Optimization based on timed Petri nets using CP-SAT - an integrated SAT and CP solver

Discrete Event Dynamic Systems (2026) 36:29
https://doi.org/10.1007/s10626-026-00447-8

RESEARCH

Optimization based on timed Petri nets using CP-SAT - an integrated SAT and CP solver

Bengt Lennartson[1]

Received: 5 December 2024 / Accepted: 3 July 2026

Abstract
A simple but general semantics for timed Petri nets is presented in this paper. Based on this semantics, an optimization formulation is introduced and implemented in a recently developed optimization solver, where a satisfiability (SAT) solver is integrated with constraint programming (CP). The solver, called CP-SAT, is a part of Google's OR-Tools. The optimization formulation includes both concurrent and alternative sequences of operations, involving shared as well as alternative resources. The proposed optimization strategy is compared with the SAT/SMT-based solver Z3Opt and Gurobi's mixed integer linear programming (MILP) solver. The conclusion is that the computation time for CP-SAT is much shorter than for Z3Opt, while MILP is able to handle deep problems, including long sequences with many transitions, with similar computational performance as CP-SAT. On the other hand, CP-SAT is much faster than MILP for wide problems, including many parallel sequences. An evaluation of an industrial-sized flexible manufacturing system, which involves uncontrollable events, also demonstrates how efficient and easy to implement the proposed strategy is compared to existing results. In addition, it is also demonstrated how basic functionality in SAT and constraint programming are related and integrated in CP-SAT. The conclusion is that the strength of CP-SAT depends on its successful integration of search, inference, and OR-based relaxation on top of a satisfiability solver.

Keywords Optimization · Timed Petri nets · Satisfiability solvers · Constraint programming

1 Introduction

Time-optimal supervisors are often formulated based on time-weighted automata (Su et al. 2012; Ware and Su 2017), where abstractions have also been proposed (Hill and Lafortune 2016), including weak bisimulation (Vilela and Hill 2022). An alternative abstraction is based on local optimization, in which only the best (minimal time) local paths are preserved

✉ Bengt Lennartson
bengt.lennartson@chalmers.se

[1] Division of Systems and Control, Department of Electrical Engineering, Chalmers University of Technology, SE-412 96, Göteborg, Sweden

(Hagebring and Lennartson 2019). In (Lennartson 2022), this compositional abstraction is simplified using timed automata (Alur and Dill 1994). For Petri nets (PNs), the main focus of this paper, such abstractions can only be applied to safe nets, which have at most one token in each place. When multiple tokens are introduced that require, for instance, a specific single-capacity resource, it implies that only one token can be served by this resource at a time. The resource is then shared among the tokens, resulting in non-local behavior, and abstractions involving shared resources in a timed framework yield only minor reductions. Thus, rather than abstractions, the solution is to seek effective global optimization strategies.

Optimization strategies for systems with discrete states have been developed in both operations research (OR) and artificial intelligence, especially within the CP community. In (Hooker 2012), it is observed that combining search, inference, and relaxation strategies from both OR and CP can yield significant algorithmic improvements. Satisfiability solvers based on SAT and satisfiability modulo theories (SMT) have also been extended towards optimization. This is possible since Boolean and linear constraints on integers can be formulated as pseudo-Boolean constraints, which can be translated to pure satisfiability problems (Eén and Sörensson 2006). A constraint on the optimization criterion is then introduced, where the constraint can be decreased until the solution becomes unsatisfiable. The lowest value that yields a satisfiable solution then generates the minimal time-optimal solution. This strategy has been implemented in the SMT solver Z3 and is called Z3Opt (Bjørner and Phan 2014). A second SAT-based solver has followed Hooker's recommendation and combined the satisfiability strategy with specific CP-oriented functions and OR-based relaxations to more efficiently handle both shared and alternative resources in concurrent sequences. The solver, CP-SAT, is part of Google's OR-Tools (CP-SAT 2024a, b).

This paper shows that the improvement using CP-SAT is significant compared with Z3Opt. For typical scheduling problems with a large number of shared resources, CP-SAT is often 100 times faster, or no solution is found by Z3Opt. For deep problems that involve many sequential operations but a limited number of sequences, Gurobi's MILP solver (Gurobi 2015) performs well, sometimes even slightly better than CP-SAT. On the other hand, for wide problems with extensive concurrency, involving many alternative paths, CP-SAT is much more efficient than MILP. Inspired by this strength, we show how the makespan optimization of timed Petri nets (TPNs) can be formulated and solved in CP-SAT. It is also shown that certain PN-related constraints are critical when optimizing TPNs with multiple tokens. Compared to a recent generic MILP formulation of TPNs (Marino et al. 2020), no explicit global marking states are introduced. The dynamic state information is determined symbolically and implicitly by constraints on the local start and end times of the tokens involved in each timed transition. Including the CP-SAT-based computation, this strategy drastically reduces memory and computation requirements. A confirmation on an industrial-sized example shows both how suitable the PN formulation is compared to modular timed automata and how much faster the optimization is performed with CP-SAT than with a near-optimal evolutionary algorithm (Pena et al. 2022). It is also demonstrated how uncontrollable events can be handled with some easily implemented additional constraints.

To better understand the integration between SAT and CP and the strength of CP-SAT, a survey of the main functionality of a modern SAT solver, called conflict-driven clause learning (CDCL) in Marques-Silva and Malik (2018), is also provided based on a small example. It is then shown how even CP can use the CDCL framework for constraint propagation, based on lazy clause generation (LCG) (Ohrimenko et al. 2009). LCG and its implementation, Chuffed,

are also mentioned in Perron (2023) as a main contribution to Google's development of CP-SAT. In the CP-SAT implementation, Hooker's strong recommendation, mentioned above, to integrate OR, CP, and SAT in an efficient search, inference, and relaxation strategy, is also demonstrated.

The main contributions of this paper are the introduction of a simple but general semantics for TPNs and an optimization formulation based on this semantics. This formulation is implemented in CP-SAT and evaluated on deep (long sequences) and wide (many concurrent sequences) scheduling problems, and compared with Z3Opt and Gurobi's MILP solver. In the next section, a simple semantics is introduced for TPNs, and the relation between timed automata and TPNs is clarified. This is followed by a timed optimization formulation, evaluated in Section 3 and applied to makespan optimization of a flexible manufacturing system in Section 4. In Section 5, the relationship between SAT and CP, and their integration in CP-SAT, are illustrated, followed by final conclusions in Section 6.

2 Timed automata and timed Petri nets

In performance analysis and more generally time optimization, operation and waiting times, as well as time delays, need to be defined. Discrete event models are therefore augmented with explicit timing information, in which duration is a central notion. Duration is the amount of time that has elapsed between two events, often called start and end events. The inclusion of duration in discrete event models is therefore presented in this section. We will consider both timed automata and timed PNs, and discuss their relations and differences to better understand the benefits of using PNs for timed systems.

2.1 Clocks and timed automata

The most common way to introduce duration in automata is to include a set of clocks C, resulting in a *timed automaton* (Alur and Dill 1994; Bouyer et al. 2018). Constraints on these clocks are used to specify time conditions, and to be able to define a timed automaton, let $\Phi(C)$ denote the set of clock constraints over C.

A *timed automaton* is a five-tuple $G = \langle L, \Sigma, C, T, \ell_0 \rangle$, where L is a finite set of locations, Σ is a finite set of events, C is a finite set of clocks where the time domain of each clock c_i is $\Re_{\geq 0}$, $T \subseteq L \times \Sigma \times \Phi(C) \times 2^C \times L$ is a transition relation, where $t = (\ell, a, \varphi, r, \ell') \in T$ includes a source location ℓ, an event label a, a clock constraint $\varphi \in \Phi(C)$, a set r of clocks to be reset, and a target location ℓ' of the transition t. Finally, $\ell_0 \subseteq L$ is an initial location. A transition $(\ell, a, \varphi, r, \ell')$ with $r = \{c\}$ is also denoted $\ell \xrightarrow{a:\varphi, c:=0} \ell'$.

Limited set of clock constraints Focusing on time optimization of concurrent sequences of operations, mainly one clock constraint is required. In Fig. 1, two operations in the timed automaton G_ℓ are included. The first one starts when the event $a_{1\ell}$ occurs. The clock c_ℓ is then reset to zero by the assignment $c_\ell := 0$. The clock constraint $c_\ell = \Delta_{1\ell}$ specifies that the duration $\Delta_{1\ell}$ has passed when the event $b_{1\ell}$ occurs, and the operation is completed. The second operation is modeled in the same way.

Thus, the only clock constraints that are required in time optimization are inductively defined by the grammar $\varphi ::= c = \Delta \,|\, \varphi_1 \wedge \varphi_2$, where $c = \Delta$ specifies that the value of

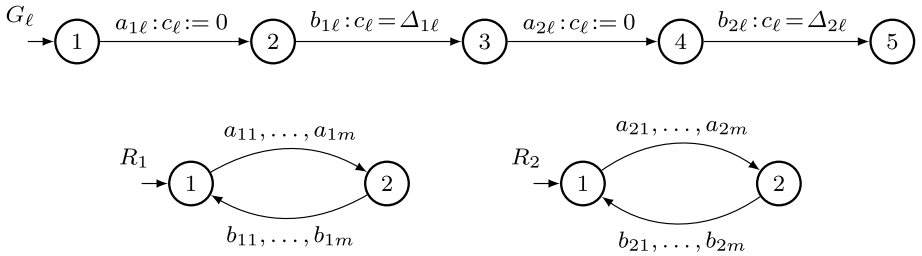

Fig. 1 Timed automata G_ℓ, $\ell = 1, \ldots, m$, each one modeling two operations with durations $\Delta_{\ell 1}$ and $\Delta_{\ell 2}$, and untimed automata R_1 and R_2, modeling resources that can only execute one operation at a time

the clock c is equal to the duration $\Delta \in \mathbb{Z}_{\geq 0}$, and $\varphi_1 \wedge \varphi_2$ is the conjunction of the clock constraints φ_1 and φ_2.

Synchronous composition of timed automata One strength of timed automata is the ability to easily define the composition of timed subsystems. The simplicity comes from the fact that the actual time, expressed symbolically by transition predicates (the clock constraints), is separated from the location nodes. The synchronous composition of two timed automata G_1 and G_2, denoted $G_1 \,\|\, G_2$, is defined for instance in Lennartson (2022). The main principle in this composition is that a shared event must occur at the same time in both automata in the synchronized system, while the conjunction of the individual clock constraints in G_1 and G_2 is satisfied. On the other hand, events that occur only in one of the timed automata occur locally, without any interaction between the two subsystems.

The synchronous composition $G_1 \,\|\, \cdots \,\|\, G_m \,\|\, R_1 \,\|\, R_2$ of the automata in Fig. 1 models two resources R_1 and R_2 that are shared between m sequences of operations $G_1, \ldots, G_m$, each one including two operations. Resource R_1 only allows one of the concurrent sequences at a time to execute its first operation. In the same way, and concurrently, resource R_2 allows a sequence to execute its second operation, one at a time.

2.2 Timed Petri nets

A PN with explicit timing information will now be introduced. It is called a timed Petri net (TPN) and is defined as a six-tuple

$$\mathcal{N} = \langle P, T, Pre, Post, m_0, \Delta \rangle,$$

where P is a set of *places* (graphically represented as circles) and T is a set of *transitions* (graphically represented as bars). For arbitrary places $p \in P$ and transitions $t \in T$, *Pre* and *Post* are matrices, defining the graphic structure of the net, where $Pre(p, t) = w$ means that there is an arc from place p to transition t with weight w, while $Post(p, t) = w$ means that there is an arc from transition t to place p. A *marking* is a vector m that assigns to each place $p \in P$ a non-negative integer number of tokens $m(p)$, and m_0 is the initial marking. Furthermore, $\Delta : T \to \Re_{\geq 0}$ is a function that assigns a duration $\Delta(t)$ to the firing of transition $t \in T$ (Ramchandani 1973; Chretienne 1986; Silva and del Foyo 2012). A transition t with duration $\Delta(t) > 0$ is graphically

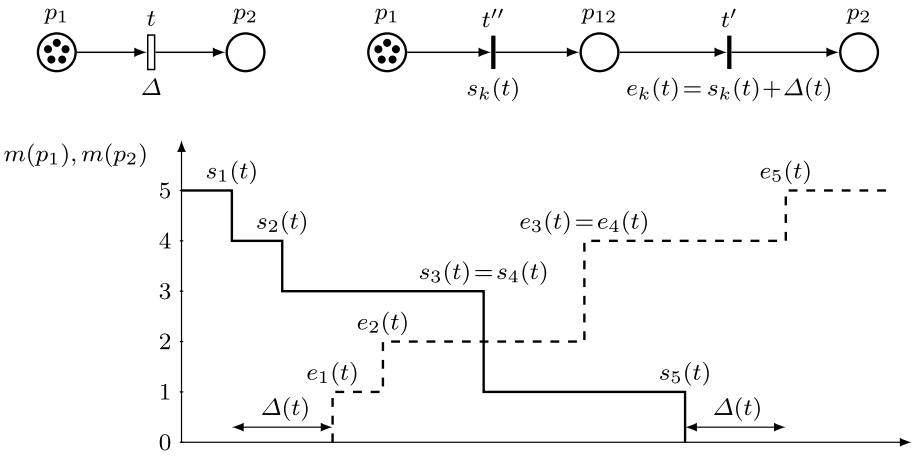

Fig. 2 A TPN that includes one transition t with duration $\Delta(t)$ is shown at the top, along with a more detailed PN. In the detailed model, the start and end times of the timed transition are introduced as labels below the instantaneous transitions. When transition t is fired repeatedly with free speed, the change of the number of tokens m in the two places p_1 (solid) and p_2 (dashed) is also shown together with the start and end times $s_k(t)$ and $e_k(t)$

shown as a non-filled bar with transition label Δ, see the TPN at the top left of Fig. 2, while an instantaneous transition with zero-time duration is shown as a filled bar.

Fixed durations and duration intervals A TPN has fixed (deterministic) durations $\Delta(t)$, $t \in T$, which can be generalized to non-deterministic duration intervals spanning from a minimal duration Δ_{min} to a maximal duration Δ_{max}. PNs that include such non-deterministic timing behavior are called Time PNs, as proposed by Merlin and Farber (1976). TPNs are suitable for scheduling and performance evaluation, i.e. the focus of this paper, while Time PNs are more relevant for verification, including safety analysis of systems with explicit lower and upper time limits.

Semantics of timed Petri nets A transition $t \in T$ in a PN is *enabled* when the number of tokens $m(p) \geq Pre(p,t)$, and then t can occur. When the transition occurs, it is said to be fired, and in a TPN, the firing or execution of a transition t has a duration $\Delta(t) \geq 0$. This means that the time $s(t)$, when the firing of t starts, results in the end time $e(t) = s(t) + \Delta(t)$ when the firing of t is completed. Since a transition can occur repeatedly, it is also of interest to know the times at which individual transitions are fired. When transition t is fired the k-th time, the corresponding start and end times are denoted $s_k(t)$ and $e_k(t)$. Similar terminology is introduced in Ramchandani (1973), where the start and end times are referred to as the initiation and termination times.

At the start time $s_k(t)$, the number of tokens in place p is updated as

$$m''(p) = m(p) - Pre(p,t),$$

which means that $Pre(p, t)$ tokens are taken from the input places to transition t at time instance $s_k(t)$. These tokens are then hidden until the firing of the transition is completed at

the end time $e_k(t)$, when *Post*(*p*, *t*) tokens are added to the output places of transition *t*. The number of tokens in place *p* is then updated as

$$m'(p) = m''(p) + Post(p,t) = m(p) - Pre(p,t) + Post(p,t).$$

As mentioned above, a TPN with one transition *t* is shown at the top left of Fig. 2, where the duration is $\Delta(t)$. To the right, a more detailed PN is shown, in which the start and end times of the timed transition are labeled below the instantaneous transitions t'' and t'. The notation t'' as the start transition is used to match the temporary marking m'', only relevant for TPNs but not untimed PNs.

Acceptable transition times The relation between the *k*-th and the $(k+1)$-th start time is restricted to

$$s_{k+1}(t) \geq s_k(t) \tag{1}$$

This means that a transition *t* that has been fired for the *k*:th time with start time $s_k(t)$ can be fired again before the firing of the *k*:th transition has been completed, i.e. $s_{k+1}(t) < s_k(t) + \Delta(t) = e_k(t)$ is acceptable as long as $s_{k+1}(t) \geq s_k(t)$. For the TPN in Fig. 2, initially with five tokens in place p_1, this phenomenon is illustrated at the start of the second transition, where $s_2(t) < s_1(t) + \Delta(t) = e_1(t)$.

Multiple firing of transitions The start time condition Eq. 1 even includes the case when $s_{k+1}(t) = s_k(t)$. This implies that multiple firing of transitions with the same start time is also accepted, as long as there are enough tokens in the input places. A transition *t* is then enabled with multiplicity μ when $m(p) \geq \mu Pre(p,t)$. Multiple firing of transitions is illustrated in Fig. 2 for $\mu = 2$ where $s_3(t) = s_4(t)$ and $e_3(t) = e_4(t)$, and a second illustration is presented in Example 2.

Free and maximal speed The start and end times shown in Fig. 2 are examples of repeated transitions that are fired at arbitrary time instances after their enabling. This is called *free speed* behavior (David and Alla 2010). The opposite, when transitions are fired as soon as possible, is called *maximal speed*. For the TPN in Fig. 2, there is no restriction on the number of tokens to be included in the firing of transition *t*. Thus, firing with maximal speed implies that all tokens are moved from the input place p_1 at time zero, resulting in a multiple firing of multiplicity 5 where $s_k = 0$ and $e_k = \Delta$ for all $k \in \{1, \ldots, 5\}$. Since efficient algorithms for time optimization based on TPNs are developed in this paper, the focus from now on will be on TPNs with maximal speed behavior. The following example gives a first illustration of a time-optimal makespan (minimal total execution time) for a TPN with two sequences and three shared resources.

Example 1 Consider the TPN including two sequences and three resources R_1, R_2, and R_3 in Fig. 3. The transitions that generate the optimal makespan $MS = 10$ are also shown in the Gantt scheme. The shared resource R_1 is serving both sequences, while R_2 is serving the first and R_3 the second sequence. Since all three resources have capacity one (one token in their initial places), only one token is moved from the correspond-

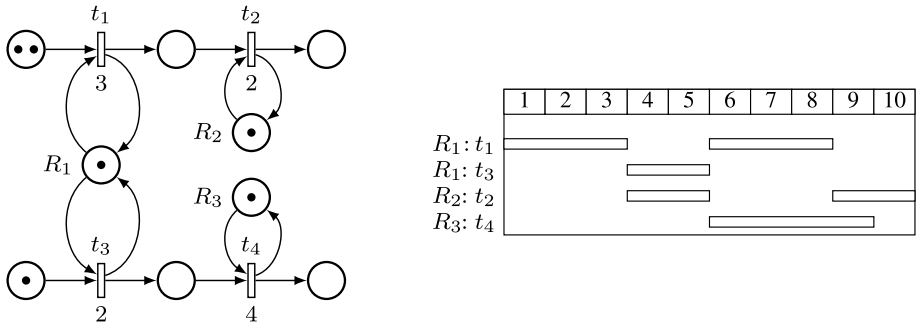

Fig. 3 A TPN including two sequences and three resources R_1, R_2, and R_3 with capacity one, and a Gantt scheme that shows the time-optimal makespan solution

ing input places when each transition is fired. It implies that for each transition t_i, the next start time $s_{k+1}(t_i) \geq s_k(t_i) + \Delta(t_i)$. The two tokens in the initial place of the first sequence mean that the transitions t_1 and t_2 are both fired twice. From the Gantt scheme, we see that the start and end times for the second transition t_2 with duration $\Delta = 2$ are $s_1(t_2) = 3$ and $e_1(t_2) = 5$ for the first firing of t_2 and $s_2(t_2) = 8$ and $e_2(t_2) = 10$ for the second firing of t_2.

Multiple resources and multiple firing In the next example, multiple resource capacity is illustrated, as well as weighted transitions and multiple firing of a transition.

Example 2 A TPN including two sequences and a resource *R* with capacity three is shown in Fig. 4, together with the Gantt scheme that generates the optimal makespan $MS = 7$. Due to the weight two on the arc from the resource place *R* to the transition t_1 and the same weight on the return arc, two of the three available tokens in place *R*, denoted *R*(1) and *R*(2) in the Gantt scheme, are first used in the single firing of transition t_1 with duration $\Delta(t_1) = 3$. Concurrently, the third token *R*(3) is used in the firing of transition t_2 with duration $\Delta(t_2) = 4$. When t_1 is completed at time $e_1(t_1) = 3$, the returned two tokens *R*(1) and *R*(2) are used in the firing of transition t_2 for the remaining tokens in the second sequence. This results in the simultaneous firing of transition t_2 the second and third times, where the start times $s_2(t_2) = s_3(t_2) = 3$ and the end times $e_2(t_2) = e_3(t_2) = 7$, i.e. a firing with multiplicity $\mu = 2$ is achieved.

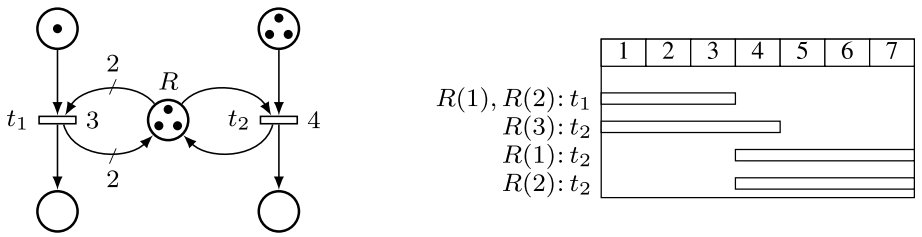

Fig. 4 A TPN including two sequences, one resource *R* with capacity three, and two transitions with weight two. The Gantt scheme shows the time-optimal makespan solution, in which the transition t_2 is simultaneously fired the second and third times

In Zaitsev and Sleptsov (1997), a similar multiple firing of transitions is defined for a special type of TPNs called *Sleptsov nets*. It is said to be a more general class of TPNs that allows multiple firing of tokens in the same way as demonstrated here, while the transitions are called multichannel transitions. The definition of Sleptsov nets in Zaitsev and Sleptsov (1997) is quite complex, but their behavior appears to be the same as the multiple-firing behavior defined in this paper. Although Ramchandani (1973) introduces a similar formalism with start and end times for repeated transitions in a firing schedule table, no multiple firing behavior or repeated firing with next start time earlier than the current end time ($s_{k+1}(t) < e_k(t)$) are demonstrated. The reason is that the presentation is limited to safe TPNs with a maximum of one token per place, such that no multiple resources are introduced.

Alternative transitions When alternative resources are able to serve individual parts (tokens), but also when a resource is able to serve alternative sequences, alternative transitions are introduced. These phenomena are illustrated in the following example.

Example 3 Consider the TPN including two sequences in Fig. 5. Resource R_2 is shared between these sequences, but with different duration times. This is possible since the

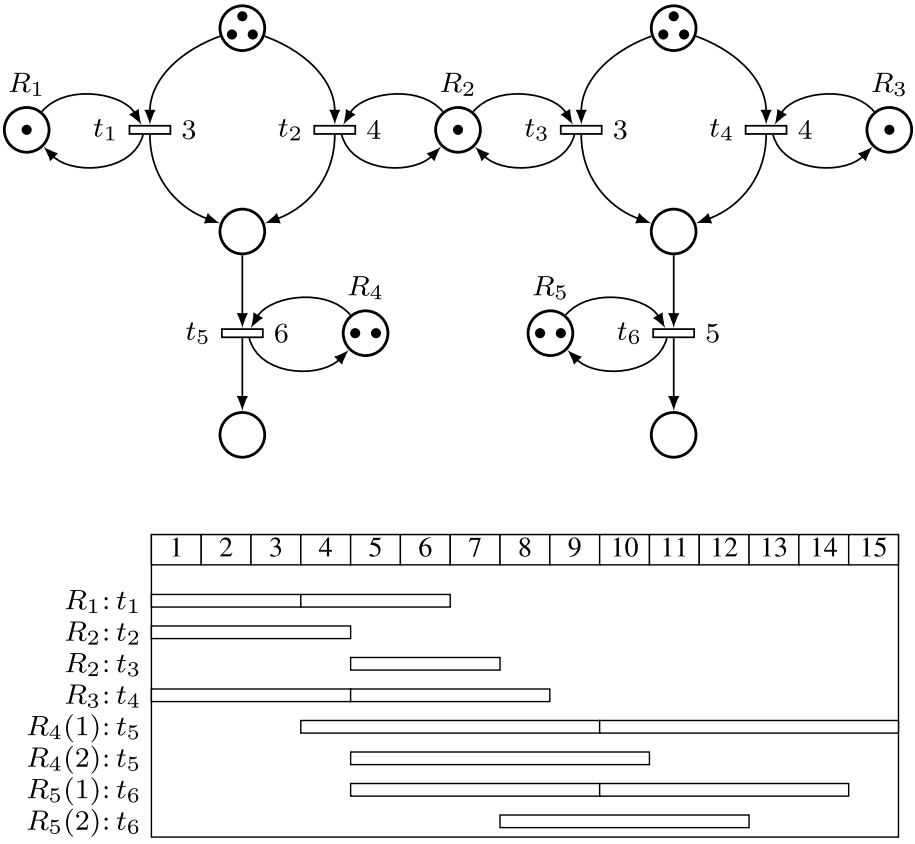

Fig. 5 A TPN with two sequences where the second resource R_2 is serving both the left and the right sequence. The Gantt scheme shows the time-optimal makespan solution where $MS = 15$. The transitions t_1 and t_4 are fired twice, t_2 and t_3 once, while both t_5 and t_6 are fired three times

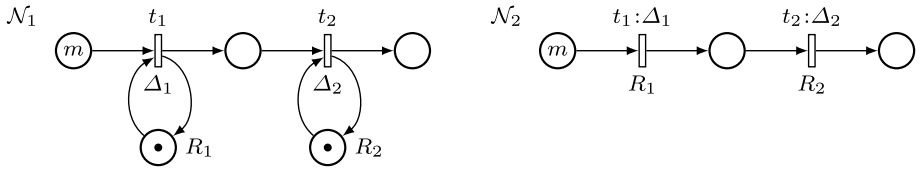

Fig. 6 A TPN $\mathcal{N}_1$ with m tokens in the initial place to the left, two resources R_1 and R_2, both with capacity one, and transitions t_1 and t_2 with durations Δ_1 and Δ_2 for each token. The second TPN $\mathcal{N}_2$ is a simplified notation of $\mathcal{N}_1$

duration time is determined by the actual transition, not by the selected resource. The left sequence for R_2 has duration $\Delta(t_2) = 4$, while the right one has duration $\Delta(t_3) = 3$. The resulting Gantt scheme in the same figure shows the time-optimal makespan $MS = 15$, where the optimal choice of resource R_2 is to first serve transition t_2 in the left sequence, followed by transition t_3 in the right sequence. Also note that the double capacity of the resources R_4 and R_5 implies that both t_5 and t_6 start their second firing before their first firing has been completed, i.e. $s_2(t_5) < e_1(t_5)$ and $s_2(t_6) < e_1(t_6)$.

Simplified notation for shared resources The TPN $\mathcal{N}_1$ in Fig. 6, with m tokens in the initial place and two resources R_1 and R_2, both with capacity one, models m parts that are first served by R_1, one at a time, with duration Δ_1, followed by R_2 with duration Δ_2. When a number of resources are involved in a TPN, such as in the optimization examples that follow in Section 3 and especially in Section 4, a simplified notation, introduced in the TPN $\mathcal{N}_2$, is preferable. The resource places R_1 and R_2 are then excluded and replaced by corresponding transition labels R_1 and R_2, respectively.

For a resource R with multiple capacity $c > 1$ (c tokens in place R), the simplified notation cR is used as an extended transition label. In Fig. 7, these simplified resource notations are applied to the TPN in Fig. 5, including two sequences with an alternative resource R_2 and multiple capacity resources R_4 and R_5. The resource transition label cR can be interpreted as a semaphore with initial value $R = c$, and a cR-transition at a transition t can only start if $R > 0$. When t starts, the action $R := R - 1$ is performed, and when it is completed, the action $R := R + 1$ is performed. This means that no cR transition can be started when $R = 0$.

2.3 Relation between timed automata and timed Petri nets

The sequences including two operations with durations $\Delta_{1\ell}$ and $\Delta_{2\ell}$, modeled as timed automata G_ℓ, $\ell = 1, \ldots, m$ in Fig. 1, are naturally represented in a TPN as m sequences with two transition $t_{1\ell}$ and $t_{2\ell}$ in each sequence $\ell = 1, \ldots, m$. Both operations in the timed automata and the transitions in the TPN are served by shared resources R_1 and R_2, each with capacity one. A TPN that is equivalent to the synchronous composition of the timed automata in Fig. 1 is shown as the net $\mathcal{N}_3$ with simplified resource notation in Fig. 8.

When the durations are equal for the operations that are served by the same resource R_1 and R_2, respectively, i.e. $\Delta_{1\ell} = \Delta_1$ and $\Delta_{2\ell} = \Delta_2$, $\ell = 1, \ldots, m$, the timed automata may alternatively be interpreted as the same operation performed by resource R_1 and R_2, respectively, on a set of m parts, but one at a time. This alternative behavior is perfectly modeled as

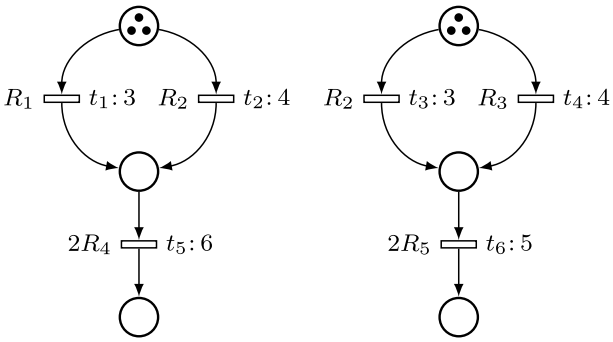

Fig. 7 The TPN in Fig. 5 is here expressed with a simplified resource notation, where resource places are replaced by added transition labels, and resource capacity $c > 1$ for a resource R is denoted by a prefix cR

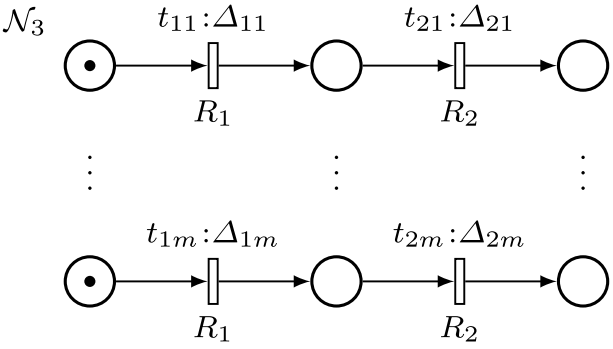

Fig. 8 A TPN $\mathcal{N}_3$ with simplified resource notation, where two shared resources R_1 and R_2 are serving m sequences, each one with two operations/transitions with individual durations $\Delta_{11}, \ldots, \Delta_{1m}$ and $\Delta_{21}, \ldots, \Delta_{2m}$, respectively, as in Fig. 1

a TPN with m tokens in the initial place, representing the m parts, served by the resources R_1 and R_2 with capacity one, i.e. the TPN shown in Fig. 6. In the next section, time optimization for this type of model is performed when both multiple initial tokens and multiple sequences of operations are included, also including alternative sequences.

3 Time optimization

It will now be shown how time optimization of TPNs can be performed by formulating a CP problem. This problem can be solved efficiently, especially by solvers that combine strategies from both OR, CP, and SAT. The integration between OR and CP is well known and strongly recommended by Hooker (2012). Satisfiability solvers based on SAT and SMT have also been extended with optimization strategies, for example, Z3Opt (Bjørner and Phan 2014). Another example is Google's OR-Tools introduction of CP-SAT (Chen 2020; CP-SAT 2024a, b). In this section, we will demonstrate how efficient the combined CP-SAT solver is, and how different types of TPNs can be modeled and optimized by this solver. First, it is shown how concurrent sequences including multiple tokens and shared as well as alternative resources are modeled. Then follows a generalization to mixed alternative and concurrent sequences. The integration between SAT and CP is also shortly explained in Section 5.2.

3.1 Optimization formulation of concurrent sequences

Optimization problems, including timing aspects, are often based on performance measures. Two of the most common criteria are *makespan*, meaning the total time to perform a number of concurrent operations, and *tardiness*, considering the deviation between completion time and deadline for individual operations (Pinedo 2022). In this paper, we will focus on makespan, noting that the proposed methods are easily reformulated to tardiness criteria.

Sets and variables Optimizing a TPN generally involves several concurrent sequences of operations, with multiple tokens and both shared and alternative resources. A set of tokens in initial places, representing discrete elements, is then moved to a set of goal places via transitions. At these transitions, services with varying durations are performed by shared resources. It is therefore natural to express time information, such as transition start and end times, for each individual token. Makespan optimization is therefore defined based on the following sets:

- $Seq = \{1, \ldots, nSeq\}$ = index set for a set of sequences,
- $Op(i) = \{1, \ldots, nOp(i)\}$ = index set for operations in sequence $i \in Seq$,
- $Tok(i) = \{1, \ldots, nTok(i)\}$ = index set for tokens in sequence $i \in Seq$,
- $R_{\oplus}(i, j) = \{1, \ldots, nR_{\oplus}(i, j)\} =$ index set for alternative resources in sequence $i \in Seq$ and operation $j \in Op(i)$,
- $\mathcal{R}$ = set of all system resources,
- *Sys* = set of tuples $(R_{i,j,\nu}, \Delta_{i,j,\nu})$, where $R_{i,j,\nu} \in \mathcal{R}$ is one alternative resource with duration $\Delta_{i,j,\nu} \in \mathbb{Z}_{\geq 0}$ in sequence $i \in Seq$ and operation $j \in Op(i)$ with alternative resource index $\nu \in R_{\oplus}(i, j)$.

Based on these sets, the following variables are introduced in sequence $i \in Seq$ for operation $j \in Op(i)$ and token $\ell \in Tok(i)$:

- $s_{i,j,\ell}$ = start time,
- $d_{i,j,\ell}$ = duration time,
- $e_{i,j,\ell} = s_{i,j,\ell} + d_{i,j,\ell}$ = end time,
- $MS = \max\{e_{i,nOp(i),\ell} | i \in Seq,\ \ell \in Tok(i)\}$ = makespan, i.e. the end time when all concurrent operations have been completed.

To be precise, sequences, operations, and tokens are denoted by their index *i*, *j*, and ℓ, while resources are denoted by their name in the resource set $\mathcal{R}$. One exception is the alternative resource index $\nu \in R_{\oplus}(i, j)$. Note that the same resource may have individual durations in different sequences and operations, such as R_2 in Fig. 5 that has duration 4 in the left and 3 in the right sequence. Also observe that $e_{i,nOp(i),\ell}$ is the end time for token ℓ in sequence *i*, since no alternative sequences, only concurrent sequences, are considered in this first optimization problem. The extension, including alternative sequences, is presented in Section 3.3.

Makespan optimization formulation The following optimization formulation of concurrent sequences, denoted OptConcurSeq, generates the minimal makespan when a number of tokens are involved in each sequence of operations. Each operation may also include a number of alternative resources, each one with individual duration and capacity.

Optimization formulation OptConcurSeq

$$\begin{aligned}
&\min MS \quad \text{subject to}\\
&\quad \forall i \in Seq,\, j \in Op(i),\, \ell \in Tok(i),\, \nu \in R_{\oplus}(i,j):\\
&\quad (R_{i,j,\nu}, \Delta_{i,j,\nu}) \in Sys,\\
&\quad b_\nu = \text{exactOne}(b_1, \ldots, b_{nR_{\oplus}(i,j)}), && (2)\\
&\quad b_\nu \rightarrow (r_{i,j,\ell} = R_{i,j,\nu}) \wedge (d_{i,j,\ell} = \Delta_{i,j,\nu}),\\
&\quad e_{i,j,\ell} = s_{i,j,\ell} + d_{i,j,\ell}, && (3)\\
&\quad s_{i,j+1,\ell} \geq e_{i,j,\ell}, \qquad j < nOp(i), && (4)\\
&\quad \text{cumulative}(i, j, \ell, r_{i,j,\ell}), && (5)\\
&\quad MS \geq e_{i,nOp(i),\ell}. && (6)
\end{aligned}$$

Alternative resources The logical function exactOne in Eq. 2, which is executed for all $\nu \in R_{\oplus}(i,j)$, introduces the constraint that exactly one of the Boolean variables included in the exactOne( ) input list is true, and the rest are false. The optimal combination of Boolean variables for the different operations will then be selected to minimize MS.

For each operation *j* in the individual sequences *i*, the chosen resource and related duration are assigned to the variables $r_{i,j,\ell}$ and $d_{i,j,\ell}$. Thus, Eqs. 2 and 3 are the only information that is required to implement alternative resources. Both the function exactOne and the conditional equalities in Eq. 3 are implemented by built-in functions included in CP-SAT, where, for instance, the conditional equality $\text{Add}(r_{i,j,\ell} == R_{i,j,\nu}, b_\nu)$ is only performed when $b_\nu = \top$. The reason why the duration $d_{i,j,\ell}$ depends on the token ℓ is that generally alternative resources with different durations can be utilized for each individual token. The optional alternative resources are removed by excluding index ν and the Boolean variable b_ν, only implementing unconditional equalities in Eq. 3 as $(r_{i,j,\ell} = R_{i,j}) \wedge (d_{i,j,\ell} = \Delta_{i,j})$. The absence of alternative resources also means that operation *j* in sequence *i* is expressed as a transition $t_{i,j}$.

Sequential relations and makespan The constraint in Eq. 4 specifies the sequential relation between the operations in all sequences. Since no alternative sequences are considered in this first concurrent optimization formulation, the end time of the last operation $e_{i,nOp(i),\ell}$ is the final time for each sequence *i* and each token ℓ. Thus, according to Eq. 6, $MS \geq e_{i,nOp(i),\ell}$, and the global minimization of MS generates the minimal makespan.

Mutual exclusion The function cumulative Eq. 5 implements mutual exclusion among the shared resources, with given capacity defined by the number of tokens in the resource places, see $R_1, \ldots, R_5$ in Fig. 5. This function generates the constraint

$$\begin{aligned}&\text{cumulative}(i,j,\ell,r_{i,j,\ell}) = \\ &\quad \forall i' \in Seq, j' \in Op(i'), \ell' \in Tok(i') : r_{i,j,\ell} = r_{i',j',\ell'} \\ &\quad \wedge \Big(|\{(i',j',\ell') : [s_{i,j,\ell}, e_{i,j,\ell}] \cap [s_{i',j',\ell'}, e_{i',j',\ell'}] \neq \emptyset\}| \\ &\quad \leq \text{capacity}(r_{i,j,\ell}) \Big)\end{aligned} \tag{7}$$

in an effective way in CP-SAT, where the timing data is delivered to this function by a dictionary with the resources as keys. Note that $(|\{(i',j',\ell') : [s_{i,j,\ell}, e_{i,j,\ell}] \cap [s_{i',j',\ell'}, e_{i',j',\ell'}] \neq \emptyset\}|$ means number of tuples (i',j',ℓ') such that the intervals $[s_{i,j,\ell}, e_{i,j,\ell}]$ and $[s_{i',j',\ell'}, e_{i',j',\ell'}]$ overlap each other. This number is restricted by the capacity of the actual resource $r_{i,j,\ell}$.

Symbolic state-space representation An important observation is that no explicit number of tokens is involved in the optimization formulation OptConcurSeq. Since the number of tokens in each place together represents the system's state space, the actual state is indirectly determined by the ordering of the start and end times. For instance, token ℓ in sequence i in the TPN in Fig. 10 is waiting in the third place between time $e(i,2,\ell)$ and $s(i,3,\ell)$ until the shared resource R_3 is available. Assuming that R_3 has single capacity, mutual exclusion is achieved when there is no time overlap between the time interval $[s(i,3.\ell), e(i,3.\ell)]$ and any other operation that is also using R_3, cf. Eq. 7.

This non-explicit state-space representation in the OptConcurSeq formulation, without any explicit token information, can be interpreted as a *symbolic state-space* representation. This symbolic formulation can be compared with other efficient symbolic state-space representations used, for instance, in planning, model checking, and supervisory control, such as binary decision diagrams (Bryant 1992) and Boolean satisfiability formulations (Eén and Sörensson 2004). In the following evaluation, the proposed symbolic OptConcurSeq formulation is shown to handle the optimization of large state-space systems. Some preliminary evaluations also show that this formulation is a powerful alternative symbolic representation for untimed planning.

Multiple tokens and linear token order When a transition t is repeated, either because of loops or multiple tokens, there is an ordering between the start times such that $s_{k+1}(t) \geq s_k(t)$. Multiple tokens in a PN normally imply that only the number of tokens m is counted, not the identity of the individual tokens. As compensation for the fact that our data representation of a TPN is symbolic, where the number of tokens in each place is not represented explicitly, the start time of each individual token ℓ is introduced. This implies that the ordering between the start times of the individual tokens is not always linear in ℓ, as illustrated in the following example. In this example, we use the notation $s^\ell(t)$ as the start time for transition t and token ℓ.

Example 4 Consider the TPN in Fig. 9 where the transition start times for the three tokens in the initial place are ordered linearly in the token index ℓ such that $s^1(t_1) = s^2(t_2) = 0 < s^3(t_1) = 2$. Due to the different durations of the two alternative transitions, the tokens arrive in the second place in a different order, resulting in the start times $s^1(t_3) = 2 < s^3(t_3) = 4 < s^2(t_3) = 6$ in the third joined transition. This is the optimal

token sequence with $MS = 8$, where we observe that token $\ell = 3$ starts before token $\ell = 2$. Forcing an ℓ ordered sequence gives the start times $s^1(t_3) = 2 < s^2(t_3) = 5 < s^3(t_3) = 7$, but the non-optimal makespan $MS = 9$.

Obviously, the order in which multiple tokens will start may change after they have passed alternative transitions with different durations. In this example, the optimal makespan requires the third token ($\ell = 3$) to be executed at t_3 before the second one, whereas keeping the token order ℓ yields a non-optimal solution. Since all combinations of start times for the individual tokens are included in the optimization formulation OptConcurSeq, the optimal makespan $MS = 8$ is achieved. This individual evaluation is, however, often unnecessarily detailed. In all examples in the following evaluations, there is no joining of alternative operations or sequences in which token start times can be mixed, as in the example above. This means that the optimal solution is also obtained when a simplified linear order between the tokens is forced in OptConcurSeq, based on the index ℓ such that

$$s_{i,j,\ell+1} \geq s_{i,j,\ell}. \tag{8}$$

To better understand the complexity reduction of this order restriction, consider the makespan optimization of the TPN $\mathcal{N}_2$ in Fig. 6. The OptConcurSeq formulation is related to the equivalent safe TPN $\mathcal{N}_3$ in Fig. 8 with $\Delta_{1\ell} = \Delta_1$ and $\Delta_{2\ell} = \Delta_2$, $\ell = 1, \ldots, m$. In this TPN, the state of the individual tokens is introduced, while only the number of tokens is considered in $\mathcal{N}_2$. For instance, one token in the final place in $\mathcal{N}_2$ is in $\mathcal{N}_3$ represented as one token in the final place in one of the m sequences. When the second transition is excluded in $\mathcal{N}_2$, the total number of discrete states (different marking vectors) is $m + 1$, while $\mathcal{N}_3$ has $\sum_{k=0}^{m} \binom{m}{k} = 2^m$ corresponding states when the second transition in each of the m sequences is excluded. Thus, the number of states then increases linearly in $\mathcal{N}_2$ but exponentially in $\mathcal{N}_3$. The same number of states as in $\mathcal{N}_2$ is achieved in this example when the linear order Eq. 8 is introduced in $\mathcal{N}_3$. In the OptConcurSeq optimization formulation, we remind readers that no explicit states are introduced, but including the linear order Eq. 8 in OptConcurSeq also simplifies computations. Based on this simple analysis, and even more so from the evaluations in Sections 3.2 and 4, adding the order restriction Eq. 8 to OptConcurSeq reduces execution time significantly when multiple tokens are introduced. The importance of this token order restriction is also highlighted by the following proposition.

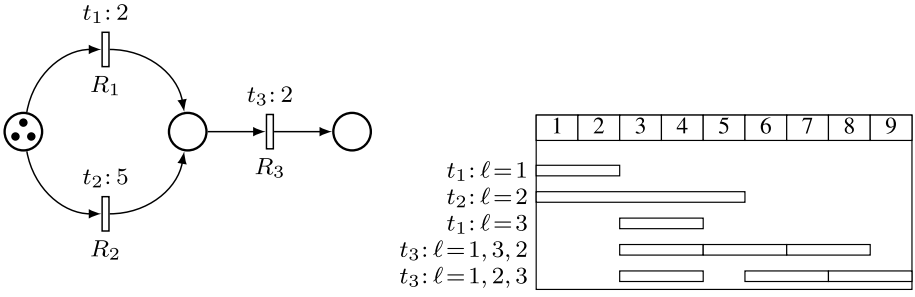

Fig. 9 A TPN with start times $s^1(t_1) = 0$, $s^2(t_2) = 0$, and $s^3(t_1) = 2$ for the three tokens in the first two alternative transitions and start times $s^1(t_3) = 2$, $s^3(t_3) = 4$, and $s^2(t_3) = 6$ in the joined third transition for the optimal token sequence with $MS = 8$. An ℓ ordered but non-optimal token sequence with $MS = 9$ is achieved with the start times $s^1(t_3) = 2$, $s^2(t_3) = 5$, and $s^3(t_3) = 7$

Proposition 1 *Consider the optimization formulation OptConcurSeq, and assume furthermore that any alternative resources have the same duration. The optimal makespan is then also achieved when the linear token-order restriction* $s_{i,j,\ell+1} \geq s_{i,j,\ell}$ *is included in OptConcurSeq.*

Proof For each individual transition $t_{i,j}$, the duration and competition for shared resources are the same for all multiple tokens in OptConcurSeq, as illustrated in Fig. 8 when $\Delta_{1\ell} = \Delta_1$ and $\Delta_{2\ell} = \Delta_2$, $\ell = 1, \ldots, m$. According to Example 4, it is also crucial that the duration is the same among alternative resources. Altogether, this means that the order in which multiple tokens pass the individual transitions $t_{i,j}$ does not influence the makespan. Thus, a simplified linear order in ℓ also yields an optimal makespan. □

3.2 Evaluation of OptConcurSeq

In the following two examples, the efficiency of the proposed CP-SAT solver is illustrated by a benchmark test. It is based on the symbolic OptConcurSeq optimization formulation, where CP-SAT is compared with two other powerful strategies.

Example 5 Different numbers of concurrent sequences *nSeq* and operations *nOp* will now be evaluated, where each sequence includes only one token and each resource has capacity one. Let the number of operations *nOp* in each sequence be equal and an even number, and let every second operation be performed by a resource that is shared among all sequences. A TPN for such a sequence is shown in Fig. 10, where in this example the number of tokens in each initial place $m = 1$. The duration for each resource is a random integer number with equal distribution in the interval [1, 100], and each shared resource R_k has the same duration Δ_k, $k = 1, 3, \ldots, nOp - 1$ in all sequences. The local resources $R_{i,2}, R_{i,4}, \ldots, R_{i,nOp}$ in each individual sequence *i* have no impact in this example, while they are relevant in the next example based on the same model structure, but then extended with multiple tokens $m > 1$.

Since all resources have capacity one, the cumulative constraint in Eq. 7 is simplified to a noOverlap condition, which for all $i, i' \in \{1, \ldots, nSeq\}$ and $j, j' \in \{1, \ldots, nOp\}$ can be expressed by the disjunctive condition

$$(s_{i,j} \geq e_{i',j'} \vee s_{i',j'} \geq e_{i,j}) \wedge i < i' \wedge r_{i,j} = r_{i',j'} \tag{9}$$

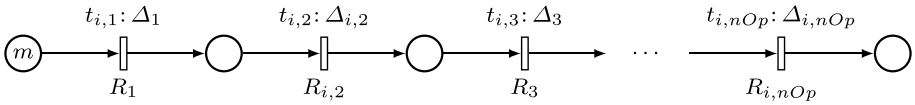

Fig. 10 One sequence *i* in a TPN with an even and equal number *nOp* of operations in each sequence. The initial place in each sequence has *m* tokens, and every second operation has a resource $R_1, R_3, \ldots, R_{nOp-1}$, respectively. These resources are shared among all sequences, while the rest of the operations have local resources $R_{i,2}, R_{i,4}, \ldots, R_{i,nOp}$ for each individual sequence *i*

Table 1 Optimal makespan *MS* and execution times (sec) using CP-SAT, CP-SAT-disjunctive, MILP, and Z3Opt for different numbers of sequences *nSeq* and operations *nOp* in Example 5. Time out (T.O.) is 200 sec

nSeq	*nOp*	*MS*	CP-SAT	CP-SAT-disjunctive	MILP	Z3Opt
3	250	12595	0.12	0.33	0.04	2.82
3	500	25430	1.90	1.60	1.22	101
3	1000	51357	3.90	3.84	1.98	T.O.
3	2000	99939	66.8	44.4	15.5	T.O.
6	50	3094	0.15	0.17	0.32	8.21
6	100	5605	0.60	0.51	1.31	41.4
6	150	8032	12.3	2.61	9.15	T.O.
6	200	10487	91.8	37.2	T.O.	T.O.
10	10	934	0.085	0.78	7.03	30.0
10	20	1553	7.76	7.91	68.7	T.O.
10	40	2683	11.2	13.1	89.7	T.O.
10	60	3720	13.7	5.21	94.4	T.O.
10	100	5818	T.O.	150	T.O.	T.O.

where the third token index ℓ is excluded, since only one token is involved in all sequences, i.e. $\ell = 1$. In Table 1 the makespan MS and execution times[1] for four different implementations are compared, 1) CP-SAT with its built-in function noOverlap, 2) CP-SAT with the disjunctive constraint Eq. 9 called CP-SAT-disjunctive, 3) a formulation corresponding to OptConcurSeq using Gurobi's MILP solver (Gurobi 2015), also including the disjunctive constraint Eq. 9, and 4) the same formulation as 2) but now using the SAT/SMT based solver Z3Opt (Bjørner and Phan 2014).

Since CP-SAT only includes discrete variables, the same type of variables are also used for the decision variables in Z3Opt and Gurobi's MILP solver. In the majority of the evaluated cases, including Example 6, the MILP solver is also shown to be more efficient when using discrete start and end times rather than the corresponding continuous variables. Since Gurobi clearly states that continuous variables are much more efficient, the result may be a bit surprising, but caused by the fact that all durations are integer values. However, the makespan to be optimized is treated as a continuous variable, as it is more efficient in a majority of the evaluated cases.

For three sequences ($nSeq = 3$), the evaluation shows that the MILP formulation is competitive, and even faster than CP-SAT. An increased number of sequences means, on the other hand, that CP-SAT becomes more efficient than MILP, especially for ($nSeq = 10$). It is also interesting to see that the built-in noOverlap function is slightly better than the disjunctive constraint Eq. 9 for smaller problems, while this hardcoded constraint, somewhat surprisingly, works clearly better than the built-in noOverlap function when the complexity increases. Even more remarkable is that Gurobi's MILP solver is more efficient than Z3Opt across all evaluated cases. This conclusion is in clear contrast to previously reported results, which state that Z3Opt is typically about 10 times faster than Gurobi's MILP solver (Roselli et al. 2018) for similar scheduling problems. As a final remark, an A* implementation was also evaluated on this problem in Lennartson (2024) for $nSeq = 3$, showing that A* was much less efficient than the solvers evaluated in this paper.

[1] The execution times in Tables 1-4 are median times.

Multiple tokens In the following example, CP-SAT and Gurobi's MILP solver are further evaluated when multiple tokens are also included. The benefit of including constraint Eq. 8 is also demonstrated. Since each operation also involves a resource that is shared among other sequences or only among multiple tokens in a single sequence, the token order is further restricted by the extended and stronger condition

$$s_{i,j,\ell+1} \geq e_{i,j,\ell} \tag{10}$$

where the start time $s_{i,j,\ell}$ in Eq 8 is extended to the end time $e_{i,j,\ell}$. This means that the costly evaluation of the noOverlap condition Eq. 9 does not need to be included for local resources that are only shared among multiple tokens in the single operation j. Although the constraint Eq. 8 has the most significant impact, the extended condition Eq. 10 implies further execution time reduction.

Example 6 A number of tokens are now introduced ($m > 1$ in Fig. 10), while the number of sequences $nSeq = 2$ and the number of operations in each sequence is $nOp = 10$. The resource capacity is still equal to one, but is now also active for the resources only involved in one of the sequences, due to the multiple tokens. In the same way as in Example 5, integer variables are used in the MILP solver for the start and end times, while the makespan is represented by a continuous variable. The individual start times for each token means that each token has an individual identity, implemented by the index $\ell \in Tok(i)$ where $i \in Seq$. Thus, the optimization formulation OptConcurSeq, without the constraint Eq. 10, is a TPN in which the identity of each token is traced. Adding Eq. 10 as a constraint reduces the complexity of the optimization by introducing an order among the tokens, while still yielding an optimal makespan, since no alternative resources are involved in this example.

The results in Table 2 show that keeping the order between the individual tokens by constraint Eq. 10 reduces the execution time dramatically when the number of tokens increases. This is indeed a significant benefit of using TPNs instead of modular timed automata, one for each individual token, when tracing the identity of each part (token) is not required. Table 2 also shows that the efficiency of CP-SAT is much less sensitive to the number of tokens than Gurobi's MILP solver, which has nearly the same execution time for 12 tokens as CP-SAT has for 400 tokens. Although the order between the individual tokens is kept by the constraint Eq. 10, each token is still implemented as an individual sequence. As noted in Example 5, we observed greater sensitivity to the number of sequences for MILP than

Table 2 Optimal makespan *MS* and execution times (sec) using CP-SAT, including and not including constraint Eq. 10, and Gurobi's MILP solver including constraint Eq. 10 for different number of tokens *nTok* in Example 6. Time out (T.O.) is 200 sec

m	*MS*	CP-SAT incl. Eq. 10	CP-SAT not incl. Eq. 10	MILP incl. Eq. 10
5	1102	0.012	0.72	0.030
10	1777	0.032	16.1	4.52
12	2047	0.044	47.4	38.5
14	2317	0.064	58.5	T.O.
20	3127	0.134	T.O.	T.O.
50	7177	1.19	T.O.	T.O.
100	13927	6.25	T.O.	T.O.
200	27427	23.0	T.O.	T.O.
400	54427	44.8	T.O.	T.O.

for CP-SAT. It is therefore no surprise that the computation time increases more rapidly for MILP than for CP-SAT when the number of tokens increases. The extreme difference in favor of CP-SAT is, however, impressive.

Generally, an additional number of tokens and sequences increases concurrency, resulting in more possible states. An additional number of operations in existing sequences, on the other hand, leads to a deeper problem, with longer sequences before reaching a goal state. The conclusion based on Example 5 and Example 6 is therefore that MILP can handle deep problems competitively, while CP-SAT is computationally much more efficient than MILP, particularly with increased concurrency.

3.3 Mixed alternative and concurrent sequences

So far, TPNs, including a set of concurrent sequences, have been considered, with shared and alternative resources potentially involved. It is also necessary to handle alternative complete sequences, not only alternative resources in individual operations. In CP-SAT, this can be easily achieved in a similar way as for alternative resources. By using the exactOne function, cf. Eq. 2, and assigning $d_{i,j,\ell} = 0$ to all operations in the alternative sequence(s), not selected by the exactOne function, only one of the alternative sequences at a time is included in the makespan evaluation. In the final minimization, all possible alternative sequences are evaluated, and the one with the shortest duration is chosen.

In the next optimization formulation, called OptConcurAltSeq, both concurrent and a set of alternative sequences $Seq_{\oplus} \subseteq Seq$ are included. The sequence $i \in Seq_{\oplus}$, which together with the other concurrent sequences $Seq \setminus Seq_{\oplus}$ minimizes the makespan MS, will be chosen. The tokens involved in the choice among the alternative sequences $i \in Seq_{\oplus}$ are defined by a shared token set $Tok_{\oplus}$, such that $Tok(i) = Tok_{\oplus}$ for all $i \in Seq_{\oplus}$.

Optimization formulation OptConcurAltSeq Same as OptConcurSeq with Eq. 3 replaced by the constraints

$$i \in Seq_{\oplus} \rightarrow \beta_{i,\ell} = \text{exactOne}([\beta_{\iota,\ell}]_{\iota \in Seq_{\oplus}}) \tag{11}$$

$$i \in Seq_{\oplus} \wedge \beta_{i,\ell} \wedge b_{\nu} \rightarrow (r_{i,j,\ell} = R_{i,j,\nu}) \wedge (d_{i,j,\ell} = \Delta_{i,j,\nu}) \tag{12}$$

$$i \in Seq_{\oplus} \wedge \neg\beta_{i,\ell} \rightarrow d_{i,j,\ell} = 0 \tag{13}$$

$$i \notin Seq_{\oplus} \wedge b_{\nu} \rightarrow (r_{i,j,\ell} = R_{i,j,\nu}) \wedge (d_{i,j,\ell} = \Delta_{i,j,\nu}) \tag{14}$$

In Eq. 11, $[\beta_{\iota,\ell}]_{\iota \in Seq_{\oplus}}$ is a list of Booleans where each $\beta_{\iota,\ell}$ corresponds to one of the alternative sequences in $Seq_{\oplus}$, i.e. different sequences may be chosen for each individual token $\ell \in Tok_{\oplus}$. Among the alternative sequences, only one sequence $i \in Seq_{\oplus}$ will be chosen, implying that $\beta_{i,\ell} = \top$, and the corresponding operation durations and resources are assigned in Eq. 12. The rest of the Booleans $\beta_{\iota,\ell} = \bot$, $\iota \in Seq_{\oplus} \setminus \{i\}$, due to the exactOne function in Eq. 11, and the corresponding durations are all assigned to zero in Eq. 13. The remaining sequences $Seq \setminus Seq_{\oplus}$ are assigned in Eq. 14 in the same way as Eq. 3 in

OptConcurSeq. This optimization formulation is easily generalized to multiple sets of alternative sequences.

Before this extended optimization formulation is evaluated, we observe in the following corollary that the token order restriction Eq. 8 can also be included in OptConcurAltSeq when alternative sequences are not joined.

Corollary 1 *Consider the optimization formulation OptConcurAltSeq, and assume furthermore that alternative sequences are not joined and any alternative resources have the same duration. The optimal makespan is then also achieved when the linear token-order restriction* $s_{i,j,\ell+1} \geq s_{i,j,\ell}$ *is included in OptConcurAltSeq.*

Proof This statement follows from the same arguments as in Proposition 1, adding the fact that if sequence $\beta_{i,\ell} = \perp$ the order restriction is trivially satisfied, since $d_{i,j,\ell} = 0$ according to Eq. 13. □

With the same motivation as before Example 6, the token-order restriction $s_{i,j,\ell+1} \geq s_{i,j,\ell}$ can also be extended to $s_{i,j,\ell+1} \geq e_{i,j,\ell}$.

Example 7 Consider the two alternative sequences in Fig. 11, in which the resource capacity tokens return after one additional timed transition. Since each resource that is booked for an operation *j* is also occupied during operation $j+1$, the end time in the interval $[s_{i,j,\ell}, e_{i,j,\ell}]$ in the cumulative constraint function Eq. 7 needs to be extended to $e_{i,j+1,\ell}$. Similarly, the end time of all other competing primed time intervals is extended. More specifically, $e_{i,j,\ell}$ is replaced by $e_{i,j+1,\ell}$ and $e_{i',j',\ell'}$ is replaced $e_{i',j'+1,\ell'}$ in Eq. 7. In the same way, the first replacement is also performed in Eq. 10. Resulting makespan MS and execution times are shown in Table 3 with and without the order restriction Eq. 10, which also here confirms the importance of including this constraint.

This TPN is closely related to a flexible manufacturing example in Marino et al. (2020), where a TPN-based MILP formulation including explicit marking states is proposed. In one of the evaluations, the durations are $\Delta_{1,j} = \Delta_{2,j} = k$ (the alternative resource index is here excluded), which for an arbitrary number of tokens m gives $MS = 3(m+1)$. The execution times, including constraint Eq. 10 in Table 3, are about

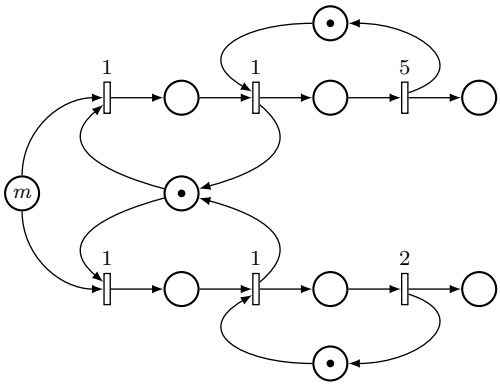

Fig. 11 A TPN with two alternative sequences and resource allocations, each one involving two operation durations

Table 3 Optimal makespan *MS* and execution times (sec) using CP-SAT, including and not including constraint Eq. 10 for different number of tokens *m* in Example 7. Time out (T.O.) is 200 sec

m	*MS*	Incl. Eq. 10	Not incl. Eq. 10
5	14	0.01	0.01
10	25	0.02	38.6
15	37	0.58	T.O.
20	49	5.57	T.O.
25	60	27.8	T.O.

500 times shorter than the results based on the MILP formulation with explicit marking states reported in Marino et al. (2020).

3.4 Start and completion of local sequences

Alternative and concurrent sequences do not always start at the beginning and terminate before all sequences are completed. Such local sequences are still modeled as ordinary sequences and included in the total set of sequences *Seq*, but an additional start constraint is added to each local sequence.

Local alternative sequences Assume that a sequence $i_1 \in Seq$ is followed by a set $Seq_{\oplus} \subseteq Seq$ of local alternative sequences. When the optimal sequence in $Seq_{\oplus}$ has been completed, another sequence $i_2 \in Seq$ is assumed to follow. This behavior is achieved by adding the constraints

$$\bigvee_{\iota \in Seq_{\oplus}} s_{\iota,1,\ell} \geq e_{i_1,nOp(i_1),\ell} \qquad \text{and} \qquad \bigvee_{\iota \in Seq_{\oplus}} s_{i_2,1,\ell} \geq e_{\iota,nOp(\iota),\ell} \tag{15}$$

to OptConcurAltSeq. Note that the choice of the optimal alternative sequence in $Seq_{\oplus}$ is obtained by Eqs. 11-13. Furthermore, recall that continuing a sequence after completion of alternative sequences means that the token order restriction Eq. 8 cannot generally be used. The reason is that joining alternative sequences with mixed token start times may yield non-optimal makespan computations, as illustrated in Example 4.

Local concurrent sequences Replace the set $Seq_{\oplus}$ in Eq. 15 with a set of local concurrent sequences $Seq_{\|} \subseteq Seq$. Local concurrent sequences are then obtained by replacing the disjunction operator in Eq. 15 with a conjunction over all local sequences in $Seq_{\|}$. This follows since all sequences $\iota \in Seq_{\|}$ can start when the sequence i_1 before the split is completed. Furthermore, the join into sequence i_2 can start first when all local concurrent sequences $\iota \in Seq_{\|}$ are completed. This fact also means that the token-order restriction Eq. 8 can be used when concurrent sequences are joined. Finally, note that both local alternative and concurrent sequences can be generalized to corresponding multiple sets of local sequences.

Local alternative sequences will be illustrated in the following manufacturing example, where additional modeling features are also discussed, including weighted arcs, uncontrollable events, and cyclic behavior.

4 Optimization of a manufacturing system

The OptConcurAltSeq optimization formulation will now be evaluated on a realistic manufacturing application, including flows of multiple tokens in both concurrent and alternative sequences. The application is a flexible manufacturing system (FMS), presented in de Queiroz et al. (2005) as a set of modular automata. In Pena et al. (2022), this model is extended to include operation durations, and an efficient near-optimal time-optimization procedure is proposed to compute the optimal makespan for the modular timed automata model. A corresponding TPN is shown in Fig. 12 where each timed transition has a resource label with a capital letter and a duration, but also a unique transition label t_{ij}, where the first index means sequence *i* and the second one means operation *j*. The shared resources are a robot *R*, a lathe *L*, a conveyor C_3, and an assembly machine *AM*, while the local resources only used by one operation are two conveyors C_1 and C_2, a mill *M*, and a painting device *PD*.

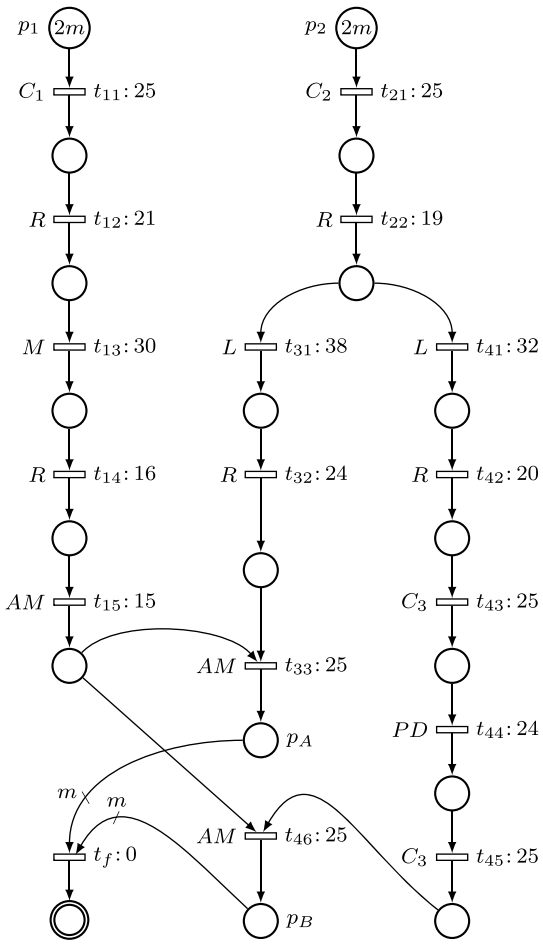

Fig. 12 A TPN for a flexible production system where *m* products of type p_A and p_B are produced based on $2m$ input parts of type p_1 and p_2

Concurrent and alternative sequences To handle the concurrency and the alternative sequences in Fig. 12, the system model Sys is divided into four sequences, $i = 1$ including the five transitions $t_{11} - t_{15}$, $i = 2$ including the two transitions t_{21} and t_{22}, $i = 3$ including the three transitions $t_{31} - t_{33}$, and $i = 4$ including the six transitions $t_{41} - t_{46}$. The two local alternative sequences $i = 3$ and $i = 4$ follow after the completion of sequence $i = 2$, and all three are executed concurrently with the first sequence. The alternative between $i = 3$ and $i = 4$ also decides the choice between transition t_{33} for $i = 3$ or t_{46} for $i = 4$, after the completion of sequence $i = 1$.

Production goal specified by weighted transitions Two types of products p_A and p_B are produced in this FMS, based on input parts of type p_1 and p_2. The number of input parts is 2*m*, and *m* products of both types are produced. Thus, the input places p_1 and p_2 in Fig. 12 have 2*m* initial tokens, and the goal is to generate *m* tokens in both places p_A and p_B. When this goal is achieved, the last zero-time transition t_f can be fired, leaving only one token in the double-circle place. As with automata, this demonstrates the possibility of graphically specifying not only the initial state but also the final goal state in a PN, a single double-circle place that receives one token when the final marking vector is reached.

Weighted transitions in CP-SAT Weighted transitions are normally added to specify logical conditions, such as the goal specification on an equal number of tokens in the places p_A and p_B. Since CP-SAT is a high-level programming language with a number of sophisticated functions, such logical conditions are often easier to formulate directly in CP-SAT, instead of first translating a logical condition to a PN specification. This argument is the same as in Lennartson et al. (2014), where we recommend adding variables to PNs, and adding logical transition conditions on these variables instead of modeling such conditions graphically in the PN formalism.

In CP-SAT, the specification on an equal number of tokens in the places p_A and p_B is simply formulated as the constraint, cf. Eqs. 11, 12

$$\sum_{\ell=1}^{2m} \beta_{3,\ell} = m. \tag{16}$$

The Boolean $\beta_{3,\ell}$, where 3 specifies the third sequence that produces product p_A, is interpreted as a 1/0 integer when it is used in summations.

Limited buffers The maximum number of tokens in the places in Fig. 12 only depends on the number of tokens in the initial places. Each place between timed transitions can be considered a buffer, and in the original model in de Queiroz et al. (2005) these buffers are limited to be of size one. This limitation is easily handled by adding the constraint $e_{i,j,\ell+1} \geq s_{i,j+1,\ell}$, which means that for token ℓ the next operation $j + 1$ must start at the latest when the next token $\ell + 1$ in the current operation *j* is completed.

Uncontrollable events In both de Queiroz et al. (2005) and Pena et al. (2022), it is assumed that the event related to the start of an operation is controllable, while the corresponding

event related to the completion of an operation is uncontrollable, a very common assumption in Supervisory control examples. The constraint on the end time of the current operation must then be moved to the more conservative constraint on the start time of the next operation. This results in the uncontrollability constraint

$$s_{i,j,\ell+1} \geq s_{i,j+1,\ell} \tag{17}$$

Since the exit buffers after the mill *M*, the lathe *L*, the painting device *PD*, and the third conveyor C_3 are the same as the corresponding input buffers, the following additional constraints are also required

$$s_{1,2,\ell+1} \geq s_{1,4,\ell}, \quad s_{2,2,\ell+1} \geq \beta_{3,\ell}\, s_{3,2,\ell} + (1-\beta_{3,\ell})\, s_{4,2,\ell}, \quad s_{4,2,\ell+1} \geq s_{4,6,\ell} \tag{18}$$

Optimal solution comparison Adding the constraints Eqs. 15-18 to OptConcurAltSeq, together with the resource and duration information in Fig. 12 and the token order restriction Eq. 10, implies that the optimization formulation can be solved by CP-SAT. The resulting optimal makespan MS and computation time for different values of m are shown in Table 4. These results are compared with the solution generated by an evolutionary clonal selection algorithm (de Castro and Zuben 2002) with DS mutation, evaluated for the FMS in Pena et al. (2022), and then called CSA-DS. For $m = 10$, the near-optimal CSA-DS algorithm takes more than 3 times longer than the optimal CP-SAT algorithm, and the optimal solution, first reported in Malik and Pena (2018), takes 20 times longer to compute. It is, however, interesting that an approach similar to the one used in this paper is employed, based on model checking, to find the minimum total duration that yields a feasible solution. The symbolic state-space representation is, in that case, a binary decision diagram (Bryant 1992).

Linear increasing makespan The optimal makespan is shown in Table 4 for $m \leq 10$. For higher *m* values, the computation time increases very quickly, a typical behavior of all SAT-based solvers when the system complexity becomes too large. The question is only when this computational limitation occurs, and for CP-SAT, it occurs for much higher system complexity than, for instance, Z3Opt. It is, however, interesting that a simple pattern on the optimal makespan is immediately obtained for $m \geq 1$, where

$$MS = 157m + 81$$

This formula yields the optimal makespan for $m \leq 10$, and although there is no formal proof, the result indicates that this behavior will also hold for larger *m*. It is supported by the fact that the structure of the TPN exhibits a cyclic behavior as *m* increases.

Table 4 Optimal makespan *MS* for the TPN in Fig. 12 and computation times *CT* using CP-SAT for different number of products *m*

m	1	2	3	4	6	8	10
MS	238	395	552	709	1023	1337	1651
CT (sec)	0.007	0.030	0.090	0.199	2.24	21.1	90.4

Summary The evaluations of the proposed time optimization method show that CP-SAT is a flexible modeling language for optimization problems and is very efficient compared to alternative optimization strategies. To better understand the reason for this, we will, in the next section, show how modern SAT solvers work, and especially how CP and SAT can be integrated, and even be supported by more traditional OR methods.

5 CP-SAT - an integrated satisfiability and constraint programming solver

One of the main principles involved in satisfiability (SAT) solvers is conflict-driven clause learning (CDCL) (Marques-Silva and Malik 2018). This principle, which is based on Boolean variables, has more recently been extended to integers in Constraint Programming (CP) (Feydy and Stuckey 2009). A brief survey of these principles is provided in this section, along with comments on their implementation in CP-SAT. Before detailing CDCL in SAT and CP, we remind readers that an optimization problem in CP-SAT is transformed into a satisfiability problem by simply introducing a constraint on the optimization criterion. This constraint can be decreased until the solution becomes unsatisfiable, or increased until it becomes satisfiable. The lowest value of this constraint that results in a satisfiable (feasible) solution then generates the optimal solution.

5.1 Satisfiability solvers

The main functionality of an efficient SAT solver will be briefly illustrated in this subsection. A more detailed presentation can be found in the handbook on SAT (Marques-Silva et al. 2009) and the handbook on model checking (Marques-Silva and Malik 2018). A SAT solver evaluates if a propositional logic formula can be satisfied or not. Well known applications are model checking (Biere and Kröning 2018) and planning (Rintanen 2009), but also optimization (Bjørner and Phan 2014; Chen 2020).

Conjunctive normal form The input format for formulas to be evaluated by SAT solvers is the conjunctive normal form (CNF). A CNF formula ψ consists of the conjunction of a set of *clauses* $\{c_1, \ldots, c_n\}$, where each clause c_i includes a disjunction of a set of literals $\{\ell_{i1}, \ldots, \ell_{im_i}\}$, and each literal ℓ_{ij} is a Boolean variable p_{ij} or its negation $\neg p_{ij}$. For convenience, the negation of p is also denoted $\bar{p}$. The assignment $p = \bot$ then corresponds to the literal $\ell = \bar{p}$, meaning that the complement $\bar{\ell} = p$. Altogether, a CNF formula can generally be expressed as

$$\psi = \bigwedge_{i=1}^{n} c_i = \bigwedge_{i=1}^{n} \bigvee_{j=1}^{m_i} \ell_{ij}$$

As a simple illustration, we see that a biconditional statement is easily reformulated as a CNF formula, since $p \leftrightarrow q \equiv (p \rightarrow q) \wedge (q \rightarrow p) \equiv (\bar{p} \vee q) \wedge (\bar{q} \vee p)$. In Ben-Ari (2012), a couple of rules are given to transform a general Boolean formula to CNF.

Unit propagation and pure literal elimination A CNF formula ψ has important properties that can be used as inference rules in satisfiability evaluations. Since ψ is satisfied only when all of its clauses are satisfied, the following rules are particularly useful:

i) When ψ includes a *unit clause* that only has a single literal ℓthere is no restriction to assign $\ell = \top$, which implies that every clause in ψ including ℓ is also satisfied, and can therefore be neglected.
ii) for every clause c including the complement $\bar{\ell} = \bot$, this literal can also be neglected in c, since it does not contribute to the satisfaction of c.

2. When ψ includes a Boolean variable p with only one polarity, either p or $\bar{p}$, the corresponding literal ℓ is called a *pure literal*, and every clause including ℓ can be neglected. This is called *pure literal elimination*.

The first rule is called the *unit clause rule*, and repeating this rule is called *unit propagation*, but also *Boolean constraint propagation (BCP)* (Davis et al. 1962). When no unit clause currently exists in unit propagation and unassigned literals remain, an external decision is taken to assign one of the unassigned literals by some heuristic rule (Moskewicz et al. 2001). Unit propagation is illustrated on the CNF formula

$$\psi = (\bar{p} \vee q) \wedge (\bar{p} \vee \bar{q} \vee r) \wedge (\bar{q} \vee \bar{r}) \stackrel{\text{def}}{=} c_1 \wedge c_2 \wedge c_3 \tag{19}$$

Since there is no unit clause in ψ, the decision $q = \top$ is taken as the first assignment. Then, $c_1 = \bar{p} \vee q \equiv \top$ can be neglected, and $c_3 = \bar{q} \vee \bar{r} = \bot \vee \bar{r} \equiv \bar{r}$ is a unit clause, which implies that $r = \bot$. Finally, $c_2 = \bar{p} \vee \bar{q} \vee r = \bar{p} \vee \bot \vee \bot \equiv \bar{p}$ is also a unit clause, such that $p = \bot$. Thus, the CNF formula ψ in Eq. 19 is satisfiable.

Learned clauses by conflicts In unit propagation, a number of conflicts often occur before a complete assignment of all variables is achieved, especially in cases where only a few assignments satisfy the formula. To illustrate this, let the decision $p = \top$ instead be the first assignment in Eq. 19. Then, $c_1 = \bar{p} \vee q = \bot \vee q \equiv q$, and unit clause gives $q = \top$, while $c_2 = \bar{p} \vee \bar{q} \vee r = \bot \vee \bot \vee r \equiv r$, and ones again unit clause gives $r = \top$. This results in the conflict $c_3 = \bar{q} \vee \bar{r} = \bot \vee \bot \equiv \bot$. Since the cause of this conflict is the assignment $p = \top$, we learn that this assignment must be avoided, and its negation becomes a *learned clause* $c_4 = \bar{p}$.

Backjumping The assignments above $p = \top$, $q = \top$, and $r = \top$, which generated the conflict $\bar{q} \vee \bar{r} = \neg(q \wedge r) = \bot$ and the learned clause $c_4 = \bar{p}$, are illustrated in the satisfiability search tree in Fig. 13. The learned clause results in a jump two levels back in the tree and a significant reduction in the search space. Unit propagation on Eq. 19, conjuncted with the learned clause $\bar{p} = \top$ such that

$$\psi \wedge \bar{p} \equiv \psi = (\bar{p} \vee q) \wedge (\bar{p} \vee \bar{q} \vee r) \wedge (\bar{q} \vee \bar{r}) \equiv \top \wedge \top \wedge (\bar{q} \vee \bar{r}) \equiv (\bar{q} \vee \bar{r}),$$

together with the decision $q = \top$, implies that $\psi \equiv \bot \vee \bar{r} \equiv \bar{r}$ and unit clause gives $r = \bot$. Hence, ψ is ones again satisfied for $p = \bot$, $q = \top$, and $r = \bot$, which is also illustrated in Fig. 13.

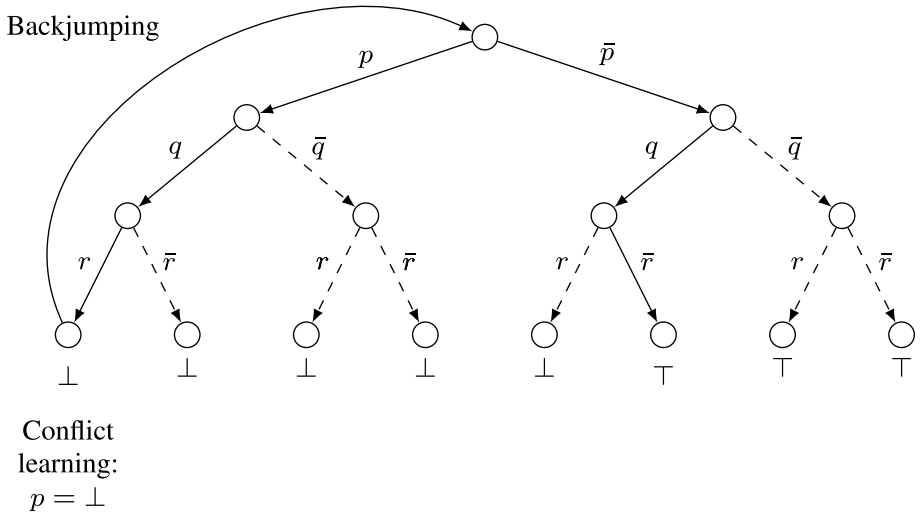

Fig. 13 Satisfiability search tree for the CNF-formula $(\bar{p} \vee q) \wedge (\bar{p} \vee \bar{q} \vee r) \wedge (\bar{q} \vee \bar{r})$, including the decision $p = \top$, followed by $q = \top$ and $r = \top$ caused by unit propagation. This results in a conflict, the learned clause $\bar{p}$, and backjumping. The learned clause $\bar{p}$, a second decision $q = \top$, and unit propagation finally give $r = \bot$

Conflict-driven clause learning and implication graphs The main principles in conflict-driven clause learning (CDCL) include heuristic search with inference rules based on unit propagation, clause learning, and non-chronological backtracking, also called backjumping, as illustrated in the small example above.

An interesting observation in unit propagation and clause learning is that the involved clauses are preferably reformulated as implications. By rewriting Eq. 19 as a conjunction of unit clauses expressed as implications and a conflict

$$\psi = (p \rightarrow q) \wedge ((p \wedge q) \rightarrow r) \wedge \neg(q \wedge r), \tag{20}$$

a graphical formulation of this expression, called an implication graph, is shown in Fig. 14. Every node except the conflicting node is then labeled with a literal, and input nodes joined into a single output node represent a unit clause. The input nodes are the premises in the implication, and the output node is the implied literal, assigned by unit propagation. The transitions from the input nodes to the output node are labeled by the unit clause. In a falsified clause, the nodes representing all literals in the falsified clause are joined in an output node labeled $\bot$.

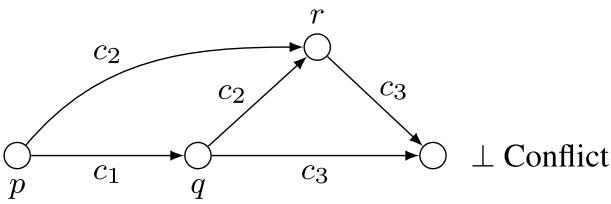

Fig. 14 Implication graph for the CNF formula ψ in Eq. 19, reformulated as the implications $(p \rightarrow q) \wedge ((p \wedge q) \rightarrow r)$ defined by unit propagation and the resulting conflict $\neg(q \wedge r)$

Based on these implication graphs, further analysis is performed to identify efficient learned clauses, including clause minimization. Efficient data structures, as well as suitable variable ordering, clause deletion, and restart procedures, are additional tools used in modern SAT solvers. These additional steps are described in Moskewicz et al. (2001) and Eén and Sörensson (2004), and the SAT surveys Marques-Silva et al. (2009) and Marques-Silva and Malik (2018).

5.2 Constraint programming

Constraint programming (CP) has functionality very similar to that of a SAT solver, with the fundamental difference that Boolean variables are replaced by integer variables. A CP problem is defined by a set of integer variables with given domains and a set of constraints on the variables. Both CP and SAT solvers combine search and inference. The aim here is to illustrate how CDCL, originally developed for SAT solvers, can also be generalized to CP problems.

Constraint propagation The domain of an integer variable x can be defined as an interval $x \in [\mathrm{lb}(x), \mathrm{ub}(x)]$ or, more generally, as a set D_x that is assumed to be finite. The goal is either to decide *feasible* values of the variables, i.e. values that satisfy the constraints, or to determine the optimal value(s) when an optimization criterion is also included. The main inference rule in CP is domain reduction based on *constraint propagation*, also called finite domain propagation and bounds propagation. For the usual addition constraint $y = x_1 + x_2$, also expressed as the subtractions $x_1 = y - x_2$ and $x_2 = y - x_1$, the propagators are

$$\begin{aligned} \mathrm{lb}(x_1) + \mathrm{lb}(x_2) \le y \le \mathrm{ub}(x_1) + \mathrm{ub}(x_2) \\ \mathrm{lb}(y) - \mathrm{ub}(x_2) \le x_1 \le \mathrm{ub}(y) - \mathrm{lb}(x_2) \\ \mathrm{lb}(y) - \mathrm{ub}(x_1) \le x_2 \le \mathrm{ub}(y) - \mathrm{lb}(x_1) \end{aligned} \tag{21}$$

The left inequality defines the lower bound and the right one the upper bound. The individual domains can be the original ones or reduced domains obtained after earlier constraint propagations.

From constraint propagation to conflict-driven clause learning Constraint propagation is naturally formulated as implications on Boolean variables by introducing the *order literals* $[\![x \le i]\!]$ and $[\![x \ge i]\!] \equiv \neg[\![x \le i-1]\!]$ (Petke and Jeavons 2011). These implications on Boolean expressions may then be interpreted as unit clauses, resulting in CDCL in the same way as for SAT solvers. The next example demonstrates this concept.

Example 8 Consider the constraint optimization problem

$$\min y \qquad \text{where} \qquad y = x_1 + x_2, \quad x_1 \ge 7 \vee x_2 \ge 9 \tag{22}$$

with the integer variable domains $y \in [0, 20]$, $x_1 \in [2, 20]$, and $x_2 \in [4, 20]$. The literal decision $[\![y \le 10]\!]$ is introduced to evaluate if a feasible solution is obtained. The upper limits on x_1 and x_2 in the second and third propagators in Eq. 21, together with the decision on y, generate the unit clauses c_1 and c_2 below. Furthermore, the disjunctive constraint

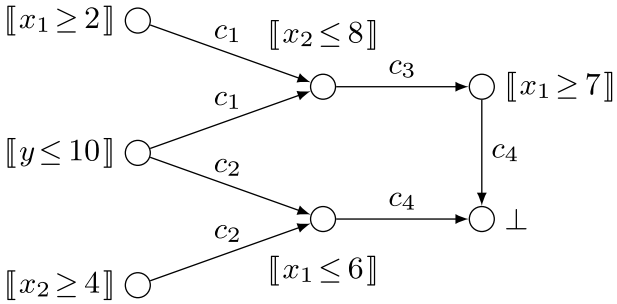

Fig. 15 Implication graph representing the constraints in Eq. 22, including the variable domains and the literal decision $[\![y \leq 10]\!]$

$x_1 \geq 7 \vee x_2 \geq 9 \equiv \neg(x_2 \leq 8) \vee x_1 \geq 7 \equiv x_2 \leq 8 \rightarrow x_1 \geq 7$ generate the unit clause c_3, while the conflict c_4 is generated by the conjunction of the conclusions in c_2 and c_3.

$$
\begin{aligned}
c_1 &: [\![\, y \leq 10 \,]\!] \wedge [\![\, x_1 \geq 2 \,]\!] \rightarrow [\![\, x_2 \leq 8 \,]\!] \\
c_2 &: [\![\, y \leq 10 \,]\!] \wedge [\![\, x_2 \geq 4 \,]\!] \rightarrow [\![\, x_1 \leq 6 \,]\!] \\
c_3 &: [\![\, x_2 \leq 8 \,]\!] \rightarrow [\![\, x_1 \geq 7 \,]\!] \\
c_4 &\; [\![\, x_1 \leq 6 \,]\!] \wedge [\![\, x_1 \geq 7 \,]\!]
\end{aligned}
$$

An implication graph representing these implications, including the resulting conflict, is shown in Fig. 15. The conjunction of the inputs in this graph generates the conflict, and the negation

$$\neg\big([\![x_1 \geq 2]\!] \wedge [\![x_2 \geq 4]\!] \wedge [\![y \leq 10]\!]\big) \equiv [\![x_1 \geq 2]\!] \wedge [\![x_2 \geq 4]\!] \rightarrow [\![y \geq 11]\!]$$

results in the learned literal $[\![y \geq 11]\!]$. Then, sharpen the decision to $[\![y = 11]\!]$ generates the lowest feasible, and therefore optimal, solution $y = 11$. □

This small example illustrates that a CP problem can be transformed into a SAT problem by introducing order literals. It implies that, since heuristic search in SAT problems can be improved by the CDCL algorithm and efficient data structures (see the end of Section 5.1), the same concepts can also be applied to heuristic search in CP problems.

Lazy clause generation Based on the similarity between tree search in SAT and CP, the CDCL algorithm has been generalized to CP problems by P.J. Stuckey and his team in Melbourne (Feydy and Stuckey 2009; Ohrimenko et al. 2009). Their generalized approach, called *lazy clause generation* (LCG), is said to be a hybrid of SAT and CP, in which SAT-based methods are applied directly to constraint propagators, as illustrated in the example above, inspired by the main example in Stuckey (2010). The notion of laziness arises because literals may be generated on demand when propagators include specific intervals or values. The motivation for this laziness is that many clauses may be required to represent an integer constraint, but in practice, most are never used. Lazy clause generation means that only the ones required are generated, and only when needed.

Different encodings from CP to SAT The *order literal encoding* in the transformation from CP to SAT was proposed already by Crawford and Baker (1994) when they solved scheduling problems by SAT algorithms. In a more recent contribution, it was shown both theoretically and practically in Petke and Jeavons (2011) that order encoding is superior to the

better-known direct and logarithmic encodings when a constraint satisfaction problem (CSP) is converted to SAT. Direct encoding means that each integer value $x = i$ is represented by an equality literal $[\![x = i]\!]$. In LCG, both order and equality literals are used. Depending on the domain size, different versions of laziness are proposed. Either only equality literals are generated on demand, or both equality and order literals are generated when needed (Feydy and Stuckey 2009).

Important influences in the development of CP-SAT In a recent Google presentation, one of the developers (Perron 2023) mentions three important influences on their development of CP-SAT. The first one is the breakthrough on CDCL in the SAT community around the turn of the millennium, the second one is the implementation made by Stuckey's group, called Chuffed (Chuffed 2024) together with Stuckey's presentation "Search is Dead" (Stuckey 2013), and the third one is the reboot of integrating integer linear programming, constraint programming, and MaxSAT on top of SAT. The first two topics have been briefly presented in this section, which now concludes with some short remarks on the third topic.

Integration between operation research, constraint programming, and SAT When CP includes optimization, constraint propagation and backpropagation are complemented by branch and bound search. Lower and upper bounds on the optimization criterion are then critical, and these bounds are typically obtained by relaxation, including linear programming and cutting planes (Hooker 2012). A complementary SAT-based direction is to integrate MaxSAT and pseudo-Boolean constraints (PBCs) as an interface between pure SAT solvers and the original optimization formulation. MaxSAT seeks an assignment that maximizes the number of satisfied clauses, whereas PBCs involve linear inequalities or equations over Boolean variables with integer coefficients. An overview of MaxSAT and PBC is given in Marques-Silva (2012).

To summarize this section, we first recall that CP-SAT was shown to be much faster than the SAT/SMT-based solver Z3Opt when evaluating larger jobshop scheduling problems in Section 3. This can be explained by the fact that CP-SAT integrates SAT with CP, including constraint propagation and global constraints, as well as relaxation based on simplex and linear programming. Thus, Hooker's strong recommendation to merge search, inference, and relaxation (Hooker 2012) has been successfully realized and implemented in CP-SAT.

6 Concluding remarks

An optimization formulation for TPNs has been introduced in this paper. Compared to earlier MILP formulations, no explicit global marking states are introduced. The dynamic state information is implicitly determined by constraints on the local start and end times of the involved tokens in each timed transition. This yields an easy, general optimization formulation, and by including specific constraints, valid only for optimizing a particular but common type of Petri nets, the computation time is drastically reduced. It is also shown how typical examples of uncontrollable events, which can be abstracted for modular automata but not for Petri nets with multiple tokens, can be handled in the optimization formulation. A topic for further study is how this formulation can be generalized to arbitrary uncontrollable

event scenarios. The final conclusion is that the impressive performance of CP-SAT can be explained by the strong integration of pure logics, high-level constraints, smart search strategies, and well-known relaxation tools from operations research, all included in a single implementation.

Acknowledgements Many thanks to Dr. Sabino Roselli for his support in using Z3Opt. The author is also very thankful to the reviewers for their important comments and suggested improvements of the manuscript.

Author Contributions Only contribution from the first and only author.

Funding Open access funding provided by Chalmers University of Technology. No funding was received to assist with the preparation of this manuscript.

Data Availability No datasets were generated or analysed during the current study.

Declarations

Competing Interests The authors declare no competing interests.

References

Alur R, Dill DL (1994) A theory of timed automata. Theoret Comput Sci 126(2):183–235
Ben-Ari M (2012) Mathematical Logic for Computer Science, 3rd edn. Springer
Biere A, Kröning D (2018) SAT-based model checking. In: Clarke EM, Henzinger TA, Veith H, Bloem R (eds) Handbook of Model Checking, Springer 10:277–303
Bjørner N, Phan AD (2014) νZ-maximal satisfaction with Z3. In: 6th International Symposium on Symbolic Computation in Software Science (SCSS 2014), La Marsa, Tunisia, EPiC Series in Computing 30:1–9
Bouyer P, Fahrenberg U, Larsen KG, Markey N, Ouaknine J, Worrell J (2018) Model checking real-time systems. In: Clarke EM, Henzinger TA, Veith H, Bloem R (eds) Handbook of Model Checking, Springer, chap 29, pp 1001–1046
Bryant RE (1992) Symbolic Boolean manipulation with ordered binary-decision diagrams. ACM Comput Surv 24(3):293–318
de Castro LN, Zuben FJV (2002) Learning and optimization using the clonal selection principle. IEEE Trans Evol Comput 6(3):239–251
Chen X (2020) How the CP-SAT solver works. http://xiang.es/posts/explaining-cp-sat
Chretienne P (1986) Timed Petri nets: A solution to the minimum-time-reachability problem between two states of a timed-event graph. J Syst Softw 6(1–2):95–101
Chuffed (2024) Chuffed, a lazy clause generation solver. http://github.com/ chuffed/chuffed
CP-SAT (2024a) CP-SAT Solver - Developers Guide. https://developers.google.com/optimization/cp/cp_solver
CP-SAT (2024b) CP-SAT Solver - Function List. http://or-tools.github.io/docs/ pdoc/ortools/sat/python/cp_model
Crawford JM, Baker AB (1994) Experimental results on the application of satisfiability algorithms to scheduling problems. In: Proc 12th National Conference on Artificial Intelligence (AAAI 1994), pp 1092–1097

David R, Alla H (2010) Discrete, Continuous, and Hybrid Petri Nets, 2nd edn. Springer
Davis M, Logemann G, Loveland D (1962) A machine program for theorem proving. Commun ACM 5(7):394–397
Eén N, Sörensson N (2004) An extensible SAT-solver. In: Giunchiglia E, Tacchella A (eds) Lecture Notes in Computer Science, Springer 2919:502–518
Eén N, Sörensson N (2006) Translating pseudo-boolean constraints into SAT. J Satisfiabil Boolean Model Comput 2(1–4):1–26
Feydy T, Stuckey PJ (2009) Lazy clause generation reengineered. In: Gent I (ed) Principles and Practice of Constraint Programming (CP 2009), Springer, Lecture Notes in Computer Science 5732:352–366
Gurobi (2015) Gurobi optimizer reference manual. http://www.gurobi.com
Hagebring F, Lennartson B (2019) Time-optimal control of large-scale systems of systems using compositional optimization. Discrete Event Dynamic Systems 29:411-443
Hill R, Lafortune S (2016) Planning under abstraction within a supervisory control context. IEEE 55th Conference on Decision and Control (CDC), pp 4770-4777
Hooker JN (2012) Integrated Methods for Optimization, International Series in Operations Research & Management Science, vol 170, 2nd edn. Springer
Lennartson B (2022) Reductions and abstractions for optimization of modular timed automata. In: 16th International Workshop on Discrete Event Systems (WODES'22), IFAC-PapersOnLine 55(28), pp 344–349
Lennartson B (2024) Optimization of timed Petri nets using CP-SAT. In: 17th International Workshop on Discrete Event Systems (WODES'24), IFAC-PapersOnLine 58(1):pp 90–95
Lennartson B, Basile F, Miremadi S, Fei Z, Noori-Hosseini M, Fabian M, Åkesson K (2014) Supervisory control for state-vector transition models - A unified approach. IEEE Trans Autom Sci Eng 11(1):33-47
Malik R, Pena PN (2018) Optimal task scheduling in a flexible manufacturing system using model checking. In: 14th International Workshop on Discrete Event Systems (WODES'18), pp 241–246
Marino ED, Su R, Basile F (2020) Makespan optimization using timed Petri nets and mixed integer linear programming problem. In: 15th International Workshop on Discrete Event Systems (WODES'20), IFAC-PapersOnLine 53(4), pp 129–135
Marques-Silva J (2012) MaxSAT and related optimization problems. In: SAT/SMT Summer School 2012, Fondazione Bruno Kessler, Trento. http://es-static.fbk.eu/events/satsmtschool12/slides/1x04_SS12.pdf
Marques-Silva J, Malik S (2018) Propositional SAT solving. In: Clarke EM, Henzinger TA, Veith H, Bloem R (eds) Handbook of Model Checking, Springer 9:247–275
Marques-Silva J, Lynce I, Malik S (2009) CDCL solvers. In: Biere A, Heule M, van Maaren H, Walsh T (eds) Handbook of Satisfiability, IOS Press 4:131–153
Merlin P, Farber D (1976) Recoverability of communication protocols - Implications of a theoretical study. IEEE Trans Commun 24(9):1036–1043
Moskewicz MW, Madigan CF, Zhao Y, Zhang L, Malik S (2001) Chaff: Engineering an efficient SAT solver. In: Proc 38th annual design automation conference, pp 530–535
Ohrimenko O, Stuckey PJ, Codish M (2009) Propagation via lazy clause generation. Constraints 14:357–391
Pena PN, Vilela JN, Alves MRC, Rafael GC (2022) Abstraction of the supervisory control solution to deal with planning problems in manufacturing systems. IEEE Trans Autom Control 67(1):344–350
Perron L (2023) The CP-SAT solver. In: Seminar, Czech Technical University, Prague, https://www.youtube.com/live/vvUxusrUcpU
Petke J, Jeavons P (2011) The order encoding: From tractable CSP to tractable SAT. In: Sakallah K, Simon L (eds) Theory and Applications of Satisfiability Testing (SAT 2011), Springer, Lecture Notes in Computer Science 6695:371–372
Pinedo ML (2022) Theory, algorithms, and systems Scheduling, 6th edn. Springer
de Queiroz MH, Cury JER, Wonham WM (2005) Multitasking supervisory control of discrete-event systems. Discrete Event Dynamic Systems 15(4):375–395
Ramchandani C (1973) Analysis of asynchronous concurrent systems by timed Petri nets. PhD thesis, MIT
Rintanen J (2009) Planning and SAT. In: Biere A, Heule M, van Maaren H, Walsh T (eds) Handbook of Satisfiability, IOS Press 15:483–504
Roselli SF, Bengtsson K, Åkesson K (2018) SMT solvers for job-shop scheduling problems: Models comparison and performance evaluation. In: IEEE 14th International Conference on Automation Science and Engineering (CASE 2018). Munich, Germany, pp 547–552
Silva JR, del Foyo PM (2012) Timed Petri nets. Petri Nets - Manufacturing and Computer Science IntechOpen chap 16:359–378
Stuckey P (2010) Lazy clause generation: Combining the power of SAT and CP (and MIP?) solving. In: Lodi A, Milano M, Toth P (eds) Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR 2010), Springer, Lecture Notes in Computer Science 6140:5–9

Stuckey PJ (2013) Search is dead, long live proof. https://people.eng.unimelb.edu.au/pstuckey/PPDP2013.pdf
Su R, van Schuppen JH, Rooda JE (2012) The synthesis of time optimal supervisors by using heaps-of-pieces. IEEE Trans Autom Control 57(1):105–118
Vilela J, Hill R (2022) Hierarchical planning in a supervisory control context with compositional abstraction. Discrete Event Dynamic Systems 32:89–113
Ware S, Su R (2017) Time optimal synthesis based upon sequential abstraction and its application to cluster tools. IEEE Trans Autom Sci Eng 14(2):772–784
Zaitsev DA, Sleptsov AI (1997) State equations and equivalent transformations for timed Petri nets. Cybern Syst Anal 33(5):659–672

Bengt Lennartson received the Ph.D. degree from the Chalmers University of Technology, Gothenburg, Sweden, in 1986. Since 1999 he has been a Professor of the Chair of Automation at the Department of Electrical Engineering, Chalmers University of Technology, where he was the Dean of Education 2004-2007. He was the General Chair of the 11th IEEE Conference on Automation Science and Engineering, CASE 2015, and the 9th International Workshop on Discrete Event Systems, WODES'08, and he has been Associate Editor of Automatica and IEEE Transactions on Automation Science and Engineering. He is the (co)author of more than 300 peer reviewed international papers, and his research is currently focused on discrete-event systems, AI planning and learning, as well as sustainable production and robust feedback control. He is a Fellow of the IEEE for his contributions to hybrid and discrete event systems for automation and sustainable production.