify the labels of conditions. We introduce an algorithm to aggregate causal nets to a system net, such that the original causal nets are indeed runs of that system net. In Figure 6, we again show our small example of the aggregation problem, only this time there are no labels for conditions $p1$ and $p2$, which we did have in Figures 2 and 4.

Consider the four runs of Figure 6. Remember that they are *parts* of causal nets, in such a way that the tasks A, B, C, D, E, F and G do not appear in any other way in another causal net. In contrast to the algorithms presented in previous sections, we cannot always derive a unique aggregated system net for causal nets if we do not have labels for the conditions. Instead, we define an *aggregation class*, describing a class of WF-nets that could have generated these causal nets. The following table shows some requirements all WF-nets in the aggregation class of our example should satisfy.

Fragment	Conclusions
Fig. 6 (a)	$A\bullet = \bullet B \uplus \bullet C$
Fig. 6 (b)	$A\bullet = \bullet D$
Fig. 6 (c)	$E\bullet = \bullet B$
Fig. 6 (d)	$F \bullet \uplus G\bullet = \bullet D$

This information is derived using the concept of a segment, which can be considered to be the context of a condition in a causal net.

Definition 6.1. (Segment)
Let $N = ((C, E, K), S_0)$ be a causal net and let $N' = (C', E_{in}, E_{out})$ be such that $C' \subseteq C$, $E_{in} \cup E_{out} \subseteq E$ and $E_{in} \neq \emptyset$ and $E_{out} \neq \emptyset$. We call N' a *segment* if:

– for all $c \in C'$ holds that $\bullet c \subseteq E_{in}$ and $c\bullet \subseteq E_{out}$, and
– for all $e \in E_{in}$ holds that $e\bullet \subseteq C'$, and
– for all $e \in E_{out}$ holds that $\bullet e \subseteq C'$, and
– the subgraph of N made up by $C' \cup E_{in} \cup E_{out}$ is connected.

We call the events in E_{in} the input events and the events in E_{out} the output events. Furthermore, a segment is called *minimal* if C' is minimal, i.e. if there does not exist a segment $N'' = (C'', E'_{in}, E'_{out})$ with $C'' \subset C'$ and $C'' \neq \emptyset$.

For the fragments of Figure 6, it is easy to see that each of them contains only one minimal segment, where the input events are the events on the left hand side and the output events are the events on the right hand side.

The meaning of a segment is as follows. If we have a run and a segment in that run, then we know that after each of the events in the input set of the segment occurred,

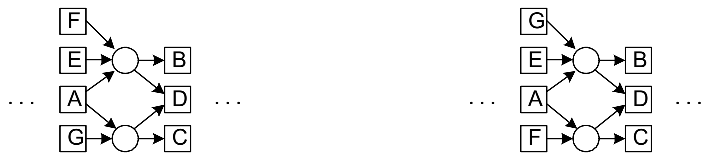


Fig. 7. Two aggregated nets.

all the events in the output set occurred in the execution represented by this run. This translates directly to a marking in a system net, since the occurrence of a set of transitions would lead to some marking (i.e. a bag over places), which enables another set of transitions. Furthermore, each transition only produces one token in each output place. Combining this leads to the fact that for each minimal segment in a causal net the bag of places following the transitions corresponding to the input events of the segment should be the same as the bag of places preceding the transitions corresponding to the output set of events.

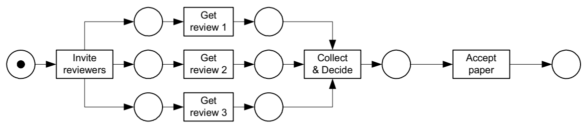
Clearly, when looking only at these fragments, what we are looking for are the places that should be put between tasks A, E, F and G on the one hand, and B, C and D on the other hand. Therefore, we only focus on this part of the causal nets. For this specific example, there are two possibilities, both of which are equally correct, namely the two WF-net fragments shown in Figure 7.

From the small example, we have seen that it is possible to take a set of causal nets without labels for any of the conditions (but with labels for all the events) and to define a class of WF-nets that could be system nets of the causal nets. In the remainder of this section, we show that this is indeed possible for all causal sets. For this, we first introduce the NCL algorithm.

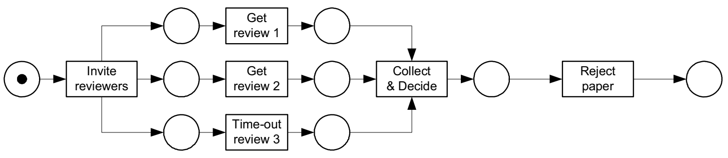
6.1 NCL Algorithm

Before presenting the NCL algorithm (which stands for "No Condition Labels"), we first take a look at a more intuitive example. Consider Figure 8, where we present three causal nets, each of which corresponds to a paper review process. In the first causal net, three reviewers are invited to review the paper and after the three reviews are received, the paper is accepted. In the second causal net, only two reviews are received (the third one is not received on time), but the paper is rejected nonetheless (apparently the two reviewers that replied rejected the paper). In the third example only one review is received in time, and therefore an additional reviewer is invited, which hands in his review in time, but does not accept the paper.

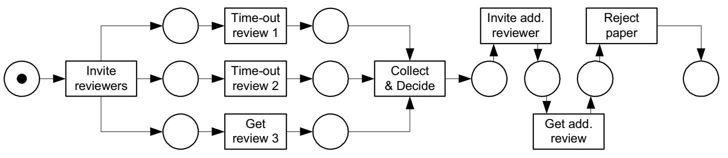
As we stated before, we define an aggregation class of a causal set that contains all WF-nets that are capable of generating the causal nets in the causal set. The information needed for this aggregation class comes directly from the causal nets, using minimal segments. In Table 1, we present the conclusions we can draw based on the three causal nets. In this table we consider *bags* of pre- and post-sets of transitions in the aggregation class. The information in this table is obtained from the causal nets in the following way. Consider for example Figure 8(a), where `invite reviewers` is followed by



(a)



(b)



(c)

Fig. 8. Three causal nets of a review process of a paper.

`Get review 1`, `Get review 2` and `Get review 3`. This implies that the bag of output places of `invite reviewers`, should be the same as the sum over the bags of the input places of `Get review 1`, `Get review 2` and `Get review 3`.

Definition 6.2. (Aggregation class)
Let $\Phi = \{(C_i, E_i, K_i, S_i) \mid 0 \le i < n\}$ be a causal set, and let $\sigma = (P, T, F, M_0)$ be a marked WF-net. For each causal net, let $\beta_i : E_i \to T$ be a mapping from the events of that causal net to T, such that β_i is a labelling function for (C_i, E_i, K_i, S_i). We define A_Φ, the *aggregation class* of Φ, as the set of all pairs $(\sigma, \mathcal{B})$ such that the following conditions are satisfied:

1. $T = \bigcup_{0 \le i < n} \text{rng}(\beta_i)$ is the set of transitions, i.e. each transition appears as an event at least once in some causal net,
2. $\mathcal{B}$ is the set of all labelling functions, i.e. $\mathcal{B} = \{\beta_i \mid 0 \le i < n\}$. We use $\beta_i \in \mathcal{B}$ to denote the labelling function for events belonging to $(C_i, E_i, K_i, S_i) \in \Phi$,
3. For all $p \in P$ holds that $\overset{\sigma}{\bullet}p \cup p\overset{\sigma}{\bullet} \neq \emptyset$,
4. $M_0 = [p_{ini}]$ and $\overset{\sigma}{\bullet}p_{ini} = \emptyset$,
5. For each causal net $\gamma = (C_i, E_i, K_i, S_i)$, with $e \in E_i$ and $\beta_i(e) = t$ holds that if $S_i(\overset{\gamma}{\bullet}e) = 1$ then $p_{ini} \in \overset{\sigma}{\bullet}t$,
6. For each causal net $\gamma = (C_i, E_i, K_i, S_i)$, with $e \in E_i$ and $\beta_i(e) = t$ holds that $|t\overset{\sigma}{\bullet}| = |e\overset{\gamma}{\bullet}|$ and $|\overset{\sigma}{\bullet}t| = |\overset{\gamma}{\bullet}e|$,
7. For each causal net $\gamma = (C_i, E_i, K_i, S_i)$, with $e \in E_i$, $\beta_i(e) = t$ and $T' \subseteq T$ holds that $|t\overset{\sigma}{\bullet} \cap \bigcup_{t' \in T'}(\overset{\sigma}{\bullet}t')| \ge \sum_{e' \in E_i, \beta(e') \in T'} |e\overset{\gamma}{\bullet} \cap \overset{\gamma}{\bullet}e'|$,

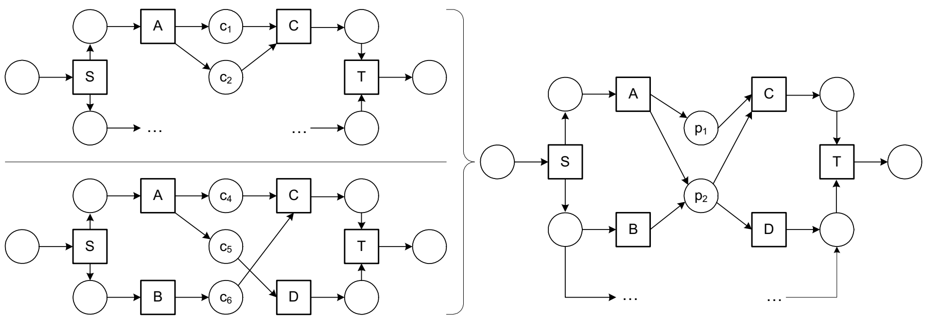


Fig. 9. Example explaining the use of bags.

8. For each causal net $\gamma = (C_i, E_i, K_i, S_i)$, with $e \in E_i$, $\beta_i(e) = t$ and $T' \subseteq T$ holds that $|\bigcup_{t' \in T'} (t' \overset{\gamma}{\bullet}) \cap \overset{\gamma}{\bullet} t| \geq \sum_{e' \in E_i, \beta(e') \in T'} |e' \overset{\gamma}{\bullet} \cap \overset{\gamma}{\bullet} e|$,
9. For each causal net $\gamma = (C_i, E_i, K_i, S_i)$ and any minimal segment (C'_i, E_{in}, E_{out}) of γ, holds that $\biguplus_{e \in E_{in}} \left(\beta_i(e) \overset{\gamma}{\bullet}\right) = \biguplus_{e \in E_{out}} \left(\overset{\gamma}{\bullet} \beta_i(e)\right)$.

Figure 9 is used to gain more insight into part 9 of Definition 6.2. In the lower causal net of that figure, there is a token travelling from A to D and from B to C. The upper causal net only connects A and C. Assuming that these are the only causal nets in which these transitions appear, we know that the conditions between A and D and between B and C should represent a token in the same place, since there is a minimal segment $(\{c_4, c_5, c_6\}, \{A, B\}, \{C, D\})$ in the lower causal net and therefore, $A \bullet \uplus B \bullet = \bullet C \uplus \bullet D = [p_1, 2p_2]$.

The NLC algorithm takes a set of runs without condition labels as a starting point. From these runs, an aggregation class of WF-nets is defined. If the runs were generated from some sound WF-net, then the WF-net itself is in that aggregation class. We conclude this section with an elaborate example of the application of the NCL algorithm.

Consider the four causal nets presented in Figure 10. These causal nets originate from a workflow system in which two activities need to be performed. These activities are labelled `L` and `R`. However, in the workflow design, there are several options. First, the system initializes the two activities trough event `Init LR`. Then, a person can decide to perform both activities at once, which is represented by the event `Do LR`. When both activities have been performed, the workflow can be finished through event `exit LR`. However, in a typical workflow environment, people can make mistakes and therefore, in Figure 10(b), both activities have been undone, thus generating events `Undo L` and `Undo R`. Finally, in Figure 10 (c) and (d) it is shown that the workflow system allows for the two activities to be executed separately, through `Do L` and `Do R`. To keep things interesting, the last causal net belongs to a case in the workflow system that is not finished yet. However, this set of causal nets still conforms to the definition of a causal set (i.e. Definition 3.3).

Using the NCL algorithm given by Definition 6.2, we can generate the aggregation class of the causal set of Figure 10. In fact, this aggregation class only contains one workflow net, namely the workflow net shown in Figure 11.

The workflow net in Figure 11 actually allows for more behaviour than is shown in the four causal nets of Figure 10. It is for example possible to execute a long sequence

Table 1. Information derived from review example.

Causal net	Conclusions on transitions in the aggregation class
Fig. 8 (a)	$\bullet$“Invite reviewers” $= [p_{ini}]$
	“Invite reviewers”$\bullet$ $=$ $\bullet$“Get review 1” $\uplus$ $\bullet$“Get review 2” $\uplus$ $\bullet$“Get review 3”
	“Get review 1”$\bullet$ $\uplus$ “Get review 2”$\bullet$ $\uplus$ “Get review 3”$\bullet$ $=$ $\bullet$“Collect & Decide”
	“Collect & Decide”$\bullet$ $=$ $\bullet$“Accept paper”
	$\lvert$“Accept paper”$\bullet\rvert$ $= 1$
Fig. 8 (b)	$\bullet$“Invite reviewers” $= [p_{ini}]$
	“Invite reviewers”$\bullet$ $=$ $\bullet$“Get review 1” $\uplus$ $\bullet$“Get review 2” $\uplus$ $\bullet$“Time-out review 3”
	“Get review 1”$\bullet$ $\uplus$ “Get review 2”$\bullet$ $\uplus$ “Time-out review 3”$\bullet$ $=$ $\bullet$“Collect & Decide”
	“Collect & Decide”$\bullet$ $=$ $\bullet$“Reject paper”
	$\lvert$“Reject paper”$\bullet\rvert$ $= 1$
Fig. 8 (c)	$\bullet$“Invite reviewers” $= [p_{ini}]$
	“Invite reviewers”$\bullet$ $=$ $\bullet$“Time-out review 1” $\uplus$ $\bullet$“Time-out review 2” $\uplus$ $\bullet$“Get review 3”
	“Time-out review 1”$\bullet$ $\uplus$ “Time-out review 2”$\bullet$ $\uplus$ “Get review 3”$\bullet$ $=$ $\bullet$“Collect & Decide”
	“Collect & Decide”$\bullet$ $=$ $\bullet$“Invite add. reviewer”
	“Invite add. reviewer”$\bullet$ $=$ $\bullet$“Get add. review”
	“Get add. review”$\bullet$ $=$ $\bullet$“Reject paper”
	$\lvert$“Reject paper”$\bullet\rvert$ $= 1$

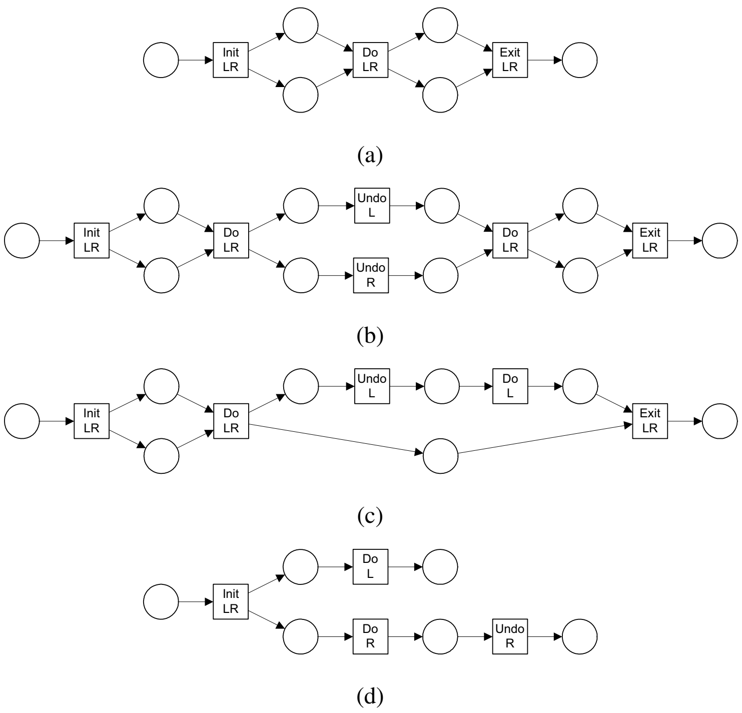


Fig. 10. Four causal nets without condition labels.

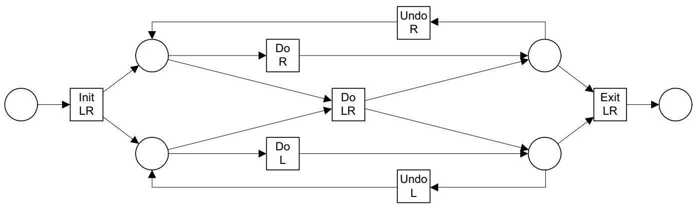


Fig. 11. The only element of the aggregation class of the four nets of Figure 10.

of `Do L` and `Undo L`, which is not shown in the causal nets. Therefore, this example once more shows that each net in the aggregation class can actually generate the runs of the causal set it was constructed from, but it might be able to generate more runs.

7 Conclusion and Future Work

In this paper, we looked at process mining from a new perspective. Instead of starting with a set of traces, we started with runs, which constitute partial orders on events. We presented three algorithms to generate a Petri net from these runs. The first algorithm assumes that, for each run, all labels of both conditions and events are known. The second algorithm relaxes this by assuming that some transitions can have the same label (i.e. duplicate labels are allowed in the system net). This algorithm can also be used if only condition/place-labels were recorded. Finally, we provided an algorithm that does not require condition labels, i.e. the event/transition labels are known, the condition/place labels are unknown and duplicate transition labels are not allowed.

The results presented in this paper hold for a subclass of Petri nets, so-called WF-nets. However, the first two algorithms presented here can easily be generalized to be applicable to any Petri net. For the third algorithm this can also be done, however, explicit knowledge about the initial marking would be required. When taking a set of runs as a starting point, this knowledge is not present in the general case.

References

1. Desel, J.: Validation of Process Models by Construction of Process Nets. In Aalst, W., Desel, J., Oberweis, A., eds.: Business Process Management: Models, Techniques, and Empirical Studies. Volume 1806 of Lecture Notes in Computer Science, Springer-Verlag, Berlin (2000) 110–128
2. Aalst, W., van Dongen, B., Herbst, J., Maruster, L., Schimm, G., Weijters, A.: Workflow Mining: A Survey of Issues and Approaches. Data and Knowledge Engineering **47**(2) (2003) 237–267
3. Smith, E.: On net systems generated by process foldings. In Rozenberg, G., ed.: Advances in Petri Nets. Volume 524 of Lecture Notes in Computer Science, Springer-Verlag, Berlin (1991) 253–276
4. Desel, J., Erwin, T.: Hyprid specifications: looking at workflows from a run-time perspective. Computer Systems Science and Engineering **5** (2000) 291–302
5. Roychoudhury, A., Thiagarajan, P.: Communicating transaction processes. In Lilius, J., Balarin, F., Machado, R., eds.: Proceedings of Third International Conference on Application of Concurrency to System Design (ACSD2003), IEEE Computer Society (2003) 157–166
6. Harel, D., Kugler, H., Pnueli, A.: Synthesis revisited: Generating statechart models from scenario-based requirements. In Kreowski, H.J., Montanari, U., Orejas, F., Rozenberg, G., Taentzer, G., eds.: Formal Methods in Software and Systems Modeling. Volume 3393 of Lecture Notes in Computer Science, Springer (2005) 309–324
7. Lorenz, R., Juhás, G.: Towards synthesis of petri nets from scenarios. In Donatelli, S., Thiagarajan, P.S., eds.: ICATPN. Volume 4024 of Lecture Notes in Computer Science, Springer (2006) 302–321
8. Dongen, B., Aalst, W.: Multi-phase Process mining: Building Instance Graphs. In Atzeni, P., Chu, W., Lu, H., Zhou, S., Ling, T., eds.: Conceptual Modeling - ER 2004. Volume 3288 of Lecture Notes in Computer Science, Springer-Verlag, Berlin (2004) 362–376
9. Dongen, B., Aalst, W.: Multi-phase Process mining: Aggregating Instance Graphs into EPCs and Petri Nets. In: PNCWB 2005 workshop. (2005) 35–58
10. Aalst, W., de Medeiros, A.A., Weijters, A.: Genetic Process Mining. In: Proceedings of the 26th International Conference on Applications and Theory of Petri Nets. Volume 3536 of Lecture Notes in Computer Science (2005)
11. de Medeiros, A.A., Weijters, A., van der Aalst, W.: Genetic Process Mining: A Basic Approach and its Challenges. In: BPM - BPI workshop. (2005)
12. Herbst, J., Karagiannis, D.: Workflow mining with InWoLvE. Comput. Ind. **53**(3) (2004) 245–264