SCHEDULING OF MULTI-PRODUCT BATCH PLANTS USING REACHABILITY ANALYSIS OF TIMED AUTOMATA MODELS

Subanatarajan Subbiah, Sebastian Panek, Sebastian Engell
Process Control Laboratory (BCI-AST), University of Dortmund, 44221 Dortmund, Germany

Olaf Stursberg
Industrial Automation Systems (EI-LSR), Technical University of Muenchen, Muenchen, Germany

Keywords: Scheduling, timed automata, reachability analysis, multi-product batch plant.

Abstract: Standard scheduling approaches in process industries are often based on algebraic problem formulations solved as MI(N)LP optimization problems to derive production schedules. To handle such problems techniques based on timed automata have emerged recently. This contribution reports on a successful application of a new modeling scheme to formulate scheduling problems in process industries as timed automata (TA) models and describes the solution technique to obtain schedules using symbolic reachability analysis. First, the jobs, resources and additional constraints are modeled as sets of synchronized timed automata. Then, the individual automata are composed by parallel composition to form a global automaton which has an initial location where no jobs have been started and at least one target location where all jobs have been finished. A cost optimal symbolic reachability analysis is performed on the composed automaton to derive schedules. The main advantage of this approach over other MILP techniques is the intuitive graphical and modular modeling and the ability to compute better solutions within reasonable computation time. This is illustrated by a case study.

1 INTRODUCTION

Scheduling has applications in many fields and one among them are the chemical industries employed with multi-product batch plants. Multi-product batch plants in contrast to continuous plants offer the advantage of an increased flexibility with respect to the variation of the products, the volume of production, and the range of recipes that can be processed by the equipment. This flexibility makes the scheduling task challenging. The scheduler has to take a large number of decisions to answer questions of what, when, where and how to produce the products in order to satisfy the market demand. The scheduling task is particularly challenging due to numerous constraints arising from process topology, the interlinks between pieces of equipment, inventory policies, material transfers, batch sizes, batch processing times, demand patterns, changeover procedures, resource and time constraints, cost functions and degrees of uncertainty. Many scientists in the process logistics community have contributed approaches to model and solve such problems (Kallrath, 2002). In most prominent approaches the problem is specified using the frameworks of State Task Networks (STN) or Resource Task Networks (RTN) (Pantelides, 1994), further transformed to an optimization problem modeled as a mixed integer program (MILP or MINLP) and solved using state-of-the art solvers such as CPLEX, XPRESS-MP, etc. The modeling of mathematical optimization problems is generally cumbersome, tedious and requires not only experience in algebraic modeling but also a deep knowledge of the solver and its internal algorithms.

Recently, techniques based on reachability analysis for timed automata to solve scheduling problems have gained attention. The framework of TA has been originally proposed by (Alur and Dill, 1994) to model timed systems with discrete dynamics. Many powerful tools for the modeling and analysis of real-time systems such as UPPAAL, IF, KRONOS and TAOPT have emerged. The authors of (Abdeddaim and Maler, 2001) have proposed to use TA to solve job-shop scheduling problems based on the symbolic reacha-

Subbiah S., Panek S., Engell S. and Stursberg O. (2007).
SCHEDULING OF MULTI-PRODUCT BATCH PLANTS USING REACHABILITY ANALYSIS OF TIMED AUTOMATA MODELS.
In *Proceedings of the Fourth International Conference on Informatics in Control, Automation and Robotics*, pages 141-148
DOI: 10.5220/0001627501410148

bility analysis of TA. Promising results on challenging benchmark instances in job-shops were reported in (Panek et al., 2006). The particular appeal of this approach comes from the modular and partly graphical modeling which enables inexperienced users to build models. Another advantage is the availability of powerful search algorithms that can easily be modified and extended for special purposes. Similar to other approaches the reachability analysis of TA also suffers from the drastic increase in the computational cost with an increase in the problem size. The problem of state-space explosion in TA models can be reduced to some extent by state-space reduction techniques (Panek et al., 2007), but arises again when the problem instance is scaled. An advantage of the TA-based approach is that good, though not provable optimal solutions are often found relatively quickly.

The structure of the rest of this paper is as follows: In Section 2, Resource Task Networks (RTN) are introduced as the basic model for specifying batch scheduling problems. Section 3 explains how the RTN model can be translated into TA, and how the corresponding scheduling problem can be solved by symbolic reachability analysis. In section 4, different greedy strategies to reduce the solution space are explained. The description of the case study and the results of the experiments for the example considered are provided in section 5. The paper closes with conclusions and an outlook.

2 MODELING WITH RTN

In the context of mathematical programming, STN and RTN serve as starting points for formulating integrated mathematical models that can be solved to obtain production schedules. A short description of the RTN representation is given in this section based upon a small example: a two stage process with 3 tasks and 3 resources. The resources R_1, R_2 and R_3 handle batch sizes of b_1, b_2 and b_3 units. In the first stage, the single raw material S_1 is processed by task T_1 in resource R_1 to produce the intermediate material S_2. In the second stage, the intermediate material S_2 is processed by tasks T_2 in R_2 and T_3 in R_3 to produce the end products S_3 and S_4. The tasks T_1, T_2 and T_3 requires d_1, d_2 and d_3 time units, respectively. There exists a timing constraint between tasks T_1 and T_3 such that T_3 must be started within Y time units after T_1 has finished, as the intermediate product S_2 is unstable and becomes unusable after Y time units. We assume in the example that dedicated storage tanks are available for each of the products that have the same names as the products. The RTN model consists of four basic elements; (i) *state nodes*, to represent the feeds, intermediate materials and the final products, (ii) *task nodes* to represent the process operations that transforms one or more input states to one or more output states, (iii) *resource nodes* to represent the production units/equipments, and (iv) *arcs* with fractional values to indicate the flow and the percentage of material transported to the successor state. The default values for the flow are 1. The RTN description of the example process is shown in Figure 1. The feed, intermediate materials and products (S_1,$\cdots$ S_4) are represented as circles with diagonal lines. The tasks T_1, T_2 and T_3 are represented as rectangles and connected to the corresponding resources R_1, R_2 and R_3, respectively by dashed lines.

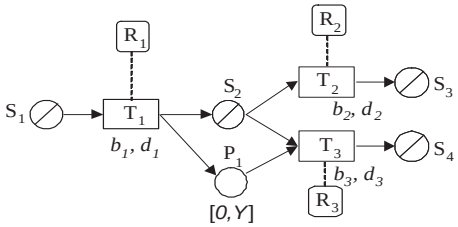

Figure 1: RTN description of the example process.

The standard RTN framework offers only limited possibilities to describe timing constraints between tasks. The timing constraint between task T_1 and T_3 is modeled by a new type of node called *place* denoted by P_1. The notion of places is motivated by *Timed and Hybrid Petri Nets* (Barthomieu and Diaz, 1991). Places can be connected to tasks in the same fashion as states, however unlike states they represent no physical products. They contain either one or no *token*. When one of the predecessor operations is finished, a token is assigned in the place and it is consumed when one of the successor operation starts. The place in the RTN is marked by the time interval [0, Y], signifying that a token can stay in the place for at least 0, and for at most Y time units.

2.1 Timed Automata

Timed automata (TA) are extensions of finite-state automata with clocks and are used to model and analyze systems with discrete dynamics and timed behaviour. Time in TA is modeled using a set of clocks. A short and informal definition of TA is given below, for a detailed definition for TA please refer to (Alur and Dill, 1994).

2.1.1 Syntax

A timed automata is defined by a tuple, TA = (L, l_o, F, $\mathbb{C}$, E, inv), in which L represents the finite set of

locations, l_0 represents the initial location and F represents the set of final locations. $\mathbb{C}$ represents the set of clocks assigned to the TA. The set $E \subset L \times \Phi(\mathbb{C}) \times Act \times \mathcal{P}(\mathbb{C}) \times L$ represents the transitions. $\Phi(\mathbb{C})$ are guards specified as conjunctions of constraints of the form $c_i \otimes n$ or c_i - $c_j \otimes n$, where c_i, $c_j \in \mathbb{C}$, $\otimes \in \{ <, \leq, ==, \neq, >, \geq \}$ and $n \in \mathbb{N}$. *Act* denotes a set of actions (e.g. invoking a new event or changing the value of a variable). $\mathcal{P}(\mathbb{C})$ represent the power set over elements of $\mathbb{C}$. *inv* represents a set of invariants which assign conditions for staying in locations. An invariant must evaluate to true when the corresponding location is active. The automaton is forced to leave the location when the invariant evaluates to false.

(l, g, a, r, l') denotes a transition between a source location l and target location l' with a guard $g \in \Phi(\mathbb{C})$, performing an action $a \in Act$ and resetting the clocks $r \in \mathcal{P}(\mathbb{C})$. Such a transition can occur only when the transition guard $g \in \Phi(\mathbb{C})$ is satisfied.

Figure 2 shows a simple TA with three locations l_1, l_2 and l_3 connected by transitions. The clock c is reset in the first transition. The invariant $c \leq 3$ in location l_2 refers to the clock c and enforces the following transition which is enabled by the guard $c \geq 2$. The transitions are labeled with actions α and ϕ, respectively. In the context of scheduling we use the transitions α for allocating a resource to a task and ϕ for releasing a resource of the task after completion.

TA models are created in a modular fashion as sets of interacting automata. The interaction between TA is established by synchronized transitions. Two transitions are synchronized when they have the same synchronization labels from *Act*. The corresponding automata change their locations simultaneously. The parallel composition of a pair of TA is performed by combining a set of TA into one composed global TA using the synchronization labels.

The parallel composition of two individual automata $A_1 = (L_1, l_{1,0}, F_1, \mathbb{C}_1, E_1, inv_1)$ and $A_2 = (L_2, l_{2,0}, F_2, \mathbb{C}_2, E_2, inv_2)$ is given by $A_{comp} = A_1||A_2 = (L_1 \times L_2, (l_{1,0}, l_{2,0}), F_1 \times F_2, \mathbb{C}_1 \cup \mathbb{C}_2, E, inv)$, with $l = (l_1, l_2)$ and $E = E_1(l_1) \wedge E_2(l_2)$. A transition in the composed automaton A_{comp} $(l, g, a, r, l') \in E$ exist iff there exist transitions $(l_1, g_1, a_1, r_1, l'_1) \in E_1$ in A_1 and $(l_2, g_2, a_2, r_2, l'_2) \in E_2$ in A_2, with guard g = $g_1 \wedge g_2$ satisfied, $r = r_1 \cup r_2$ and $a \in ((Act_1 \cup \{0\}) \times (Act_2 \cup \{0\}))$. The symbol 0 here denotes an empty action in which no state transition occurs. Similarly n individual TA can be composed step by step to obtain the composed automaton $A_{comp} := A_1||A_2||\cdots||A_n$.

An extension of TA with the notion of costs are *priced* TA (PTA) (Behrmann et al., 2001). A priced TA is equipped with an additional function P: $L \cup E \rightarrow \mathbb{R}^{\geq 0}$ which assigns a cost to transitions and cost rates to locations. The cost of staying in a location for d time units is given by $d \cdot P(L)$.

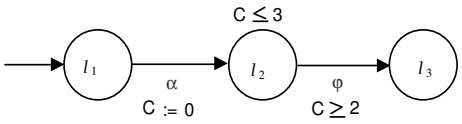

Figure 2: A simple timed automata.

2.1.2 Semantics

The semantics of a composed PTA A is defined in terms of a labeled transition system $(\mathbb{Q}, (l_0, u_0, 0), \Delta)$, with a state space $\mathbb{Q} \subseteq L \times \mathbb{R}^{|\mathbb{C}|} \times R^{\geq 0}$, the initial state given by $(l_0, u_0, 0) \in \mathbb{Q}$ with cost 0 and Δ representing the infinite transition relation. The state is a tuple (l, u, p), where l denotes the location, u denotes a vector of clock valuations and p represents the cost. Δ contains the following transitions:

1. Time transitions: $(l, u, p) \xrightarrow{\tau} (l, u + \mathbf{1}\tau, p + P(L) \cdot \tau)$ iff for every $\theta \in [0, \tau]$ the invariant $inv(l)$ evaluates to true. The vector $\mathbf{1}$ is composed of ones and has the dimension equal to the number of clocks in the set $\mathbb{C}$.

2. Discrete transitions: $(l, u, p) \xrightarrow{a} (l', u', p + P(e))$ where there exists a transition $e = (l, g, a, r, l')$ in E and $\theta \geq 0$ such that $g(u + \mathbf{1}\theta)$ is satisfied and $u' = (u'_1, u'_2, \cdots, u'_{|\mathbb{C}|})^T$ with $u'_i = 0$ for every $c_i \in r$ and $u'_i = u_i + \theta$ otherwise.

A sequence of states and state transitions from Δ constitutes a trace ρ of the composed priced automaton A given by $\rho = (l_0, u_0, 0) \xrightarrow{\delta_1} (l_1, u_1, p_1) \xrightarrow{\delta_2} \cdots$ with $\delta_i \in \Delta$. The state transitions in the trace are alternating time transitions and discrete transitions. Each trace corresponds to runs of the individual automata thereby characterizing a possible evolution of the system. An optimal trace $\rho*$ is a path from the initial state (l_0, u_0, 0) to a target state (l_n, u_n, p_n) with minimal p_n. The set of traces is infinite due to the fact that time delays τ are elements of time intervals defined by invariant and guard constraints. This then makes computing the optimal run from all possible runs a challenging task.

A solution to this problem is provided by the *zone abstraction* technique. A zone Z is an infinite set of possible clock valuations in a location l, which is expressed as a conjunction of finitely many inequalities such as c_i - $c_j \leq n$ or $c_i \leq n$ with c_i, $c_j \in \mathbb{C}$ and $n \in \mathbb{R}^{\geq 0}$. A symbolic state is formed by the combination of a location and the zone for the corresponding location (l, z). All symbolic states q^s constitute the symbolic state space Q_s. The symbolic semantics is defined in terms of a symbolic transition system in

which zones Z replace single clock valuations u. A symbolic trace ρ^s is a sequence of symbolic states in which the zones are computed by applying intersections, delay and reset operations defined for zones.

The main advantage of the zone abstraction is that the symbolic state space Q_s is enumerable and forms a *directed symbolic reachability graph* with nodes from Q_s and arcs from Δ_s. The aim of the cost optimal reachability analysis is to find the cheapest path from the initial state to a target symbolic state using weighted shortest path algorithms. The resulting shortest path obtained corresponds to the optimal symbolic trace ρ^{s*}. The shortest trace ρ^* can then be derived by picking the minimal clock valuations u_i^* from the zones Z_i^* of the symbolic states along the path.

3 SCHEDULING BASED ON TA

After the global composed timed automaton is constructed through *parallel composition*, a (cost-optimal) reachability analysis is performed to derive the schedules. This enumerative technique explores the symbolic state space step-by-step by evaluating the symbolic successor relation on-the-fly. The basic reachability algorithm from (Panek et al., 2007) is given below:

Algorithm 1 *A cost-optimal reachability algorithm*

```
1   cost* = ∞
2   W := {q0}; P := ∅
3   While W ≠ ∅
4       q = selectRemove(W)
5       If q ∉ P
6           P := P ∪ {q}
7           If cost(q) < cost*
8               If final(q)
9                   cost* := cost(q)
10              Else
11                  S := succ(q)
12                  S' := reduce(S)
13                  W := W ∪ S'
14              End
15          End
16      End
17  End
```

The algorithm operates on the nodes of the graph which contains the information on the location, clock valuations, accumulated cost and some additional data. The list of nodes that are to be explored is represented by the set $\mathcal{W}$ called the *waiting list*. The waiting list $\mathcal{W}$ initially contains only the initial node q_0 which in our case corresponds to the state where no jobs have been started. The set $\mathcal{P}$ represents the *passed list*, a list of already explored nodes, which is normally empty at the start of the search algorithm. $cost^*$ holds the cost of the best path found so far and q corresponds to the node that is currently explored. $\mathcal{S}$ is the set of immediate successors that can be reached by a single transition from q. The function $selectRemove(\mathcal{W})$ selects and removes a node from $\mathcal{W}$ which is minimal according to the ordering $\leq_{sel}$ defined by the tree search strategy. In order to avoid visiting and exploring the same node repeatedly, the test $q \notin \mathcal{P}$ is performed. $final(q)$ checks whether q is a target node. The function $succ(q)$ computes the immediate successor nodes of q and stores them temporarily in $\mathcal{S}$. The function *reduce* ($\mathcal{S}$) is used to remove those nodes that cannot contribute to a better path or to the optimal run. The test $cost(q) < cost^*$ compares the cost value of the current node with the best cost value found, thereby avoiding exploration of non-optimal runs if the comparison fails. The decision criteria for the function *reduce* are based on timed transitions to successor nodes and will be discussed in detail in the following sections.

4 REDUCTION TECHNIQUES

This section discusses three methods to make the search for the optimal schedule more efficient.

4.1 Immediate Schedules

In (Abdeddaim and Maler, 2001), it has been shown that for job-shop problems with the makespan minimization as objective, the trace corresponding to an optimal schedule is always *immediate*. A non-immediate trace contains a transition sequence $\cdots \rightarrow (l,u) \xrightarrow{\tau} (l,u+\tau) \xrightarrow{\alpha} (l',u') \rightarrow \cdots$ where the α transition taken at $(u+\tau)$ is already enabled at $(u+\tau')$ with $\tau' < \tau$. Such traces exhibit periods of useless waiting in the schedule. Waiting is useless if, e.g., no tasks are started while the required resource is available (see Figure 3). Non-immediate traces can be transformed to immediate traces by reducing such waiting periods to zero. A reduction by $\tau - \tau'$ leads to the partial trace $(l,u) \xrightarrow{\tau'} (l,u+\tau') \xrightarrow{\alpha} (l',u')$ and can reduce the overall length of the corresponding schedule. This infers that it is sufficient to search for immediate traces in the symbolic state space. The effect of the reductions of the waiting times is that the zones in the symbolic states collapse to single clock valuations and the symbolic reachability graph is reduced to nodes representing symbolic states (l,u). The arcs

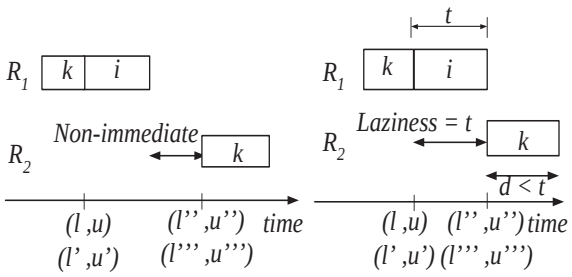

Figure 3: Non-immediate and Lazy schedule.

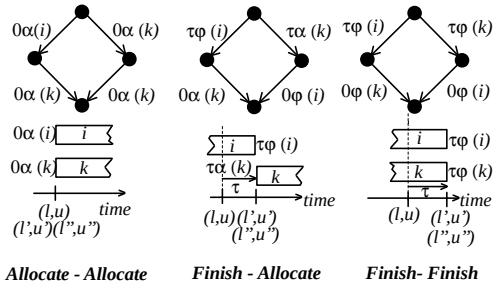

Figure 4: Sleep set configurations.

of the symbolic reachability graph become timed transitions: pairs of discrete and time transitions τa in which τ is the smallest possible delay needed to satisfy the guard of the discrete transition with the action a. Zones Z in the symbolic states can be replaced by vectors of clock valuations u such that (l,u) is a state of an immediate trace.

4.2 Ordinary and Strict Non-Laziness

In (Abdeddaim and Maler, 2001) also a concept to avoid unnecessary waiting known as *non-laziness* was introduced. Consider a case in which a current node (l,u) with two discrete transitions as successors namely $\alpha_i^{R_1}$ and $\alpha_k^{R_2}$ that allocate resources R_1 and R_2 to jobs i and k, exists. The releasing transitions are represented as $\phi_i^{R_1}$ and $\phi_k^{R_2}$. The transition $\alpha_k^{R_2}$ was already enabled at the same time when the transition $\alpha_i^{R_1}$ was enabled. In the partial trace $(l,u) \overset{0\alpha_i^{R_1}}{\rightarrow} (l',u') \overset{t\phi_i^{R_1}}{\rightarrow} (l'',u'') \overset{0\alpha_k^{R_2}}{\rightarrow} (l''',u''')$ the enabled transition $\alpha_k^{R_2}$ is taken at a later stage by waiting to finish job i thereby introducing laziness. In order to remove laziness in a trace, waiting in a node for t or more than t units is forbidden when all the following conditions are satisfied:

(a) there exists a successor of (l,u) in which an operation of duration d can be started by an enabled discrete transition that allocates a resource to a job.

(b) The duration d of this operation satisfies $d \leq t$

(c) if the allocating transition is permanently enabled and the resource allocated by that transition is available for more than $d-t$ units (provided $d > t$).

The term *ordinary laziness* is used since a short waiting time according to conditions (b) and (c) is permitted. A stricter condition with respect to waiting is the *strict non-laziness*, which is obtained by removing conditions (b) and (c). Thus, whenever there exists at least one timed transition to start a new task waiting is forbidden in a state. In comparison to ordinary non-laziness this scheme prunes more time successors. However the strict condition might also prune the optimal trace as in some problems the optimal schedules may have a short waiting period.

4.3 Sleep-Set Method

The sleep-set method was originally introduced in (Godefroid, 1991) to reduce possible partial orders and to avoid exploring obsolete runs in the reachability graph. The application of the method to job-shop problems was first implemented in (Panek et al., 2006). In the symbolic reachability graph different symbolic traces which correspond to the same schedule may exist. This is due to the *interleaving semantics* of the TA model. Such traces includes transitions that are taken at the same absolute time but can be traversed in different logical orders. In general, such transitions share a common symbolic state at the start and at the end of the transition with the same cost. Three common and possible configurations that can be identified in the reachability graph are shown in Figure 4. The left part of the figure shows a pair of α transitions that can be traversed in two different ways from a common start node (l,u). The possible logical order of the transition pair are $\{\alpha(i),\alpha(k)\}$ and $\{\alpha(k),\alpha(i)\}$. Both allocating transitions share the same end state, cost and corresponds to the same schedule shown. Such traces form so called *diamond structures* in the reachability graph. Similar kind of structures arising due to pairs of α and ϕ transitions and the corresponding schedule are shown in the same figure. Thus, it is evident that only one of the traces in the diamond structure has to be explored and the rest can be pruned.

The idea of the sleep-set method to remove obsolete traces is to compare pairs of immediate successor transitions from a node and mark some of them as candidates to be pruned. However pruning of marked candidates can only take place when both transitions are independent. Pairs of timed transitions are said to be independent if the corresponding operations do not require the same resource and taking one transition does not block the other. It should be noted that

pairs of ϕ and mixed pairs of α and ϕ transitions are independent as a single resource cannot finish two operations at the same time. Not all pairs of α transitions are always independent as they might compete for the same resource. An outgoing transition $\tau\delta$ is said to be time persistent in a given symbolic state (l,u,p) at time t, if the discrete transition δ is taken at $t+\tau$ for every partial run starting from (l,u,p) with $\tau \geq 0$. Given a symbolic state with outgoing transitions $\tau\delta_i$ and $\tau\delta_j$, $\tau\delta_j$ is marked as a candidate for reduction if $\tau\delta_i$ is time persistent and vice versa. If both are time persistent, then the transition with a higher resource index is removed. This reduction however does not work when comparing two α transitions as they are not time persistent. An allocated task must always be finished. This shows that all outgoing transitions that release a resource are time-persistent and appear in future parts of the trace. Since α transitions are not time persistent, in general they cannot be removed. However it is possible to identify *eager* α transitions which become implicitly time persistent in a non-lazy run. A transition $\tau\alpha_i$ which is an immediate successor $\mathcal{S}$ of a state is called eager if for all $\tau_j\phi_j \in \mathcal{S}$ either (a) $\tau + dur(\alpha_i) \leq min_j\{\tau_j|\tau_j\phi_j \in \mathcal{S}\}$ or (b) $\tau \geq max_j\{\tau_j|\tau_j\phi_j\}$ or (c) $\tau \geq 0$ if no ϕ transitions exist in $\mathcal{S}$. A strict approach is to use the same strategy to remove transitions if they are independent and not to check whether the transition is time persistent. This strict pruning strategy causes a considerable reduction of the reachability graph to a considerable amount. However, it may remove the optimal trace as a pruned transition can be part of the optimal trace.

5 EXAMPLE PROCESS

In order to depict the applicability and the effectiveness of the approach discussed here, the case study presented by (Bauer et al., 2000) is considered. The case study is a multi-product chemical batch plant with 3 stages in which two end-products, ***X*** and ***Y***, are produced from three raw materials, ***A***, ***B*** and ***C***. The production process considered is: one batch of material ***A*** and ***C*** each reacts to produce one batch of product ***X***; similarly one batch of ***B*** and ***C*** reacts to produce one batch of product ***Y***. The plant executing the production process is shown in Figure 5.

The first stage consists of three buffer tanks B11, B12 and B13 that store the raw materials ***A, C*** and ***B***. The second stage consists of three reactors R1, R2 and R3. Each reactor may be filled from each raw material buffer tank in the first stage; implying that it is possible to produce either product ***X*** or ***Y*** in each reactor. After processing the materials a reactor may

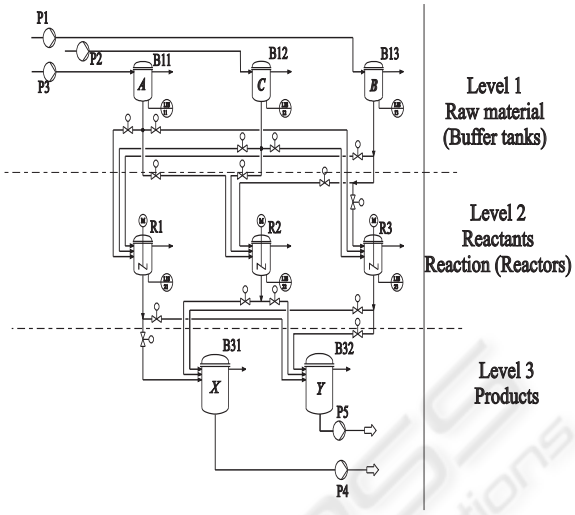

Figure 5: The structure of the example process.

contain one batch of the product (resulting from two batches of raw material). The third stage consists of two buffer tanks B31 and B32 which are used to store the end-products ***X*** and ***Y***. Each of the tanks has a maximum capacity to store three batches of the product. The proposed scheduling problem is to derive production schedules to produce 2 batches of the end-products with a minimum makespan. After being processed by the reactors the materials must be drained into the buffer tanks in the third stage immediately.

RTN description of the problem considered is shown in Figure 6. Materials ***A, C*** and ***B*** are represented as S_1, S_2 and S_3. The tasks T_1, T_2 and T_3 represents the task of pumping materials ***A, C*** and ***B*** into buffer tanks B11, B12 and B13, respectively. The tasks T_4, T_6 and T_8 represent the step of draining one batch of ***A*** from B11 into the reactors R1, R2 and R3. Similarly, tasks T_{11}, T_{13} and T_{15} represent draining of material ***B*** from B13 into the reactors. The tasks T_5, T_7 and T_9 represent mixing one batch of ***C*** from B12 with one batch of ***A***, already present in R1, R2 and R3 in order to produce ***X***. The tasks T_{10}, T_{12} and T_{14} represent similar actions for producing one batch of ***Y***. The tasks T_{16}, T_{17} and T_{18} represent the step of draining one batch of ***X*** from R1, R2 and R3, respectively, into the buffer tank B31. Tasks T_{19}, T_{20} and T_{21} represent draining one batch of ***Y*** from R1, R2 and R3 into tank B32. The tasks T_{22} and T_{23} represent draining of two batches of ***X*** from B31 and two batches of ***Y*** from B32 to the product storage, respectively. The constraint on the waiting time is expressed by the *places* P_2, P_4, P_6, P_8, P_{10} and P_{12}. The *places* P_1, P_3, P_5, P_7, P_9 and P_{11} are established to ensure that material ***C*** is drained into the reactors only after either ***A*** or ***B*** has been drained.

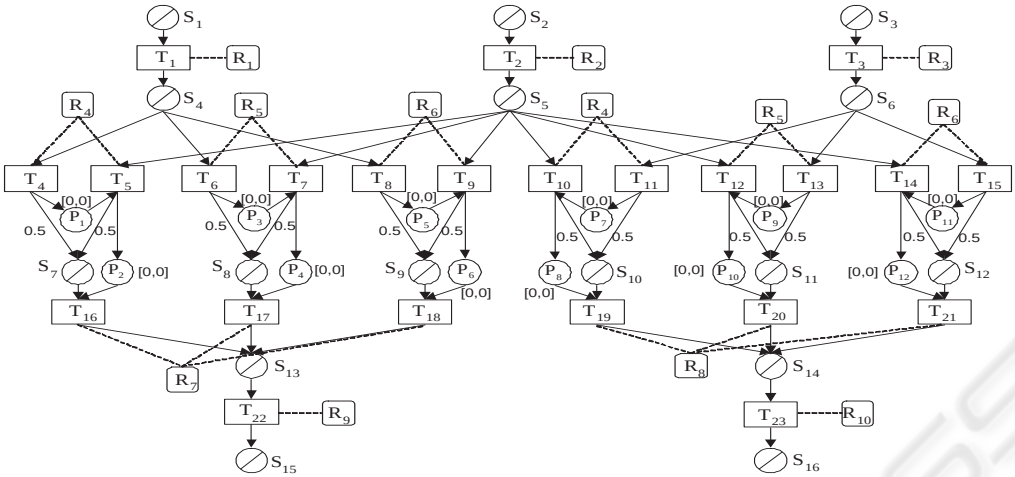

Figure 6: The RTN of the example process.

5.1 TA Model

The TA model consists of 10 resource automata to model the individual resources. The places are modeled as separate automata. The resource automata and the place automata have one clock each to measure the task durations and time interval for storage. An additional global clock which is never reset is included in the model to measure the makespan. The resource automaton for resource R_4 and the place automaton P_2 are shown in Figure 7. The automaton R_4 has one idle location and four busy locations indicating the execution of either of the tasks T_4 or T_5 or T_{10} or T_{11}. Tasks T_4 and T_{11} process one batch of material S_4 and S_6 with task duration of 15 and 11 units. The timing constraint between task T_5 and task T_{16} is modeled by the place automaton P_2 on the right side. Task T_5 after finishing its process in R_4 produces a token in the place P_1 by taking the synchronized transitions in R_5 and P_2 with the label ϕ_5. The clock C_{12} is reset and forces P_2 to leave the location *filled* without waiting. This happens by starting task T_{16} and by taking the synchronized transitions with label α_{16} in R_7 and P_2. An automatic translation from the RTN to the TA model was performed by the tool TAOPT (Panek et al., 2005).

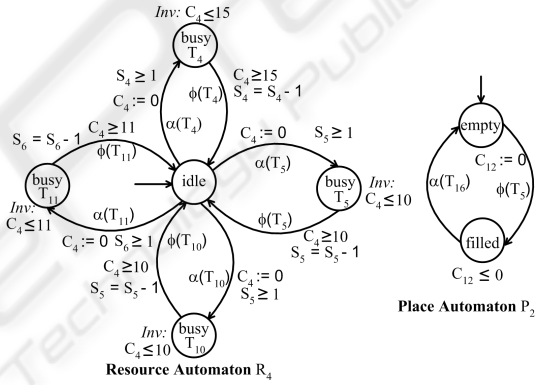

Figure 7: Structure of the resource and place automaton.

5.2 Computational Experiments

The scheduling problem presented in the example process was solved by a Xeon machine with a CPU speed of 3.06 GHz, 2.0 GB of memory and a linux operating system. For the cost optimal symbolic reachability analysis, the software tool TAOPT (Panek et al., 2005) which supports different state space reduction techniques as discussed above: immediate traces, (strict) *non-lazy runs* and *sleep-set method* was used. A combination of the depth-first and best-first strategy was used for exploring the graph. Among the nodes in the waiting list, the node with the maximum depth is given a higher priority, among the nodes with the same depth, priority is given to the node with a minimum cost. If two or more nodes have the same depth and cost, one of them is chosen randomly.

The Table 1 shows the results obtained by restricting the search to only immediate traces represented as *None* and with other state space reduction techniques: ordinary non-lazy runs ($NL_{ordinary}$), strict non-lazy runs and strict sleep-set method (SSM_{strict}). For all tests the complete reachability graph was explored. The second column that represents the total time needed to compute the reachability graph and the total number of nodes explored in the the third column reveal that the strict sleep-set method reduces the state space and computational effort considerably. On the other hand it prunes the optimal solutions. The fourth column represents the time taken in CPU seconds to

Table 1: Computational results.

			First solution			Best solution
Reduction technique	Total time	Total nodes	time	nodes	C_{max}	C_{max}
None	58.832	493167	0.032	1875	100	98
$NL_{ordinary}$	2.104	37483	0.048	2157	98	98
SSM_{strict}	0.476	13374	0.120	6492	131	128
$SSM + NL_{ordinary}$	1.084	28583	0.040	681	102	98

compute the first feasible solution when different reduction techniques were used. The test with normal sleep set method which compares and removes only time persistent pairs of transitions in combination with ordinary non-lazy runs was performed. This is represented as $SSM + NL_{ordinary}$ in the table 1. The combination of $SSM + NL_{ordinary}$ found the optimal solution. This is due to the fact that only time persistent transitions were pruned. No solutions could be found for the strict non-laziness setting when applied alone or in combination with the strict sleep-set method and the normal sleep-set method. It should be noted that the reduction techniques ordinary non-lazy runs and the normal sleep-set method are the only combinations which theoretically guaranty that the optimal solution is not pruned. The strict sleep set method in general should only be used as a heuristic.

6 CONCLUSION

A new and a straightforward approach to model scheduling problems in multi-product batch plants and to derive production schedules based on symbolic reachability analysis of TA has been introduced. The experimental results show that the method can solve scheduling problems in process industries efficiently within a limited computation time. The main aspect to be considered is the time required by the TA approach to specify and formulate the scheduling problem. It is relatively easy when compared to formulating it as MILP which are usually cumbersome. The efficiency of the approach to derive solutions with reduced computational effort have been achieved with different search space reduction techniques.

Future work on the TA based approach will focus on online scheduling problems in which the arrival time of jobs is not known in advance and subject to random perturbations.

ACKNOWLEDGEMENTS

The authors gratefully acknowledge the financial support by the Graduate School of Production Engineering and Logistics at the University of Dortmund.

REFERENCES

Abdeddaim, Y. and Maler, O. (2001). Job-shop scheduling using timed automata. *Computer Aided Verification (CAV), Springer*, pages 478–492.

Alur, R. and Dill, D. (1994). A theory of timed automata. *Theor. Comp.Science*, 126(2):183–235.

Barthomieu, B. and Diaz, M. (1991). Modeling and verification of time dependent systems using timed petri nets. *IEEE Transactions Software Engineering*, 126(2):259–273.

Bauer, N., Kowalewski, S., Sand, G., and Loehl, T. (2000). A case study: Multi product batch plant for the demonstration of scheduling and control problems. In *ADPM2000 Conference Proceedings-Hybrid Dynamic Systems*, pages 383–388. Shaker.

Behrmann, G., Fehnker, A., Hune, T., Peterson, P., Larsen, K., and Romjin, K. (2001). Efficient guiding towards cost-optimality in **UPPAAL**. In *Proceedings TACAS'01*, pages 174–188.

Godefroid, P. (1991). Using partial orders to improve automatic verification methods. *In Proc. 2nd Workshop on Computer Aided Verification*, 531 of LNCS:176–185.

Kallrath, J. (2002). Planning and scheduling in the process industry. *OR Spectrum*, 24:219–250.

Panek, S., Engell, S., and Lessner, C. (2005). Scheduling of a pipeless multi-product batch plant using mixed-integer programming combined with heuristics. In *European Symposium on Computer Aided Process Engineering, ESCAPE 15*, pages 1033–1038.

Panek, S., Engell, S., and Stursberg, O. (2006). Efficient synthesis of production schedules by optimization of timed automata. *Control Engineering Practice*, pages 149–156.

Panek, S., Engell, S., Subbiah, S., and Stursberg, O. (2007). Scheduling of multi-product batch plants based upon timed automata models. *Submitted to Comp. Chem. Eng*.

Pantelides, C. (1994). Unified frameworks for optimal process planning and scheduling. In *Proceedings 2nd conference Foundations of Computer Aided Process Operations*, pages 253–274.