Computing the Concurrency Threshold of Sound Free-Choice Workflow Nets

Philipp J. Meyer[1(✉)], Javier Esparza[1], and Hagen Völzer[2]

[1] Technical University of Munich, Munich, Germany
{meyerphi,esparza}@in.tum.de
[2] IBM Research, Zurich, Switzerland
hvo@zurich.ibm.com

Abstract. Workflow graphs extend classical flow charts with concurrent fork and join nodes. They constitute the core of business processing languages such as BPMN or UML Activity Diagrams. The activities of a workflow graph are executed by humans or machines, generically called resources. If concurrent activities cannot be executed in parallel by lack of resources, the time needed to execute the workflow increases. We study the problem of computing the minimal number of resources necessary to fully exploit the concurrency of a given workflow, and execute it as fast as possible (i.e., as fast as with unlimited resources).

We model this problem using free-choice Petri nets, which are known to be equivalent to workflow graphs. We analyze the computational complexity of two versions of the problem: computing the resource and concurrency thresholds. We use the results to design an algorithm to approximate the concurrency threshold, and evaluate it on a benchmark suite of 642 industrial examples. We show that it performs very well in practice: It always provides the exact value, and never takes more than 30 ms for any workflow, even for those with a huge number of reachable markings.

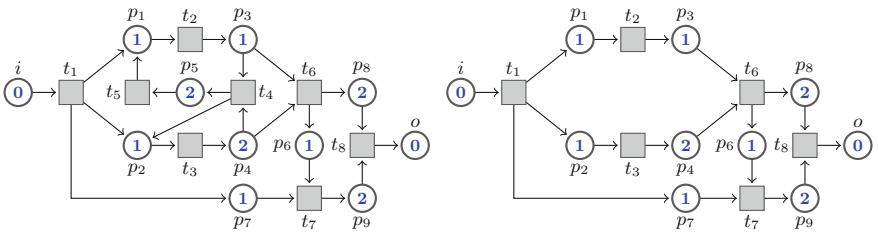
1 Introduction

A *workflow graph* is a classical control-flow graph (or flow chart) extended with concurrent fork and join. Workflow graphs represent the core of workflow languages such as BPMN (Business Process Model and Notation), EPC (Event-driven Process Chain), or UML Activity Diagrams.

In many applications, the activities of an execution workflow graph have to be carried out by a fixed number of *resources* (for example, a fixed number of computer cores). Increasing the number of cores can reduce the minimal runtime of the workflow. For example, consider a simple deterministic workflow (a workflow without choice or merge nodes), which forks into k parallel activities, all of duration 1, and terminates after a join. With an optimal assignment of resources to activities, the workflow takes time k when executed with one resource, time $\lceil k/2 \rceil$ with two resources, and time 1 with k resources; additional resources

D. Beyer and M. Huisman (Eds.): TACAS 2018, LNCS 10806, pp. 3–19, 2018.
https://doi.org/10.1007/978-3-319-89963-3_1

bring no further reduction. We call k the *resource threshold*. In a deterministic workflow that forks into two parallel chains of k sequential activities each, one resource leads to runtime $2k$, and two resources to runtime k. More resources do not improve the runtime, and so the resource threshold is 2. Clearly, the resource threshold of a deterministic workflow with k activities is a number between 1 and k. Determining this number can be seen as a scheduling problem. However, most scheduling problems assume a fixed number of resources and study how to optimize the makespan [11,17], while we study how to minimize the number of resources. Other works on resource/machine minimization [5,6] consider interval constraints instead of the partial-order constraints given by a workflow graph.



(a) Sound free-choice workflow net N (b) A run of N

Fig. 1. A sound free-choice workflow net and one of its runs (Color figure online)

Following previous work, we do not directly work with workflow graphs, but with their equivalent representation as *free-choice workflow Petri nets*, which has been shown to be essentially the same model [10] and allows us to directly use a wealth of results of free-choice Petri nets [7]. Figure 1(a) shows a free-choice workflow net. The actual workflow activities, also called *tasks*, which need a resource to execute and which consume time are modeled as the places of the net: Each place p of the net is assigned a time $\tau(p)$, depicted in blue. Intuitively, when a token arrives in p, it must execute a task that takes $\tau(p)$ time units before it can be used to fire a transition. A free choice exists between transitions t_4 and t_6, which is a representation of a choice node (if-then-else or loop condition) in the workflow.

If no choice is present or all choices are resolved, we have a deterministic workflow such as the one in Fig. 1(b). In Petri net terminology, deterministic workflows correspond to the class of marked graphs. Deterministic workflows are common in practice: in the standard suite of 642 industrial workflows that we use for experiments, 63.7% are deterministic. We show that already for this restricted class, deciding if the threshold exceeds a given bound is NP-hard. Therefore, we investigate an over-approximation of the resource threshold, already introduced in [4]: the *concurrency threshold*. This is the maximal number of task places that can be simultaneously marked at a reachable marking. Clearly, if a workflow with concurrency threshold k is executed with k resources, then we can always start the task of a place immediately after a token arrives, and this schedule already

achieves the fastest runtime achievable with unlimited resources. We show that the concurrency threshold can be computed in polynomial time for deterministic workflows.

For workflows with nondeterministic choice, corresponding to free-choice nets, we show that computing the concurrency threshold of free-choice workflow nets is NP-hard, solving a problem left open in [4]. We even prove that the problem remains NP-hard for sound free-choice workflows. Soundness is the dominant behavioral correctness notion for workflows, which rules out basic control-flow errors such as deadlocks. NP-hardness in the sound case is remarkable, because many analysis problems that have high complexity in the unsound case can be solved in polynomial time in the sound case (see e.g. [1,7,8]).

After our complexity analysis, we design an algorithm to compute bounds on the concurrency threshold using a combination of linear optimization and state-space exploration. We evaluate it on a benchmark suite of 642 sound free-choice workflow nets from an industrial source (IBM) [9]. The bounds can be computed in a total of 7 s (over all 642 nets). On the contrary, the computation of the exact value by state-space exploration techniques times out for the three largest nets, and takes 7 min for the rest. (Observe that partial-order reduction techniques cannot be used, because one may then miss the interleaving realizing the concurrency threshold.)

The paper is structured as follows. Section 2 contains preliminaries. Sections 3 and 4 study the resource and concurrency thresholds, respectively. Section 5 presents our algorithms for computing the concurrency bound, and experimental results. Finally, Sect. 6 contains conclusions.

2 Preliminaries

Petri Nets. A *Petri net* N is a tuple (P, T, F) where P is a finite set of places, T is a finite set of transitions ($P \cap T = \emptyset$), and $F \subseteq (P \times T) \cup (T \times P)$ is a set of arcs. The *preset* of $x \in P \cup T$ is ${}^\bullet x \stackrel{\text{def}}{=} \{y \mid (y, x) \in F\}$ and its *postset* is $x^\bullet \stackrel{\text{def}}{=} \{y \mid (x, y) \in F\}$. We extend the definition of presets and postsets to sets of places and transitions $X \subseteq P \cup T$ by ${}^\bullet X \stackrel{\text{def}}{=} \bigcup_{x \in X} {}^\bullet x$ and $X^\bullet \stackrel{\text{def}}{=} \bigcup_{x \in X} x^\bullet$. A net is *acyclic* if the relation F^* is a partial order, denoted by $\preceq$ and called the *causal order*. A node x of an acyclic net is *causally maximal* if no node y satisfies $x \prec y$.

A *marking* of a Petri net is a function $M \colon P \to \mathbb{N}$, representing the number of tokens in each place. For a set of places $S \subseteq P$, we define $M(S) \stackrel{\text{def}}{=} \sum_{p \in S} M(p)$. Further, for a set of places $S \subseteq P$, we define by M_S the marking with $M_S(p) = 1$ for $p \in S$ and $M_S(p) = 0$ for $p \notin S$.

A transition t is *enabled* at a marking M if for all $p \in {}^\bullet t$, we have $M(p) \geq 1$. If t is enabled at M, it may *occur*, leading to a marking M' obtained by removing one token from each place of ${}^\bullet t$ and then adding one token to each place of $t^\bullet$. We denote this by $M \xrightarrow{t} M'$. Let $\sigma = t_1 t_2 \dots t_n$ be a sequence of transitions. For a marking M_0, σ is an *occurrence sequence* if $M_0 \xrightarrow{t_1} M_1 \xrightarrow{t_2} \dots \xrightarrow{t_n} M_n$ for some markings $M_1, \dots, M_n$. We say that M_n is reachable from M_0 by σ and

denote this by $M_0 \xrightarrow{\sigma} M_n$. The set of all markings reachable from M in N by some occurrence sequence σ is denoted by $\mathcal{R}^N(M)$. A *system* is a pair (N, M) of a Petri net N and a marking M. A system (N, M) is *live* if for every $M' \in \mathcal{R}^N(M)$ and every transition t some marking $M'' \in \mathcal{R}^N(M')$ enables t. The system is *1-safe* if $M'(p) \leq 1$ for every $M' \in \mathcal{R}^N(M)$ and every place $p \in P$.

Convention: Throughout this paper we assume that systems are 1-safe, i.e., we identify "system" and "1-safe system".

Net Classes. A net $N = (P, T, F)$ is a *marked graph* if $|{}^\bullet p| \leq 1$ and $|p^\bullet| \leq 1$ for every place $p \in P$, and a *free-choice net* if for any two places $p_1, p_2 \in P$ either $p_1^\bullet \cap p_2^\bullet = \emptyset$ or $p_1^\bullet = p_2^\bullet$.

Non-sequential Processes of Petri Nets. An (A, B)-*labeled Petri net* is a tuple $N = (P, T, F, \lambda, \mu)$, where $\lambda\colon P \to A$ and $\mu\colon T \to B$ are *labeling functions* over alphabets A, B. The nonsequential processes of a 1-safe system (N, M) are acyclic, (P, T)-labeled marked graphs. Say that a set P'' of places of a (P, T)-labeled acyclic net *enables* $t \in T$ if all the places of P'' are causally maximal, carry pairwise distinct labels, and $\lambda(P'') = {}^\bullet t$.

Definition 1. *Let $N = (P, T, F)$ be a Petri net and let M be a marking of N. The set $\mathcal{NP}(N, M)$ of* nonsequential processes *of (N, M) (*processes *for short) is the set of (P, T)-labeled Petri nets defined inductively as follows:*

- *The (P, T)-labeled Petri net containing for each place $p \in P$ marked at M one place $\widehat{p}$ labeled by p, no other places, and no transitions, belongs to $\mathcal{NP}(N, M)$.*
- *If $\Pi = (P', T', F', \lambda, \mu) \in \mathcal{NP}(N, M)$ and $P'' \subseteq P'$ enables some transition t of N, then the (P, T)-labeled net $\Pi_t = (P' \uplus \widehat{P}, T' \uplus \{\widehat{t}\}, F' \uplus \widehat{F}, \lambda \uplus \widehat{\lambda}, \mu \uplus \widehat{\mu})$, where*
  - *$\widehat{P} = \{\widehat{p} \mid p \in t^\bullet\}$, with $\widehat{\lambda}(\widehat{p}) = p$, and $\widehat{\mu}(\widehat{t}) = t$;*
  - *$\widehat{F} = \{(p'', \widehat{t}) \mid p'' \in P''\} \cup \{(\widehat{t}, \widehat{p}) \mid \widehat{p} \in \widehat{P}\}$;*

  also belongs to $\mathcal{NP}(N, M)$. We say that Π_t extends *Π.*

We denote the minimal and maximal places of a process Π w.r.t. the causal order by $\min(\Pi)$ *and* $\max(\Pi)$*, respectively.*

As usual, we say that two processes are *isomorphic* if they are the same up to renaming of the places and transitions (notice that we rename only the names of the places and transitions, not their labels).

Figure 2 shows two processes of the workflow net in Fig. 1(a). (The figure does not show the names of places and transitions, only their labels.) The net containing the white and grey nodes only is already a process, and the grey places are causally maximal places that enable t_6. Therefore, according to the definition we can extend the process with the green nodes to produce another process. On the right we extend the same process in a different way, with the transition t_4.

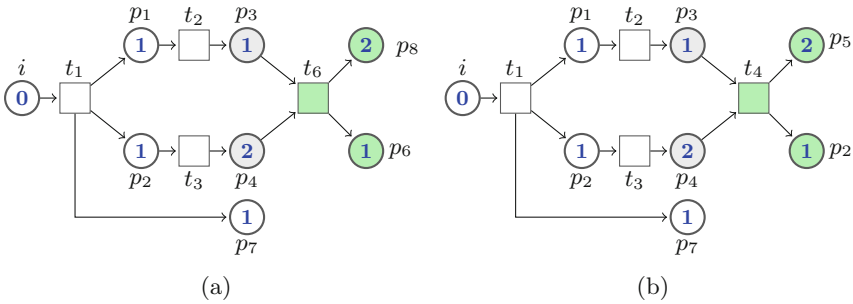


Fig. 2. Nonsequential processes of the net of Fig. 1(a) (Color figure online)

The following is well known. Let $(P', T', F', \lambda, \mu)$ be a process of (N, M):

- For every linearization $\sigma = t'_1 \dots t'_n$ of T' respecting the causal order $\preceq$, the sequence $\mu(\sigma) = \mu(t'_1) \dots \mu(t'_n)$ is a firing sequence of (N, M). Further, all these firing sequences lead to the same marking. We call it the *final marking* of Π, and say that Π leads from M to its final marking.

  For example, in Fig. 2 the sequences of the right process labeled by $t_1t_2t_3t_4$ and $t_1t_3t_2t_4$ are firing sequences leading to the marking $M = \{p_2, p_5, p_7\}$.
- For every firing sequence $t_1 \cdots t_n$ of (N, M) there is a process $(P', T', F', \lambda, \mu)$ such that $T' = \{t'_1, \dots, t'_n\}$, $\mu(t'_i) = t_i$ for every $1 \leq i \leq n$, and $\mu(t'_i) \preceq \mu(t'_j)$ implies $i \leq j$.

Workflow Nets. We slightly generalize the definition of workflow net as presented in e.g. [1] by allowing multiple initial and final places. A *workflow* net is a Petri net with two distinguished sets I and O of *input places* and *output places* such that (a) ${}^\bullet I = \emptyset = O^\bullet$ and (b) for all $x \in P \cup T$, there exists a path from some $i \in I$ to some $o \in O$ passing through x. The markings M_I and M_O are called initial and final markings of N. A workflow net N is *sound* if

- $\forall M \in \mathcal{R}^N(M_I) : M_O \in \mathcal{R}^N(M)$,
- $\forall M \in \mathcal{R}^N(M_I) : (M(O) \geq |O|) \Rightarrow (M = M_O)$, and
- $\forall t \in T : \exists M \in \mathcal{R}^N(M_I) : t$ is enabled at M.

It is well-known that every sound free-choice workflow net is a 1-safe system with the initial marking M_I [2,7]. Given a workflow net according to this definition one can construct another one with one single input place i and output place o and two transitions t_i, t_o with ${}^\bullet t_i = \{i\}, t_i^\bullet = I$ and ${}^\bullet t_o = O, t_o^\bullet = \{o\}$. For all purposes of this paper these two workflow nets are equivalent.

Given a workflow net N, we say that a process Π of (N, M_I) is a *run* if it leads to M_O. For example, the net in Fig. 1(b) is a run of the net in Fig. 1(a).

Petri Nets with Task Durations. We consider Petri nets in which, intuitively, when a token arrives in a place p it has to execute a task taking $\tau(p)$ time units before the token can be used to fire any transition. Formally, we consider tuples $N = (P, T, F, \tau)$ where (P, T, F) is a net and $\tau \colon P \to \mathbb{N}$.

Definition 2. *Given a nonsequential process $\Pi = (P', T', F', \lambda, \mu)$ of (N, M), a time bound t, and a number of resources k, we say that Π* is executable within time t with k resources *if there is a function $f\colon P' \to \mathbb{N}$ such that*

(1) for every $p'_1, p'_2 \in P'$: if $p'_1 \prec p'_2$ then $f(p'_1) + \tau(\lambda(p'_1)) \leq f(p'_2)$;
(2) for every $p' \in P'$: $f(p') + \tau(\lambda(p')) \leq t$; and
(3) for every $0 \leq u < t$ there are at most k places $p' \in P'$ such that $f(p') \leq u < f(p') + \tau(p')$.

We call a function f satisfying (1) a schedule*, a function satisfying (1) and (2) a t -*schedule*, and a function satsifying (1)–(3) a (k, t)-*schedule *of Π.*

Intuitively, $f(p')$ describes the starting time of the task executed at p'. Condition (1) states that if $p'_1 \preceq p'_2$, then the task associated to p'_2 can only start after the task for p'_1 has ended; condition (2) states that all tasks are done by time t, and condition (3) that at any moment in time at most k tasks are being executed. As an example, the process in Fig. 1(b) can be executed with two resources in time 6 with the schedule $i, p_1, p_2 \mapsto 0$; $p_3, p_4 \mapsto 1$; $p_7, p_6 \mapsto 3$, and $p_8, p_9 \mapsto 4$.

Given a process $\Pi = (P', T', F', \lambda, \mu)$ of (N, M) we define the schedule $f_{\min}$ as follows: if $p' \in \min(\Pi)$ then $f_{\min}(p') = 0$, otherwise define $f_{\min}(p') = \max\{f_{\min}(p'') + \tau(\lambda(p'')) \mid p'' \preceq p'\}$. Further, we define the *minimal execution time* $t_{\min}(\Pi) = \max\{f(p') + \tau(\lambda(p'')) \mid p' \in \max(\Pi)\}$. In the process in Fig. 1(b), the schedule $f_{\min}$ is the function that assigns $i, p_1, p_2, p_7 \mapsto 0$, $p_3, p_4 \mapsto 1$, $p_6, p_8 \mapsto 3$, $p_9 \mapsto 4$, and $o \mapsto 6$, and so $t_{\min}(\Pi) = 6$. We have:

Lemma 1. *A process $\Pi = (P', T', F', \lambda, \mu)$ of (N, M) can be executed within time $t_{\min}(\Pi)$ with $|P'|$ resources, and cannot be executed faster with any number of resources.*

Proof. For $k \geq |P'|$ resources condition (3) of Definition 2 holds vacuously. Π is executable within time t iff conditions (1) and (2) hold. Since $f_{\min}$ satisfies (1) and (2) for $t = t_{\min}(\Pi)$, Π can be executed within time $t_{\min}(\Pi)$. Further, $t_{\min}(\Pi)$ is the smallest time for which (1) and (2) can hold, and so Π cannot be executed faster with any number of resources.

3 Resource Threshold

We define the resource threshold of a run of a workflow net, and of the net itself. Intuitively, the resource threshold of a run is the minimal number of resources that allows one to execute it as fast as with unlimited resources, and the resource threshold of a workflow net is the minimal number of resources that allows one to execute *every run* as fast as with unlimited resources.

Definition 3. *Let N be a workflow net, and let Π be a run of N. The* resource threshold of Π*, denoted by $RT(\Pi)$ is the smallest number k such that Π can be executed in time $t_{\min}(\Pi)$ with k resources. A schedule of Π* realizes *the resource threshold if it is a $(RT(\Pi), t_{\min}(\Pi))$-schedule.*

The resource threshold *of N, denoted by $RT(N)$, is defined by $RT(N) = \max\{RT(\Pi) \mid \Pi$ is a run of $(N, M_I)\}$. A* schedule of *N is a function that assigns to every process $\Pi \in \mathcal{NP}(N, M)$ a schedule of Π. A schedule of N is a (k, t)-schedule if it assigns to every run Π a (k, t)-schedule of Π. A schedule of N* realizes *the resource threshold if it assigns to every run Π a $(RT(N), t_{\min}(\Pi))$-schedule.*

Example 1. We have seen in the previous section that for the process in Fig. 1(b) we have $t_{\min}(\Pi) = 6$, and a schedule with two resources already achieves this time. So the resource bound is 2. The workflow net of Fig. 1 has infinitely many runs, in which loosely speaking, the net executes t_4 arbitrarily many times, until it "exits the loop" by choosing t_6, followed by t_7 and t_8. It can be shown that all processes have resource threshold 2, and so that is also the resource threshold of the net.

In the rest of the section we obtain two negative results about the result threshold. First, it is difficult to compute: Determining if the resource threshold exceeds a given threshold is NP-complete even for acyclic marked graphs, a very simple class of workflows. Second, we show that even for acyclic free-choice workflow nets the resource threshold may not be realized by any online scheduler.

3.1 Resource Threshold Is NP-complete for Acyclic Marked Graphs

We prove that deciding if the resource threshold exceeds a given bound is NP-complete even for acyclic sound marked graphs. The proof proceeds by reduction from the following classical scheduling problem, proved NP-complete in [18]:

> **Given**: a finite, partially ordered set of jobs with non-negative integer durations, and non-negative integers t and k.
> **Decide**: Can all jobs can be executed with k machines within t time units in a way that respects the given partial order, i.e., a job is started only after all its predecessors have been finished?

More formally, the problem is defined as follows: Given jobs $\mathcal{J} = \{J_1, \ldots, J_n\}$, where J_i has duration $\tau(J_i)$ for every $1 \leq i \leq n$, and a partial order $\preceq$ on $\mathcal{J}$, does there exist a function $f \colon \mathcal{J} \to \mathbb{N}$ such that

(1) for every $1 \leq i, j \leq n$: if $J_i \prec J_j$ then $f(J_i) + \tau(J_i) \leq f(J_j)$;
(2) for every $1 \leq i \leq n$: $f(J_i) + \tau(J_i) \leq t$; and
(3) for every $0 \leq u < t$ there are at most k indices i such that $f(J_i) \leq u < f(J_i) + \tau(J_i)$.

These conditions are almost identical to the ones we used to define if a nonsequential process can be executed within time t with k resources. We exploit this to construct an acyclic workflow marked graph that "simulates" the scheduling problem. For the detailed proof, we refer to the full version of this paper [15].

Theorem 1. *The following problem is NP-complete:*

> **Given:** *An acyclic, sound workflow marked graph N, and a number k.*
> **Decide:** *Does $RT(N) \leq k$ hold?*

3.2 Acyclic Free-Choice Workflow Nets May Have no Optimal Online Schedulers

A resource threshold of k guarantees that every run *can* be executed without penalty with k resources. In other words, *there exists* a schedule that achieves optimal runtime. However, in many applications the schedule must be determined at runtime, that is, the resources must be allocated without knowing how choices will be resolved in the future. In order to formalize this idea we define the notion of an *online schedule* of a workflow net N.

Definition 4. *Let N be a Petri net, and let Π and Π' be two processes of (N, M). We say that Π is a* prefix *of Π', denoted by $\Pi \lhd \Pi'$, if there is a sequence $\Pi_1, \ldots, \Pi_n$ of processes such that $\Pi_1 = \Pi$, $\Pi_n = \Pi'$, and Π_{i+1} extends Π_i by one transition for every $1 \leq i \leq n-1$.*

Let f be a schedule of (N, M), i.e., a function assigning a schedule to each process. We say that f is an online schedule *if for every two runs Π_1, Π_2, and for every two prefixes $\Pi_1' \lhd \Pi_1$ and $\Pi_2' \lhd \Pi_2$: If Π_1' and Π_2' are isomorphic, then $f(\Pi_1') = f(\Pi_2')$.*

Intuitively, if Π_1' and Π_2' are isomorphic then they are the same process Π, which in the future can be extended to either Π_1 or Π_2, depending on which transitions occur. In an online schedule, Π is scheduled in the same way, independently of whether it will become Π_1 or Π_2 in the future. We show that even for acyclic free-choice workflow nets there may be no online schedule that realizes the resource threshold. That is, even though for every run it is possible to schedule the tasks with $RT(N)$ resources to achieve optimal runtime, this requires knowing how it will evolve before the execution of the workflow.

Proposition 1. *There is an acyclic, sound free-choice workflow net for which no online schedule realizes the resource threshold.*

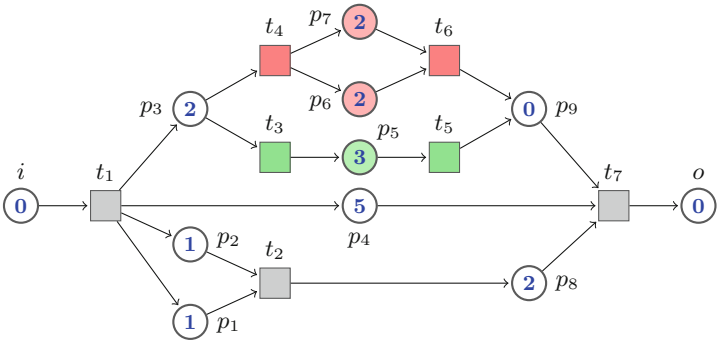


Fig. 3. A workflow net with two runs. No online scheduler for three resources achieves the minimal runtime in both runs. (Color figure online)

Proof. Consider the sound free-choice workflow net (N, M_I) of Fig. 3. It has two runs: Π_g, which executes the grey and green transitions, and Π_r, which executes the grey and red transitions. Their resource thresholds are $RT(\Pi_g) = RT(\Pi_r) = 3$, realized by the schedules f_g and f_r in Fig. 4:

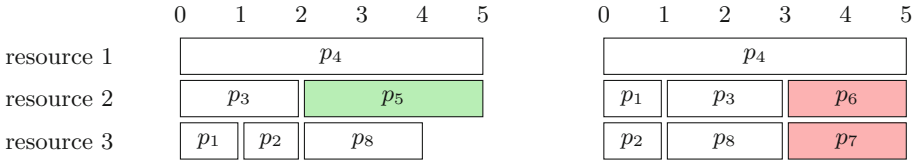


Fig. 4. Schedules f_g and f_r for the two runs Π_g and Π_r of the net of Fig. 3.

Indeed, observe that f_g and f_r execute Π_g and Π_r within time 5, and even with unlimited resources no schedule can be faster because of the task p_4, while two or fewer resources are insufficient to execute either run within time 5.

The schedule of (N, M_I) that assigns f_g and f_r to Π_g and Π_r is not an online schedule. Indeed, the process containing one single transition labeled by t_1 and places labeled by i, p_1, p_2, p_3 is isomorphic to prefixes of Π_g and Π_r. However, we have $f_g(p_3) = 0 \neq 1 = f_r(p_3)$. We now claim:

(a) Every schedule f_g of Π_g that realizes the resource threshold (time 5 with 3 resources) satisfies $f_g(p_3) = 0$.

Indeed, if $f_g(p_3) \geq 1$, then $f_g(p_5) \geq 3$, $f_g(p_9) \geq 6$, and finally $f_g(o) \geq 6$, so f_g does not meet the time bound.

(b) Every schedule f_r of Π_r that realizes the resource threshold (time 5 with 3 resources) satisfies $f_r(p_3) > 0$.

Observe first that we necessarily have $f_r(p_4) = 0$, and so a resource, say R_1, is bound to p_4 during the complete execution of the workflow, leaving two resources left. Assume $f_r(p_3) = 0$, i.e., a second resource, say R_2, is bound to p_3 at time 0, leaving one resource left, say R_3. Since both p_1 and p_2 must be executed before p_8, and only R_3 is free until time 2, we get $f_r(p_8) \geq 2$. So at time 2 we still have to execute p_6, p_7, p_8 with resources R_2, R_3. Therefore, two out of p_6, p_7, p_8 must be executed sequentially by the same resource. Since p_6, p_7, p_8 take 2 time units each, one of the two resources needs time 4, and we get $f_r(o) \geq 6$.

By this claim, at time 0, an online schedule has to decide whether to allocate a resource to p_3 or not, without knowing which of t_3 or t_4 will be executed in the future. If it schedules $f(p_3) = 0$ and later t_4 occurs, then Π_r is executed and the deadline of 5 time units is not met. The same occurs if it schedules $f(p_3) > 0$, and later t_3 occurs.

4 Concurrency Threshold

Due to the two negative results presented in the previous section, we study a different parameter, introduced in [4], called the concurrency threshold. During execution of a business process, information on the resolution of future choices is often not available, and further no information on the possible duration of a task (or only weak bounds) are known. Therefore, the scheduling is performed in practice by assigning a resource to a task at the moment some resource becomes available. The question is: What is the minimal number of resources needed to guarantee the optimal execution time achievable with an unlimited number of resources?

The answer is simple: since there is no information about the duration of tasks, every reachable marking of the workflow net without durations may be also reached for some assignment of durations. Let M be a reachable marking with a maximal number of tokens, say k, in places with positive duration, and let $d_1 \leq d_2 \leq \cdots \leq d_k$ be the durations of their associated tasks. If less than k resources are available, and we do not assign a resource to the task with duration d_k, we introduce a delay with respect to the case of an unlimited number of resources. On the contrary, if the number of available resources is k, then the scheduler for k resources can always simulate the behaviour of the scheduler for an unlimited number of resources.

Definition 5. *Let $N = (P, T, F, I, O, \tau)$ be a workflow Petri net. For every marking M of N, define the* concurrency *of M as* $\mathrm{conc}(M) \stackrel{def}{=} \sum_{p \in D} M(p)$, *where $D \subseteq P$ is the set of places $p \in P$ such that $\tau(p) > 0$. The* concurrency threshold of*N is defined by*

$$CT(N) \stackrel{def}{=} \max\left\{\mathrm{conc}(M) \mid M \in \mathcal{R}^N(M)\right\}.$$

The following lemma follows easily from the definitions.

Lemma 2. *For every workflow net N: $RT(N) \leq CT(N)$.*

Proof. Follows immediately from the fact that for every schedule f of a run of N, there is a schedule g with $CT(N)$ machines such that $g(p) \leq f(p)$ for every place p.

In the rest of the paper we study the complexity of computing the concurrency threshold. In [4], it was shown that the threshold can be computed in polynomial time for regular workflows, a class with a very specific structure, and the problem for the general free-choice case was left open. In Sect. 4.1 we prove that the concurrency threshold of marked graphs can be computed in polynomial time by reduction to a linear programming problem over the rational numbers. In Sect. 4.2 we study the free-choice case. We show that deciding if the threshold exceeds a given value is NP-complete for acyclic, sound free-choice workflow nets. Further, it can be computed by solving the same linear programming problem as in the case of marked graphs, but over the integers. Finally, we show that in the cyclic case the problem remains NP-complete, but the integer linear programming problem does not necessarily yield the correct solution.

4.1 Concurrency Threshold of Marked Graphs

The concurrency threshold of marked graphs can be computed using a standard technique based on the *marking equation* [16]. Given a net $N = (P, T, F)$, define the *incidence matrix* of N as the $|P| \times |T|$ matrix $\boldsymbol{N}$ given by:

$$\boldsymbol{N}(p,t) = \begin{cases} 1 & \text{if } p \in t^\bullet \setminus {}^\bullet t \\ -1 & \text{if } p \in {}^\bullet t \setminus t^\bullet \\ 0 & \text{otherwise} \end{cases}$$

In the following, we denote by $\boldsymbol{M}$ the representation of a marking M as a vector of dimension $|P|$. Let N be a Petri net, and let M_1, M_2 be markings of N. The following results are well known from the literature (see e.g. [16]):

- If M_2 is reachable from M_1 in N, then $\boldsymbol{M_2} = \boldsymbol{M_1} + \boldsymbol{N} \cdot \boldsymbol{X}$ for some integer vector $\boldsymbol{X} \geq 0$.
- If N is a marked graph and $\boldsymbol{M_2} = \boldsymbol{M_1} + \boldsymbol{N} \cdot \boldsymbol{X}$ for some *rational* vector $\boldsymbol{X} \geq 0$, then M_2 is reachable from M_1 in N.
- If N is acyclic and $\boldsymbol{M_2} = \boldsymbol{M_1} + \boldsymbol{N} \cdot \boldsymbol{X}$ for some *integer* vector $\boldsymbol{X} \geq 0$, then M_2 is reachable from M_1 in N.

Given a workflow net $N = (P, T, F, I, O, \tau)$, let $\boldsymbol{D} \colon P \mapsto \mathbb{N}$ be the vector defined by $\boldsymbol{D}(p) = 1$ if $p \in D$ and $\boldsymbol{D}(p) = 0$ if $p \notin D$, where D is the set of places with positive duration. We define the linear optimization problem

$$\ell^N = \max \{ \boldsymbol{D} \cdot \boldsymbol{M} \mid \boldsymbol{M} = \boldsymbol{M_I} + \boldsymbol{N} \cdot \boldsymbol{X}, \boldsymbol{M} \geq 0, \boldsymbol{X} \geq 0 \} \tag{1}$$

Since the solutions of $\boldsymbol{M} = \boldsymbol{M_I} + \boldsymbol{N} \cdot \boldsymbol{X}$ contain all the reachable markings of (N, M_I), we have $\ell^N \geq CT(N)$. Further, using these results above, we obtain:

Theorem 2. *Let N be a workflow net, and let $\ell^N_{\mathbb{Q}}$ and $\ell^N_{\mathbb{Z}}$ be the solution of the linear optimization problem (1) over the rationals and over the integers, respectively. We have:*

- $\ell^N_{\mathbb{Q}} \geq \ell^N_{\mathbb{Z}} \geq CT(N)$;
- *If N is a marked graph, then $\ell_{\mathbb{Q}} = \ell_{\mathbb{Z}} = CT(N)$.*
- *If N is acyclic, then $\ell_{\mathbb{Q}} \geq \ell_{\mathbb{Z}} = CT(N)$.*

In particular, it follows that $CT(N)$ can be computed in polynomial time for marked graphs, acyclic or not. (The result about acyclic nets is used in the next section.)

4.2 Concurrency Threshold of Free-Choice Nets

We study the complexity of computing the concurrency threshold of free-choice workflow nets. We first show that, contrary to numerous other properties for which there are polynomial algorithms, deciding if the concurrency threshold exceeds a given value is NP-complete.

Theorem 3. *The following problem is NP-complete:*

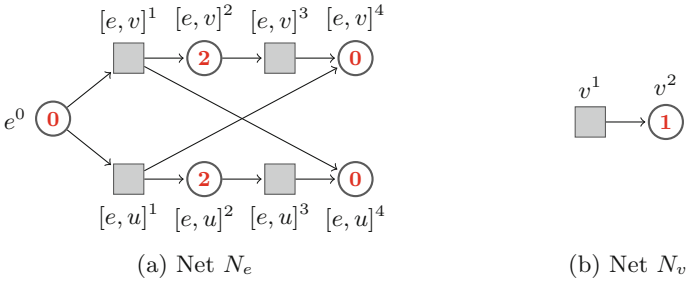
Given: *A sound, free-choice workflow net $N = (P, T, F, I, O)$, and a number $k \leq |T|$.*
Decide: *Is the concurrency threshold of N at least k?*

Proof. A detailed proof can be found in the full version of this paper [15], here we only sketch the argument. Membership in NP is nontrivial, and follows from results of [1,7]. We prove NP-hardness by means of a reduction from Maximum Independent Set (MIS):

Given: An undirected graph $G = (V, E)$, and a number $k \leq |V|$.
Decide: Is there a set $In \subseteq V$ such that $|In| \geq k$ and $\{v, u\} \notin E$ for every $u, v \in In$?

Given a graph $G = (V, E)$, we construct a sound free-choice workflow net N_G in polynomial time as follows:

- For each $e = \{v, u\} \in E$ we add to N_G the "gadget net" N_e shown in Fig. 5(a), and for every node v we add the gadget net N_v shown in Fig. 5(b).
- For every $e = \{v, u\} \in E$, we add an arc from the place $[e, v]^4$ of N_e to the transition v^1 of N_v, and from $[e, u]^4$ to the transition u^1 of N_u.
- The set I of initial places contains the place e^0 of N_e for every edge e; the set O of output places contains the places v^2 of the nets N_v.



(a) Net N_e (b) Net N_v

Fig. 5. Gadgets for the proof of Theorem 3.

It is easy to see that N_G is free-choice and sound, and in [15] we show the result of applying the reduction to a small graph and prove that G has an independent set of size at least k iff the concurrency threshold of (N_G, M_I) is at least $2|E| + k$. The intuition is that for each edge $e \in E$, we fire the transition $[e, u]^1$ where $u \notin In$, and for each $v \in In$, we fire the transition v^1, thus marking one of $[e, u]^2$ or $[e, v]^2$ for each edge $e \in E$ and the place v^2 for each $v \in In$.

4.3 Approximating the Concurrency Threshold

Recall that the solution of problem (1) over the rationals or the integers is always an upper bound on the concurrency threshold for any Petri net (Theorem 2). The question is whether any stronger result holds when the workflows are sound and free-choice. Since computing the concurrency threshold is NP-complete, we cannot expect the solution over the rationals, which is computable in polynomial time, to provide the exact value. However, it could still be the case that the solution over the integers is always exact. Unfortunately, this is not true, and we can prove the following results:

Theorem 4. *Given a Petri net N, let $\ell^N_{\mathbb{Q}}$ and $\ell^N_{\mathbb{Z}}$ be as in Theorem 2.*

(a) There is an acyclic sound free-choice workflow net N such that $CT(N) < \ell^N_{\mathbb{Q}}$.
(b) There is a sound free-choice workflow net N such that and let $CT(N) < \ell^N_{\mathbb{Z}}$.

Proof. For (a), we can take the net obtained by adding to the gadget in Fig. 5(a) a new transition with input places $[e, v]^4$ and $[e, u]^4$, and an output place o with weight 2. We take e^0 as input place. The concurrency threshold is clearly 2, reached, for example, after firing $[e, v]^1$. However, we have $\ell^N_{\mathbb{Q}} = 3$, reached by the rational solution $\boldsymbol{X} = (1/2, 1/2, \ldots, 1/2)$. Indeed, the marking equation then yields the marking M satisfying $M([e, v]^2) = M([e, u]^2) = M(o) = 1/2$.

For (b), we can take the workflow net of Fig. 6. It is easy to see that the concurrency threshold is equal to 1. The marking $\boldsymbol{M}$ that puts one token in each of the two places with weight 1, and no token in the rest of the places, is not reachable from M_I. However, it is a solution of the marking equation, even when solved over the integers. Indeed, we have $\boldsymbol{M} = \boldsymbol{M_I} + \boldsymbol{N} \cdot \boldsymbol{X}$ for $\boldsymbol{X}$=(1,0,1,1,0,0,1). Therefore, the upper bound derived from the marking equation is 2.

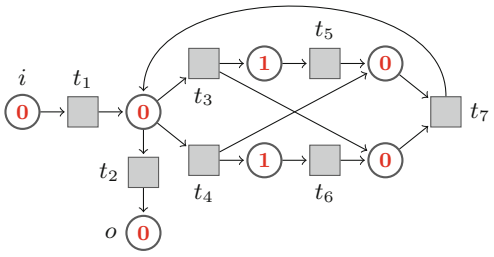


Fig. 6. A sound free-choice workflow net for which the linear programming problem derived from the marking equation does not yield the exact value of the concurrency bound, even when solved over the integers.

5 Concurrency Threshold: A Practical Approach

We have implemented a tool[1] to compute an upper bound on the concurrency threshold by constructing a linear program and solving it by calling the mixed-integer linear programming solver Cbc from the COIN-OR project [14]. Additionally, fixing a number k, we used the state-of-the art Petri net model checker LoLA [19] to both establish a lower bound, by querying LoLA for existence of a reachable marking M with $\mathrm{conc}(M) \geq k$; and to establish an upper bound, by querying LoLA if all reachable markings M' satisfy $\mathrm{conc}(M') \leq k$.

We evaluated the tool on a set of 1386 workflow nets extracted from a collection of five libraries of industrial business processes modeled in the IBM WebSphere Business Modeler [9]. For the concurrency threshold, we set $D = P \setminus O$. These nets also have multiple output places, however with a slightly different semantics for soundness allowing unmarked output places in the final marking. We applied the transformation described in [12] to ensure all output places will be marked in the final marking. This transformation preserves soundness and the concurrency threshold.

All of the 1386 nets in the benchmark libraries are free-choice nets. We selected the sound nets among them, which are 642. Out of those 642 nets, 409 are marked graphs. Out of the remaining 233 nets, 193 are acyclic and 40 cyclic. We determined the exact concurrency threshold of all sound nets with LoLA using state-space exploration. Figure 7 shows the distribution of the threshold.

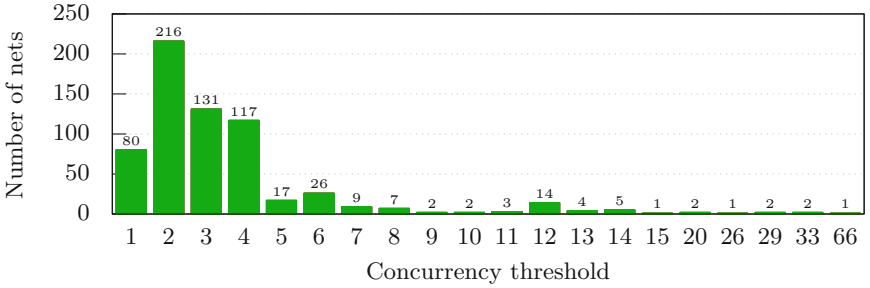


Fig. 7. Distribution of the concurrency threshold of the 642 nets analyzed.

On all 642 sound nets, we computed an upper bound on the concurrency threshold using our tool, both using rational and integer variables. We computed lower and upper bounds using LoLA with the value $k = CT(N)$ of the concurrency threshold. We report the results for computing the lower and upper bound separately.

All experiments were performed on the same machine equipped with an Intel Core i7-6700K CPU and 32 GB of RAM. The results are shown in Table 1.

[1] The tool is available from https://gitlab.lrz.de/i7/macaw.

Using the linear program, we were able to compute an upper bound for all nets in total in less than 7 s, taking at most 30 ms for any single net. LoLA could compute the lower bound for all nets in 6 s LoLA fails to compute the upper bound in three cases due to reaching the memory limit of 32 GB. For the remaining 639 nets, LoLA could compute the upper bound within 7 min in total.

We give a detailed analysis for the 9 nets with a state space of over one million. For three nets with state space of sizes 10^9, 10^{10} and 10^{17}, LoLa reaches the memory limit. For four nets with state spaces between 10^6 and 10^8 and concurrency threshold above 25, LoLA takes 2, 10, 48 and 308 s each. For two nets with a state space of 10^8 and a concurrency threshold of just 11, LoLA can establish the upper bound in at most 20 ms. The solution of the linear program can be computed in all 9 cases in less than 30 ms.

Table 1. Statistics on the size and analyis time for the 642 nets analyzed. The times marked with * exclude the 3 nets where LoLA reaches the memory limit.

	Net size				Analysis time (sec)			
	$\|P\|$	$\|T\|$	$\|\mathcal{R}^N\|$	$CT(N)$	$\ell^N_{\mathbb{Q}}$	$\ell^N_{\mathbb{Z}}$	$CT(N) \geq k$	$CT(N) \leq k$
Median	21	14	16	3	0.01	0.01	0.01	0.01
Mean	28.4	18.6	$3 \cdot 10^{14}$	3.7	0.01	0.01	0.01	0.58*
Max	262	284	$2 \cdot 10^{17}$	66	0.03	0.03	1.18	307.76*

Comparing the values of the upper bound, first we observed that we obtained the same value using either rational or integer variables. The time difference between both was however negligible. Second, quite surprisingly, we noticed that the upper bound obtained from the linear program is exact in all of our cases, even for the cyclic ones. Further, it can be computed much faster in several cases than the upper bound obtained by LoLA and it gives a bound in all cases, even when the state-space exploration reaches its limit. By combining linear programming for the upper bound and state-space exploration for the lower bound, an exact bound can always be computed within a few seconds.

6 Conclusion

Planning sufficient execution resources for a business or production process is a crucial part of process engineering [3,13,20]. We considered a simple version of this problem in which resources are uniform and tasks are not interruptible. We studied the complexity of computing the resource threshold, i.e., the minimal number of resources allowing an optimal makespan. We showed that deciding if the resource threshold exceeds a given bound is NP-hard even for acyclic marked graphs. For this reason, we investigated the complexity of computing the concurrency threshold, an upper bound of the resource threshold introduced in [4]. Solving a problem left open in [4], we showed that deciding if

the concurrency threshold exceeds a given bound is NP-hard for general sound free-choice workflow nets. We then presented a polynomial-time approximation algorithm, and showed experimentally that it computes the *exact* value of the concurrency threshold for all benchmarks of a standard suite of free-choice workflow nets.

References

1. van der Aalst, W.M.P.: Verification of workflow nets. In: Azéma, P., Balbo, G. (eds.) ICATPN 1997. LNCS, vol. 1248, pp. 407–426. Springer, Heidelberg (1997). https://doi.org/10.1007/3-540-63139-9_48
2. van der Aalst, W.M.P.: Workflow verification: finding control-flow errors using Petri-net-based techniques. In: van der Aalst, W., Desel, J., Oberweis, A. (eds.) Business Process Management. LNCS, vol. 1806, pp. 161–183. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-45594-9_11
3. Bessai, K., Youcef, S., Oulamara, A., Godart, C., Nurcan, S.: Resources allocation and scheduling approaches for business process applications in cloud contexts. In: 4th IEEE International Conference on Cloud Computing Technology and Science Proceedings, CloudCom 2012, Taipei, Taiwan, 3–6 December 2012, pp. 496–503 (2012)
4. Botezatu, M., Völzer, H., Thiele, L.: The complexity of deadline analysis for workflow graphs with multiple resources. In: La Rosa, M., Loos, P., Pastor, O. (eds.) BPM 2016. LNCS, vol. 9850, pp. 252–268. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-45348-4_15
5. Chuzhoy, J., Codenotti, P.: Resource minimization job scheduling. In: Dinur, I., Jansen, K., Naor, J., Rolim, J. (eds.) APPROX/RANDOM -2009. LNCS, vol. 5687, pp. 70–83. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-03685-9_6
6. Chuzhoy, J., Guha, S., Khanna, S., Naor, J.: Machine minimization for scheduling jobs with interval constraints. In: 45th Symposium on Foundations of Computer Science (FOCS 2004), 17–19 October 2004, Rome, Italy, Proceedings, pp. 81–90 (2004)
7. Desel, J., Esparza, J.: Free Choice Petri Nets. Cambridge University Press, Cambridge (1995)
8. Esparza, J., Hoffmann, P., Saha, R.: Polynomial analysis algorithms for free choice probabilistic workflow nets. In: Agha, G., Van Houdt, B. (eds.) QEST 2016. LNCS, vol. 9826, pp. 89–104. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-43425-4_6
9. Fahland, D., Favre, C., Jobstmann, B., Koehler, J., Lohmann, N., Völzer, H., Wolf, K.: Instantaneous soundness checking of industrial business process models. In: Dayal, U., Eder, J., Koehler, J., Reijers, H.A. (eds.) BPM 2009. LNCS, vol. 5701, pp. 278–293. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-03848-8_19
10. Favre, C., Fahland, D., Völzer, H.: The relationship between workflow graphs and free-choice workflow nets. Inf. Syst. **47**, 197–219 (2015)
11. Hall, N.G., Sriskandarajah, C.: A survey of machine scheduling problems with blocking and no-wait in process. Oper. Res. **44**(3), 510–525 (1996)
12. Kiepuszewski, B., ter Hofstede, A.H.M., van der Aalst, W.M.P.: Fundamentals of control flow in workflows. Acta Inf. **39**(3), 143–209 (2003)

13. Liu, L., Zhang, M., Lin, Y., Qin, L.: A survey on workflow management and scheduling in cloud computing. In: 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGrid 2014, Chicago, IL, USA, 26–29 May 2014, pp. 837–846 (2014)
14. Lougee-Heimer, R.: The common optimization interface for operations research: promoting open-source software in the operations research community. IBM J. Res. Dev. **47**(1), 57–66 (2003)
15. Meyer, P.J., Esparza, J., Völzer, H.: Computing the concurrency threshold of sound free-choice workflow nets. arXiv:1802.08064 [cs.LO] (2018)
16. Murata, T.: Petri nets: properties, analysis, and applications. Proc. IEEE **77**(4), 541–576 (1989)
17. Pinedo, M.L.: Scheduling: Theory, Algorithms, and Systems. Springer, New York (2016). https://doi.org/10.1007/978-1-4614-2361-4
18. Ullman, J.: NP-complete scheduling problems. J. Comput. Syst. Sci. **10**(3), 384–393 (1975)
19. Wolf, K.: Generating Petri net state spaces. In: Kleijn, J., Yakovlev, A. (eds.) ICATPN 2007. LNCS, vol. 4546, pp. 29–42. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-73094-1_5
20. Xu, J., Liu, C., Zhao, X.: Resource allocation vs. business process improvement: how they impact on each other. In: Dumas, M., Reichert, M., Shan, M.-C. (eds.) BPM 2008. LNCS, vol. 5240, pp. 228–243. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-85758-7_18