Optimization of Timed Petri Nets using CP-SAT

Available online at www.sciencedirect.com

ScienceDirect

IFAC PapersOnLine 58-1 (2024) 90–95

Optimization of Timed Petri Nets using CP-SAT [★]

Bengt Lennartson [*]

[*] *Division of Systems and Control, Chalmers University of Technology, SE-412 96 Göteborg, Sweden (e-mail: bengt.lennartson@chalmers.se).*

Abstract: An optimization formulation is presented for timed Petri nets, based on a recently developed optimization solver where a satisfiability solver is integrated with constraint programming. The solver, called CP-SAT, is a part of Google's OR-Tools. The first optimization formulation includes an arbitrary number of concurrent sequences of operations, with shared, alternative, and local resources. A benchmark shows how much faster CP-SAT is compared to both an alternative SAT optimization solver and an A* implementation. The optimization formulation is then generalized to mixed alternative and concurrent sequences. A comparison with a recent MILP formulation for timed Petri nets is presented, showing the strength of the proposed optimization formulation. Finally, an evaluation of an industrial-sized flexible manufacturing system, including uncontrollable events, demonstrates how efficient and easy to implement the proposed strategy is compared to existing results.

Keywords: Time-optimal control, timed Petri net, satisfiability solver, constraint programming.

1. INTRODUCTION

Time-optimal control and synthesis of time-optimal supervisors are often formulated based on time weighted automata, including abstractions to reduce the computational complexity Lennartson (2022). For Petri nets (PNs), such abstractions can only be applied to safe nets, including maximum one token in each place. The reason is that multiple tokens in a place, requiring the same resource, results in a shared resource between the tokens, and therefore a non-local behavior. Abstractions involving shared resources in a timed framework gives very small reductions, and the solution is instead to search for effective global optimization strategies.

Optimization strategies for systems including discrete states have been developed both in operation research (OR) and artificial intelligence, especially within the constraint programming (CP) community. In Hooker (2012) it is observed that by combining search, inference, and relaxation strategies from both OR and CP, significant algorithmic improvements can be achieved. Satisfiability solvers based on SAT and 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 is unsatisfiable. The lowest value that generates a satisfiable solution then generates the minimal time-optimal solution. This strategy has been implemented in the SMT solver Z3 and called Z3Opt (Bjørner and Phan, 2014). Evaluations for typical scheduling problems in Roselli et al. (2018) show that Z3Opt typically computes an optimal solution 10 times faster than a MILP solver in Gurobi (Gurobi, 2015). A second SAT-based solver has followed Hooker's recommendation and combined the satisfiability strategy with specific CP oriented functions to more efficiently handle both shared and alternative resources in concurrent sequences. The solver is called CP-SAT and is a part of Google's OR-Tools (CP-SAT, 2024a,b).

[★] This work was supported by SyTec – Systematic Testing of Cyber-Physical Systems, a Swedish Science Foundation grant for strong research environment and Wallenberg Artificial Intelligence, Autonomous Systems and Software Program (WASP) funded by Knut and Alice Wallenberg Foundation. The support is gratefully acknowledged.

Compared with Z3Opt, but also an A* implementation, the improvement using CP-SAT is in this paper shown to be significant. For the largest evaluated problem, CP-SAT runs about 50 times faster than Z3Opt, due to a number of important built-in CP functions. Inspired by this strength, we show how makespan optimization of timed PNs can be formulated and solved in CP-SAT. It is also demonstrated how some specific PN related constraints are critical when optimization of timed PNs is performed and multiple tokens are involved. 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 symbolically and implicitly determined by constraints on the local start and end times of the involved tokens 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, compared to 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.

2. 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 where 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 Petri nets is therefore presented in this section.

Definition of Timed Petri Net A PN with explicit timing information, called a Timed Petri Net (TPN), will now be introduced (David and Alla, 2010). It is defined as a six-tuple

$$\mathcal{N} = \langle P, T, Pre, Post, M_0, f \rangle$$

where P is a set of *places* and T is a set of *transitions*. For arbitrary places $p \in P$ and transitions $t \in T$, Pre and $Post$ are matrices, defining the graph 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 a corresponding arc from transition t to place p. A *marking* is a vector M that assigns to each place $p \in P$ a

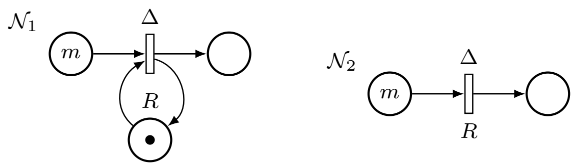

Figure 1. Two equivalent TPNs with m tokens in the initial place and a shared resource R with capacity one. Each token is assumed the have the same transition duration Δ.

non-negative integer number of tokens, and M_0 is the initial marking. Furthermore, $f : T \to \mathbb{R}_{\geq 0}$ is a function that assigns a non-negative duration $\Delta = f(t)$ to each transition $t \in T$.

Shared resource and multiple tokens The TPN $\mathcal{N}_1$ in Fig. 1 includes m tokens in the initial place, waiting to be operated by a single shared resource R. The timed transition has a non-zero transition duration Δ, graphically shown as a non-filled bar with transition label Δ. The duration Δ means that the time resource R requires to perform its operation on each individual token is equal Δ. The least total time it takes to handle m tokens is therefore $m\Delta$. A simplified equivalent graphical description, which models the same behavior as $\mathcal{N}_1$, is shown in the TPN $\mathcal{N}_2$.

Shared resource with individual durations The duration is equal for all tokens in the TPNs $\mathcal{N}_1$ and $\mathcal{N}_2$. Individual durations, one unique for each token, are introduced in the TPNs $\mathcal{N}_3$ and $\mathcal{N}_4$ in Fig. 2, where the more compact notation in $\mathcal{N}_4$ is used in the flexible manufacturing system example in Section 4. The shared resource is expressed in $\mathcal{N}_4$ by introducing the same additional transition label R for the shared resource in all m sequences. These sequences are extended in the next section with a number of sequential operations, both with safe TPNs and TPNs including multiple tokens.

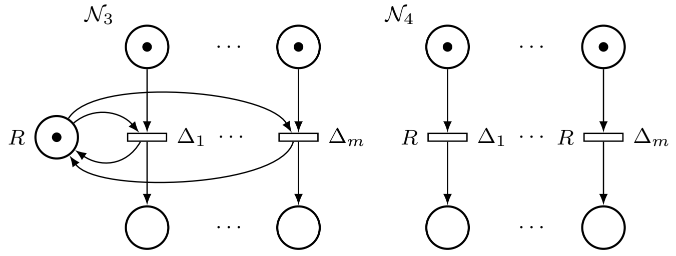

Figure 2. Two equivalent safe TPNs (maximum one token in every place). Each model includes m concurrent operations, with individual durations $\Delta_1, \ldots, \Delta_m$ and a shared resource R. Compared to the TPN $\mathcal{N}_3$, the equivalent TPN $\mathcal{N}_4$ is simplified, by replacing the resource place R with a corresponding transition label R in all m sequences.

3. TIME OPTIMIZATION

It will now be shown how time optimization of TPNs can be performed by formulating a Constraint Programming (CP) problem. This problem can be solved efficiently, especially by solvers which combine strategies from both operation research (OR), CP, and satisfiability (SAT) solvers. The integration between OR and CP is well known, and satisfiability solvers based on SAT and SMT have also been extended with optimization strategies; one example is Z3Opt (Bjørner and Phan, 2014). Another example is Google's OR-Tools introduction of a combined CP and SAT solver, called CP-SAT (CP-SAT, 2024a,b). We will in this section 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 resources, are modeled. Then follows a generalization to mixed alternative and concurrent sequences.

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. In this paper we will focus on makespan, noting that the proposed methods are easily reformulated to tardiness criteria.

Sets and variables For analysis and optimization of TPNs involving a number of concurrent sequences of operations with multiple tokens and shared as well as alternative resources, it is necessary to consider the timing for each individual token. The makespan optimization problem below is therefore based on the following sets and variables:

- $Seq = \{1, \ldots, |Seq|\}$ = index set for a set of sequences,
- $Tok(i) = \{1, \ldots, |Tok(i)|\}$ = index set for tokens in sequence $i \in Seq$,
- $Op(i) = \{1, \ldots, |Op(i)|\}$ = index set for operations in sequence $i \in Seq$,
- $R_\oplus(i,k) = \{1, \ldots, |R_\oplus(i,k)|\}$ = index set for alternative resources in sequence $i \in Seq$ and operation $k \in Op(i)$,
- $\mathcal{R}$ = set of all system resources,
- Sys = set of tuples $(R_{i,k,\ell}, \Delta_{i,k,\ell})$, where $R_{i,k,\ell} \in \mathcal{R}$ is one alternative resource with duration $\Delta_{i,k,\ell} \in \mathbb{N}$ in sequence $i \in Seq$ and operation $k \in Op(i)$ with alternative resource index $\ell \in R_\oplus(i,k)$,
- $s_{i,j,k}$ = starting time, $e_{i,j,k}$ = end time, and $d_{i,j,k}$ = duration for token $j \in Tok(i)$ and operation $k \in Op(i)$ in sequence $i \in Seq$,
- MS = end time when all concurrent operations have been executed = makespan.

To be precise, sequences, tokens, and operations are denoted by their index i, j, and k, while resources are denoted by their name in the resource set $\mathcal{R}$. One exception is the alternative resource index $\ell \in R_\oplus(i,k)$.

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

$$\min MS \quad \text{subject to}$$

$$\forall i \in Seq,\ j \in Tok(i),\ k \in Op(i),\ \ell \in R_\oplus(i,k):$$

$$(R_{i,k,\ell}, \Delta_{i,k,\ell}) \in Sys$$

$$b_\ell = \text{exactOne}(b_1, \ldots, b_{|R_\oplus(i,k)|}) \tag{1}$$

$$b_\ell \to (r_{i,j,k} = R_{i,k,\ell}) \wedge (d_{i,j,k} = \Delta_{i,k,\ell}) \tag{2}$$

$$e_{i,j,k} = s_{i,j,k} + d_{i,j,k}$$

$$s_{i,j,k+1} \geq e_{i,j,k}\,, \qquad k < |Op(i)| \tag{3}$$

$$s_{i,j+c,k} \geq e_{i,j,k}\,, \qquad c = \text{capacity}(r_{i,j,k}) \tag{4}$$

$$\text{cumulative}(i, j, k, r_{i,j,k}) \tag{5}$$

$$MS \geq e_{i,j,k}\,, \qquad k = |Op(i)|$$

Alternative resources The logical function exaktOne in (1), which is executed for all $\ell \in R_{\oplus}(i,k)$, 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 such that MS is minimized. For each operation k in the different sequences i, the chosen resource and related duration are assigned to the variables $r_{i,j,k}$ and $d_{i,j,k}$. Thus, (1) and (2) are the only information that is required to implement alternative resources. Both the function exactOne and the conditional assignments in (2) are implemented by built-in functions included in CP-SAT, where for instance the conditional assignment Add$(r_{i,j,k} == R_{i,k,\ell}, b_\ell)$ is only performed when b_ℓ = true. The alternative resources are removed by only implementing unconditional assignments of $r_{i,j,k}$ and $d_{i,j,k}$.

Mutual exclusion The function cumulative (5) implements mutual exclusion among the shared resources, with given capacity defined by the number of tokens in the resource places. In Fig. 2 the capacity is equal one. This function generates the constraint

$$\begin{aligned}&\text{cumulative}(i,j,k,r_{i,j,k}) = \\ &\quad \forall i' \in Seq, j' \in Tok(i'), k' \in Op(i') : r_{i,j,k} = r_{i',j',k'} \\ &\quad \wedge\Big(|\{(i',j',k') \,|\, [s_{i,j,k}, e_{i,j,k}] \cap [s_{i',j',k'}, e_{i',j',k'}] \neq \varnothing\}| \\ &\quad \leq \text{capacity}(r_{i,j,k})\Big) \qquad (6)\end{aligned}$$

where $[s_{i,j,k}, e_{i,j,k}]$ denotes the set of integers from the start time $s_{i,j,k}$ to the end time $e_{i,j,k}$. This constraint is implemented in an effective way in CP-SAT, where the timing data is delivered to this function by a dictionary with the resources as keys.

Sequential relations Constraint (3) specifies the sequential relation between the operations in all sequences, and (4) adds an additional ordering between the tokens. In Examples 2 and 3, this constraint will be shown to have a great impact on execution performance.

3.2 Evaluation of OptConcurSeq

In the following two examples the optimization formulation OptConcurSeq will be evaluated. In the first example the efficiency of the proposed CP-SAT solver is illustrated by a benchmark test where this solver is compared with some other powerful strategies.

Example 1. Three concurrent sequences of operations will now be evaluated, where each sequence only includes one token, and each resource has capacity one. Let n be the number of operations in each sequence where n is an even number, and let each second operation be performed by a resource that is shared among all three sequences. A TPN for such a sequence is shown in Fig. 3, where in this example the number of tokens in the initial place $m = 1$. The duration for each resource is a random integer number with equal distribution in the interval [7, 13], and each shared resource R_k has the same duration Δ_k, $k = 1, 3, \ldots, n-1$ in the three sequences.

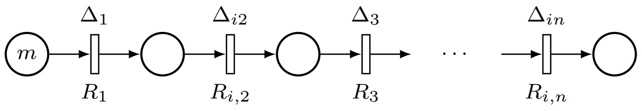

Figure 3. One sequence i in a TPN with an even number of operations equal n in each sequence. The initial place in each sequence has m tokens, and each second operation has a resource $R_1, R_3, \ldots, R_{n-1}$, respectively. These resources are shared among all sequences, while the rest of the operations have local resources $R_{i,2}, R_{i,4}, \ldots, R_{i,n}$ for each individual sequence i.

Table 1. Optimal makespan MS, makespan MS_{A*} for A*, and execution times (sec) using CP-SAT, CP-SAT-disjunctive, Z3Opt, and A* for different number of operations n in Example 1.

n	MS/MS_{A*}	CP-SAT	CP-SAT-disjunctive	Z3Opt	A*
100	1061/1079	0.06	0.14	0.40	1.90
200	2031/2065	0.12	0.32	4.15	11.2
300	3053/3076	0.20	0.53	10.9	78.8

Since all resources have capacity one, the cumulative constraint in (6) is simplified to a noOverlap condition, which for all $i, i' \in \{1,2,3\}$ and $k, k' \in \{1,\ldots,n\}$ can be expressed by the disjunctive condition

$$(s_{i,k} \geq e_{i',k'} \vee s_{i',k'} \geq e_{i,k}) \wedge i < i' \wedge r_{i,k} = r_{i',k'} \qquad (7)$$

where the third token index j is excluded, since only one token is involved in all three sequences, i.e. $j = 1$. In Table 1 the makespan MS and execution times for three different implementations are compared, 1) CP-SAT with its built-in function noOverlap, 2) CP-SAT with the disjunctive constraint (7) called CP-SAT-disjunctive, 3) the alternative SAT/SMT based solver Z3Opt (Bjørner and Phan, 2014) with the disjunctive constraint (7), and 4) an A* implementation that is built on synchronized timed automata models for the individual sequences (Lennartson, 2022), and a heuristic suggested by Abdeddaim and Maler (2001).

The A* heuristic is very efficient in the sense that few states are evaluated, but slightly suboptimal for larger systems. Despite the few evaluated states, the A* algorithm is much less efficient than the SAT based solvers. For the largest system, where each of the three concurrent sequences includes 300 operations, CP-SAT with the built-in noOverlap CP routine is 2.5 times faster than the disjunctive formulation (7) and 50 times faster than Z3Opt, while the A* implementation, with an efficient but slightly suboptimal heuristics is 400 times slower than the best CP-SAT implementation. It should also be mentioned that for similar optimization problems, it has recently been shown that, compared to classical MILP solvers, Z3Opt is typically 10 times faster than Gurobi's MILP implementation (Roselli et al., 2018). Furthermore, a related TPN optimization problem was also shown to be solved about 10 times faster using a CP solver, compared to a MILP solver in Lennartson et al. (2014).

Altogether, the results in Table 1 and earlier experiences show that CP-SAT is very efficient, even compared to the SAT based solver Z3Opt. One reason is that the built-in function noOverlap in CP-SAT includes CP based improvements. Compared to pure CP and MILP based solvers, it is the combination of CP and SAT, but also basic features from MILP, which makes CP-SAT extremely efficient for this type of problems. □

In the following examples only the CP-SAT solver is used, but it is not only the choice of solver that is important, also the problem formulation may be critical. In the next example, the benefit of including constraint (4) is demonstrated.

Example 2. A number of tokens are now introduced ($m > 1$ in Fig. 3), while the number of sequences is reduced to two. The resource capacity is still equal one, but is now also active for the resources only involved in one of the sequences, due to the multiple tokens, cf. Fig. 1. The individual starting times for each token means that each token has an individual identity, implemented by the index $j \in nTok$. Thus, the optimization formulation OptConcurSeq, without constraint (4), can be in-

Table 2. Optimal makespan MS and execution times (sec) using CP-SAT, including and not including constraint (4) for different number of tokens m in Example 2. Time out (T.O.) is 200 sec.

m	MS	Incl. (4)	Not incl. (4)
3	107	0.01	0.01
5	131	0.04	1.2
6	143	0.21	57.9
7	155	0.44	T.O.
10	191	6.41	T.O.
15	251	81.6	T.O.

terpreted as a colored TPN where the identity (color) of each token is traced. Adding (4) as constraint reduces the freedom and complexity of the solution by introducing an order between the tokens. With capacity $c = 1$, constraint (4) becomes $s_{i,j+1,k} \geq e_{i,j,k}$, specifying that token $j + 1$ is not allowed to start before the end time of token j. For pure non-colored TPNs, this restriction does not matter, since it is then only the number of tokens that counts, not in which order the specific tokens are starting each operation.

The results in Table 2 show that keeping the order between the individual tokens by constraint (4), which does not change the makespan, 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. □

3.3 Mixed alternative and concurrent sequences

So far, TPNs including a set of concurrent sequences have been considered where shared and alternative resources may be 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. (1), and assigning $d_{i,j,k} = 0$ 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.

Optimization formulation OptConcurAltSeq

Same as OptConcurSeq with (2) replaced by the constraints

$$i \in Seq_{\oplus} \to \beta_{i,j} = \text{exactOne}([\beta_{\iota,j}]_{\iota \in Seq_{\oplus}}) \quad (8)$$

$$i \in Seq_{\oplus} \wedge \beta_{i,j} \wedge b_{\ell} \to (r_{i,j,k} = R_{i,k,\ell}) \wedge (d_{i,j,k} = \Delta_{i,k,\ell}) \quad (9)$$

$$i \in Seq_{\oplus} \wedge \neg\, \beta_{i,j} \to d_{i,j,k} = 0 \quad (10)$$

$$i \notin Seq_{\oplus} \wedge b_{\ell} \to (r_{i,j,k} = R_{i,k,\ell}) \wedge (d_{i,j,k} = \Delta_{i,k,\ell}) \quad (11)$$

In (8), $[\beta_{\iota,j}]_{\iota \in Seq_{\oplus}}$ is a list of booleans where each $\beta_{\iota,j}$ corresponds to one of the alternative sequences in $Seq_{\oplus}$, i.e. different sequences may be chosen for each individual token j. Among the alternative sequences, only one sequence $i \in Seq_{\oplus}$ will be chosen, implying that $\beta_{i,j} = true$, and the corresponding operation durations and resources are assigned in (9). The rest of the booleans $\beta_{\iota,j} = false, \iota \in Seq_{\oplus} \setminus \{i\}$, due to the exactOne function in (8), and the corresponding durations are all assigned to zero in (10). The remaining sequences $Seq \setminus Seq_{\oplus}$ are assigned in (11) in the same way as (2) in OptConcurSeq. This optimization formulation is easily generalized to multiple sets of alternative sequences.

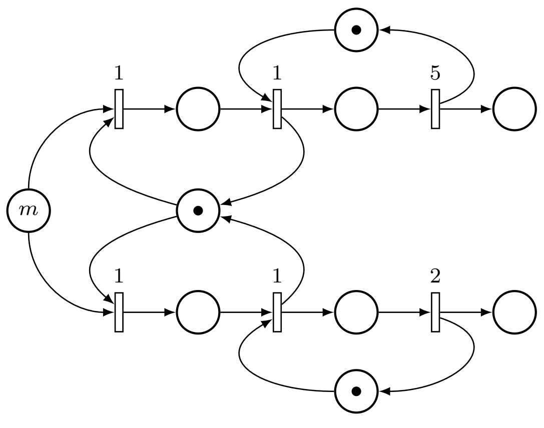

Figure 4. A TPN with two alternative sequences and resource allocations, each one involving two operation durations.

Example 3. Consider the two alternative sequences in Fig. 4 where the resource capacity tokens return after one additional timed transition. This means that the end time in the interval $[s_{i,j,k}, e_{i,j,k}]$ in the cumulative constraint function (6) needs to be extended to $e_{i,j,k+1}$, and a corresponding extension in the primed intervals for all other extended time intervals. In (4) $e_{i,j,k}$ is also replaced by $e_{i,j,k+1}$. Resulting makespan MS and execution times are shown in Table 3 with and without constraint (4), 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 in one of the evaluations the durations are $\Delta_{1,k} = \Delta_{2,k} = k$ (the alternative resource index is here excluded), which for an arbitrary number of tokens m gives $MS = 3(m + 1)$. In Marino et al. (2020), a MILP problem is formulated including explicit marking states. The execution times in Example 1 together with the evaluation in Roselli et al. (2018) indicate a potential improvement up to 500 times going from MILP solvers to CP-SAT for TPNs. This drastic computational improvement is confirmed by comparing the execution times reported in Marino et al. (2020) with the results achieved by the proposed OptConcurAltSeq formulation based on CP-SAT. □

Table 3. Optimal makespan MS and execution times (sec) using CP-SAT including and not including constraint (4) for different number of tokens m in Example 3. Time out (T.O.) is 200 sec.

m	MS	Incl. (4)	Not incl. (4)
2	7	0.01	0.01
5	14	0.02	0.02
10	25	0.05	88.3
15	37	0.74	T.O.
20	49	13.5	T.O.
25	60	135	T.O.

3.4 Start and completion of local sequences

Alternative and concurrent sequences do not always start from the beginning and terminate before all sequences have been 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

$$\bigvee_{\iota \in Seq_\oplus} s_{\iota,j,1} \geq e_{i_1,j,n_{i_1}} \quad \text{and} \quad \bigvee_{\iota \in Seq_\oplus} s_{i_2,j,1} \geq e_{\iota,j,n_\iota} \tag{12}$$

to OptConcurAltSeq. Note that the choice of the optimal alternative sequence in $Seq_\oplus$ is obtained by (8)-(10).

Local concurrent sequences Replace the set $Seq_\oplus$ in (12) with a set of local concurrent sequences $Seq_{||} \subseteq Seq$. Local concurrent sequences are then obtained by replacing the disjunction operator in (12) with a conjunction over all sequences in $S_{||}$. Both local alternative and concurrent sequences can also be generalized to corresponding multiple sets of local sequences.

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

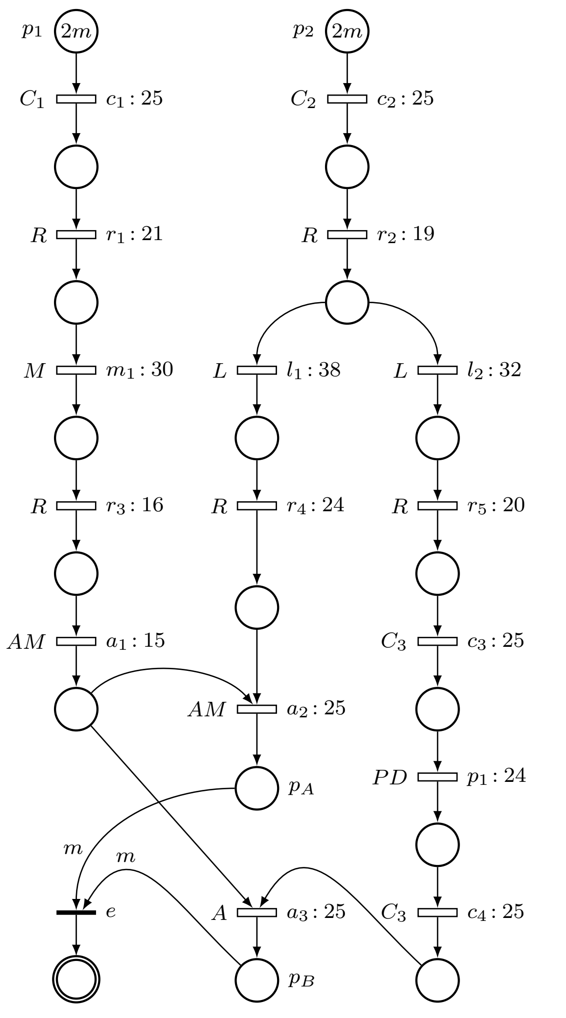

Figure 5. A TPN for a flexible production system where m products of type p_A and p_B are produced based on $2m$ input parts of type p_1 and p_2.

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 concurrent as well as 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 with 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. 5 where each timed transition has a resource label with capital letter and a duration, but also a unique small letter transition label used as operation name. 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.

Concurrent and alternative sequences To handle the concurrency and the alternative sequences in Fig. 5, the system model Sys is divided into four sequences, $i = 1$ including the operations c_1, r_1, m_1, r_3, and a_1, $i = 2$ including c_2 and r_2, $i = 3$ including l_1, r_4, and a_2, and $i = 4$ including l_2, r_5, c_3, p_1, c_4, and a_3. 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 operation a_2 for $i = 3$ and a_3 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 are $2m$ and m products of both types are produced. Thus, the input places p_1 and p_2 in Fig. 5 have $2m$ initial tokens, and the goal is to generate m tokens in both place p_A and place p_B. When this goal is achieved, the last non-timed transition with label e can be fired, resulting in only one token in the double circle place. In a similar way as for automata, this demonstrates the possibility to graphically specify 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 has been reached.

Weighted transitions in CP-SAT Weighted transitions are normally added to specify logical conditions, such as the goal specification on equal number of tokens in the places p_A and p_B. In CP-SAT, this specification is simply formulated as the constraint, cf. (8), (9)

$$\sum_{j=1}^{2m} \beta_{3,j} = m. \tag{13}$$

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

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. This means that the constraint on the end time has to be moved to the more conservative constraint on the corresponding start time, which results in the following uncontrollability constraint

$$s_{i,j+1,k} \geq s_{i,j,k+1} \tag{14}$$

Optimal solution comparison Adding the constraints (12)-(14) to OptConcurAltSeq, together with the resource and duration information in Fig. 5, implies that the optimization formulation can be solved by CP-SAT. The resulting optimal

Table 4. Optimal makespan MS for the TPN in Fig. 5 and execution times (sec) using CP-SAT (T.O. = 200 sec.) and execution times (min) using CSA-DS for different number of tokens m.

m	MS	CP-SAT Optimal	CP-SAT Feasible	CSA-DS
3	518	0.07	0.06	1.7 min
5	830	0.31	0.21	3.5 min
7	1142	2.70	0.28	5.8 min
10	1610	147	0.64	9.5 min
13	2078	T.O.	1.05	7.4 min
15	2390	T.O.	1.92	8.5 min
20	3170	T.O.	9.94	14.4 min

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. Ones again the CP-SAT solver shows its strength, being able to generate the optimal solution in a much shorter time than in Pena et al. (2022) when the feasible but not proven optimal solution, as motivated below, is also taken into account.

Feasible but not proven optimal solution In Table 4, the proven optimal solution is possible to compute for $m \leq 10$. For higher m values the computation time suddenly 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 Z3Opt. Using CP-SAT, we find for $2 \leq m \leq 10$ a simple pattern on the optimal makespan where

$$MS = 156m + 50 \tag{15}$$

Note the importance of obtaining the exact optimal solution to get this optimal pattern, which is not the case for the CSA-DS strategy. Fortunately, CP-SAT can also be asked to run for a limited time, delivering a feasible solution, which fulfills all constraints but does not necessarily generate an optimal solution. Since the most costly part of the computation is the proof of optimality, a feasible solution where MS has converged to a constant value, can be achieved much more quickly than to prove the optimality of the delivered solution. As expected, MS also converges in this example to a constant value which for each m fulfills the optimal formula (15). Thus, we can trust this as the optimal MS value, also supported by the structure of the FMS system, which is shown to converge to a cyclic behavior.

In the column CP-SAT Feasible in Table 4 the computation time is shown when the feasible solution has reached the expected optimal value (15). Unfortunately, this optimal MS value is not what we are searching for. We are asking for the optimal sequence of operations which will generate the optimal makespan, i.e. the optimal control law. However, since every feasible solution generates an executable sequence of operations, and we have a feasible solution that satisfies the optimal MS value, the obtained sequence is also the optimal sequence of operations and therefore the optimal control law.

5. 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 results in an easy and general optimization formulation, and by including specific constraints valid only for 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 studies is how this formulation can be generalized to arbitrary uncontrollable event scenarios.

ACKNOWLEDGEMENTS

Many thanks to Dr. Sabino Roselli who kindly shared his knowledge in using Z3Opt.

REFERENCES

Abdeddaim, Y. and Maler, O. (2001). Job shop scheduling using timed automata. In G. Berry, H. Comon, and A. Finkel (eds.), *13th Int. Conf. Computer Aided Verification (CAV)*, volume 2102 of *Lecture Notes in Computer Science*, 478–492. Springer, Paris, France.

Bjørner, N. and Phan, A.D. (2014). νz-maximal satisfaction with z3. In *6th International Symposium on Symbolic Computation in Software Science (SCSS 2014)*, EPiC Series in Computing 30, 1–9. La Marsa, Tunisia.

CP-SAT (2024a). *CP-SAT Solver - Developers Guide*. URL http://developers.google.com/optimization/cp/cp_solver.

CP-SAT (2024b). *CP-SAT Solver - Function List*. URL http://or-tools.github.io/docs/pdoc/ortools/sat/python/cp_model.

David, R. and Alla, H. (2010). *Discrete, Continuous, and Hybrid Petri Nets*. Springer.

de Castro, L.N. and Zuben, F.J.V. (2002). Learning and optimization using the clonal selection principle. *IEEE Transactions on Evolutionary Computation*, 6(3), 239–251.

de Queiroz, M.H., Cury, J.E.R., and Wonham, W.M. (2005). Multitasking supervisory control of discrete-event systems. *Discrete Event Dynamic Systems*, 15(4), 375–395.

Eén, N. and Sörensson, N. (2006). Translating pseudo-boolean constraints into SAT. *Journal on Satisfiability, Boolean Modeling and Computation*, 2(1-4), 1–26.

Gurobi (2015). *Gurobi optimizer reference manual*. URL http://www.gurobi.com.

Hooker, J.N. (2012). *Integrated Methods for Optimization*, volume 170 of *International Series in Operations Research & Management Science*. Springer, second edition.

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), 344–349.

Lennartson, B., Wigström, O., Fabian, M., and Basile, F. (2014). Unified model for synthesis and optimization of discrete event and hybrid systems. In *12th International Workshop on Discrete Event Systems, WODES'14*, IFAC Proceedings, 47 (2), 86–92. Cachan, France.

Marino, E.D., Su, R., and 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), 129–135.

Pena, P.N., Vilela, J.N., Alves, M.R.C., and Rafael, G.C. (2022). Abstraction of the supervisory control solution to deal with planning problems in manufacturing systems. *IEEE Transactions on Automatic Control*, 67(1), 344–350.

Roselli, S.F., Bengtsson, K., and 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)*, 547–552. Munich, Germany.