Optimal Task Scheduling in a Flexible Manufacturing System using Model Checking

Robi Malik * **Patrícia N. Pena** **

* *Department of Computer Science, University of Waikato, Hamilton, New Zealand (e-mail: robi@waikato.ac.nz)*
** *Department of Electronics Engineering, Universidade Federal de Minas Gerais, Belo Horizonte, MG, Brazil (e-mail: ppena@ufmg.br)*

Abstract: This paper demonstrates the use of model checking to solve the problem of optimal task scheduling in a flexible manufacturing system. The system is modelled as a discrete event system, for which the least restrictive safe behaviour is synthesised according to supervisory control theory. Then timing constraints are added to the model in the form of extended finite-state machines, and time-optimal schedules are computed using the discrete event systems and model checking tool *Supremica*. In the case study considered in this paper, which previously was only solved heuristically, the method successfully produces optimal schedules to manufacture up to 30 products of two different types. The method is furthermore used to find an optimal cycle, solving the scheduling problem of the case study for an arbitrary number of products in optimal or asymptotically close to optimal time.

Keywords: Applications; Performance evaluation, optimization; Supervisory control theory.

1. INTRODUCTION

The *task scheduling* problem in manufacturing systems is of great importance in industry (Saygin and Kilic, 1999). The aim of task scheduling is to organise a set of tasks over a set of resources in order to optimise some criterion measure. A suitable solution provides fault-free operation while utilising the available resources with high efficiency. Without the use of systematic techniques to create the production programs it is not possible to guarantee such features. Therefore, there has been a significant research effort in the past decades to develop systematic tools to deal with the task scheduling problem (Niebert and Yovine, 2000; Ware and Su, 2017).

In the majority of works, there is an implicit assumption that a schedule can be executed directly as developed, and there is no support for possible disruptions. Aytug et al. (2005) state that the inability of much scheduling research to address the general issue of uncertainty may be considered as a major reason for the lack of influence of scheduling research on industrial practice. They consider three key dimensions of uncertainty—cause, context, and impact—that can help to categorise problem formulations. *Cause* may be tooling not available, *context* may be a bottleneck on Monday morning, and the *impact* may be a delay in setup—the machine cannot start when expected. This paper is concerned with uncertainty caused by possible delays that cause loss of synchrony between the schedule and the actual system (Wu and Zhou, 2012).

The *supervisory control theory* of *discrete event systems* is a framework that aims to develop controllers, which automatically apply commands in a least restrictive way (Ramadge and Wonham, 1989). By explicitly considering *uncontrollable* events, the framework captures the uncertainty caused by variations in operation durations. *Least restrictiveness* ensures that only events leading the system to unsafe or undesirable states are disabled, while all other options remain available to optimisation.

Several authors have used supervisory control to address task scheduling problems. Huang and Kumar (2008), Pinha et al. (2011), and Ware and Su (2017) translate optimisation problems into discrete event systems and propose theoretical solutions and algorithms for the problem. Implementations of these algorithms are not yet widely available, and it is not clear whether they can solve large instances of scheduling problems.

The problem solved in this paper is to find a schedule that minimises the makespan for the production of a batch of products for a flexible manufacturing system introduced by de Queiroz et al. (2005). This case study previously was only solved heuristically, most recently by Pena et al. (2016). Attempts to solve the case study with the timed model checker *Uppaal* have only produced schedules for small instances of the problem (Gontijo, 2015).

This paper uses the discrete event systems tool *Supremica* (Åkesson et al., 2006; Malik et al., 2017) to develop a full model of the timed behaviour of the case study, making it possible to compute optimal schedules to produce a fixed number of up to 16 products, and to compute optimal cycles to solve the task scheduling problem for an arbitrary number of products. In the following, Section 2 presents the case study and describes the modelling of the task scheduling problem. Afterwards, Section 3 shows the computed solutions for fixed-size production batches with

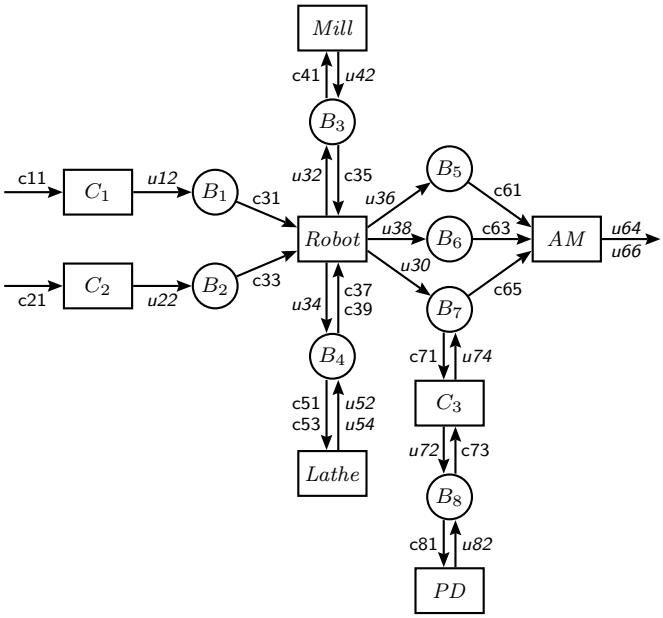

Fig. 1. Flexible manufacturing system layout.

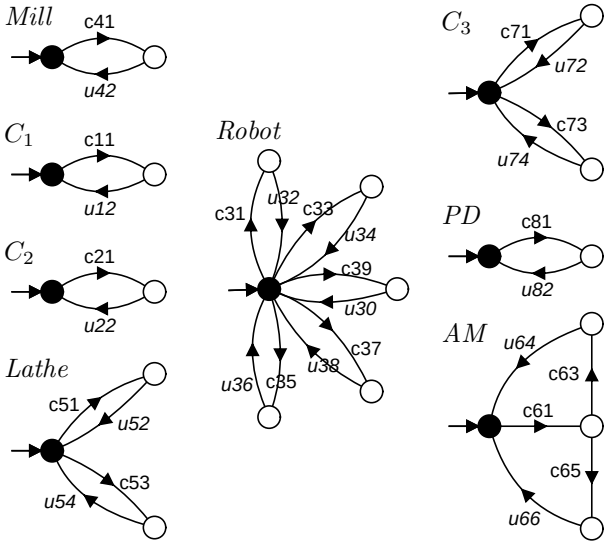

Fig. 2. Flexible manufacturing system plants.

up to 30 products, and Section 4 shows the computation of an optimal cycle and the resulting generalisation for an arbitrary number of products. Finally, Section 5 adds concluding remarks.

2. MODELLING

This section describes the model developed for the case study of this paper, the *flexible manufacturing system*. First, subsection 2.1 describes the open-loop behaviour of the plant, and subsection 2.2 shows how this behaviour is restricted to be safe by synthesis. Afterwards, subsections 2.3 and 2.4 explain how the operation times are modelled to specify the optimisation problem.

2.1 The Flexible Manufacturing System

Fig. 1 shows the layout of the flexible manufacturing system (de Queiroz et al., 2005). It consists of eight devices: the *Lathe*, *Mill*, painting device (PD), and assembly machine (AM) perform production tasks, while the *Robot* and three conveyors (C_1, C_2, C_3) move workpieces between machines and buffers. There are eight *buffers* ($B_1, \ldots, B_8$) that act as intermediate deposits, each with capacity for one workpiece. The arrows in Fig. 1 indicate the events that represent the flow of workpieces.

This system produces two types of products from raw blocks and pegs: blocks with a conical pin on top (Product A) and blocks with a painted cylindrical pin (Product B). Blocks enter the system through conveyor C_1 and are delivered by the *Robot* to the *Mill*; after milling the *Robot* puts the completed block into buffer B_5 waiting for assembly. Pegs enter the system through conveyor C_2 and are delivered by the *Robot* to the *Lathe*, which cuts them to become a cone (event c51) or cylinder (event c53). Cones are placed by the *Robot* into buffer B_6; cylinders are placed into buffer B_7, where conveyor C_3 takes them to the painting device PD and back. The assembly machine AM first takes a block from B_5 and then either a cone from B_6 or a painted cylinder from B_7 to manufacture product of a type A or B.

The open-loop behaviour of this plant is modelled by the discrete event system in Fig. 2. As usual in supervisory control (Ramadge and Wonham, 1989), the set of events is partitioned into the *controllable* events, which can be disabled by a controlling agent, and the *uncontrollable* events, which occur spontaneously and cannot be disabled. In this paper, the controllability status is distinguished by the event names starting with "c" or "*u*".

In the case study, the plant model is structured into *operations* that start with a controllable event and finish with an uncontrollable event. For example the operation of conveyor C_1 is started with the controllable event c11 and finishes with the uncontrollable event *u12*. The only exception is the assembly machine AM, which takes two controllable events before finishing. The underlying assumption is that a control strategy should be implemented that decides when and in what order operations should start, but there is no control over their completion times.

2.2 Using Supervisory Control to Impose Constraints

The open-loop plant as modelled in Fig. 2 permits unsafe behaviour, because buffer overflow cannot always be avoided due to the presence of uncontrollable events, and it also permits deadlocks. Therefore, control specifications are introduced to rule out the possibility of buffer overflow or underflow, and *synthesis* (Ramadge and Wonham, 1989) is used to obtain *supervisors* that restrict the plant to the largest possible safe and nonblocking behaviour.

Small modular supervisors, both controllable and nonblocking, can be computed automatically by *compositional synthesis* (Mohajerani et al., 2014) followed by *supervisor reduction* (Su and Wonham, 2004). Alternatively, this paper uses the supervisors proposed by Pena et al. (2016), which are synthesised for controllability using *local modular control* (de Queiroz and Cury, 2000), reduced, and afterwards verified to be nonblocking. The resulting finite-state machines are shown in Fig. 3. These supervisors interact with the plant in strict lock-step synchronisation

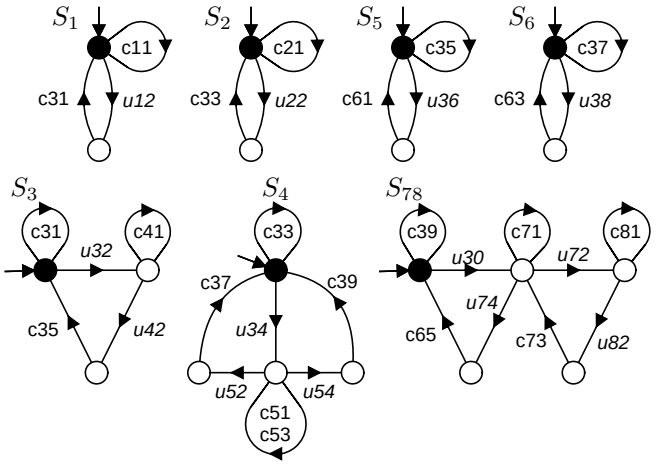

Fig. 3. Supervisors for the flexible manufacturing system.

Table 1. Operation times.

Device	Event	Time	Device	Event	Time
C_1	c11	25	C_2	c21	25
Robot	c31	21	*Mill*	c41	30
	c33	19	C_3	c71	25
	c35	16		c73	25
	c37	24	*AM*	c61	15
	c39	20		c63	25
Lathe	c51	38		c65	25
	c53	32	*PD*	c81	24

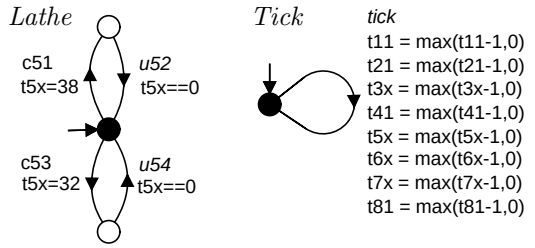

Fig. 4. EFSMs plants to capture timing constraints.

by handshaking on common events (Hoare, 1985). The state machines in Figs. 2 and 3 combined give the complete model of the *closed-loop system*.

The schedule optimisation discussed in the following uses this controlled system as the starting point. As the supervisors are synthesised based on an untimed model, they will ensure safe control and prevent buffer overflow no matter what the delay is between the start and end of operations. Also, *least restrictiveness* ensures that the optimisation has the maximum freedom of choice possible.

2.3 Modelling Time with Extended Finite-State Machines

For optimisation, the durations of the operations must be included in the model. Therefore, each operation (identified by the controllable event to start it) is assigned a fixed duration as per Table 1.

The timed behaviour is modelled using Extended Finite-State Machines (EFSM) (Chen and Lin, 2000; Sköldstam et al., 2007), which allow for the addition of guards and actions to the transitions. Each device of the flexible manufacturing system is associated with a bounded-range integer variable that records the time still needed to complete its current operation. For example, the *Lathe* is modelled using the EFSM in Fig. 4, with the variable t5x. On occurrence of the controllable event c51, the *update* t5x = 38 means that the variable assumes the value 38 after the transition. Thus, the timer is initialised to the duration of operation c51 when it starts. For the uncontrollable event *u52*, the *guard* t5x == 0 means that the transition is only possible when the variable value is zero. Thus, the lathe's operation can only finish when the timer has counted back down to zero.

Similarly, the duration of operation c53 is modelled as 32 according to Table 1. The variable t5x is constrained to the finite range $\{0, \ldots, 38\}$, because 38 is the duration of the longest operation performed by the *Lathe*. Similar timers are added to all plant components in Fig. 2.

The timers are counted down with the passing of time, which is modelled by an uncontrollable event and associated plant EFSM called *Tick*, also shown in Fig. 4. The updates on the transition are executed simultaneously on each tick of the clock. For example, the update t5x = $\max(\mathsf{t5x}-1, 0)$ means that the variable t5x is decremented by one unless it has already reached zero.

The timing model only imposes constraints on event occurrences of the untimed model. Therefore, it is clear that a supervisor synthesised for the untimed model continues to be controllable (but not necessarily nonblocking) for the timed model. The use of uncontrollable events in connection with the buffers in this model ensures that, after synthesis of a supervisor that prevents buffer overflow, the occurrence of an uncontrollable event never disables any controllable events that have been enabled before. Then a production schedule given by a sequence of controllable events can always be executed independently of the precise timing of uncontrollable events.

Also, the timing model only restricts the operations to take at least the number of ticks indicated in Table 1 to complete; they may take longer. This is enough for optimisation, because if an optimal path is found, it will use the smallest number of ticks possible.

2.4 The Optimisation Problem

The goal of optimisation is to find the fastest way to produce a given number of products of type A and B. If NA and NB are the required numbers of products, then this amounts to finding a sequence of events through the model, with the smallest number of ticks possible, that includes NA occurrences of event *u64* and NB occurrences of event *u66*.

This objective is modelled by adding two further variables countA and countB, with initial value 0, that count the occurrences of events *u64* and *u66*, using the EFSMs $Product_A$ and $Product_B$ in Fig. 5: the update countA +=1 is a shorthand for countA = countA + 1, and means that the counter is incremented by 1 on occurrence of event *u64*, i.e., on production of a type A product. The uncontrollable event *done* is only possible when both counters have reached their target values, because the guards in the two EFSMs are combined by conjunction.

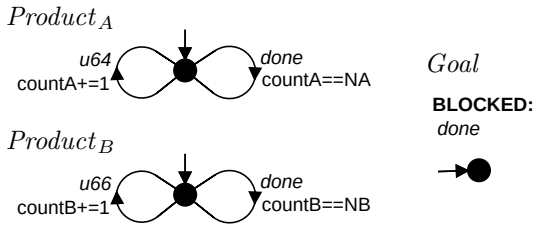

Fig. 5. EFSMs to capture the optimisation goal.

Finally, the EFSM *Goal* stipulates that *done* can never happen. This is achieved by declaring the event *done* as *blocked* (Malik et al., 2011), which means that it is included in the alphabet but not on any transition. When a model checker is asked to verify or refute this assertion, it will produce a *counterexample*, which is a sequence of events that includes *done*, thus showing how the flexible manufacturing system can produce the requested products.

In a flexible manufacturing system, the devices can work at the same time and faster completion can be achieved by performing operations concurrently. The best possible runtime under concurrency associated with a sequence of events is called the *makespan*, and the goal of optimisation is to find a *schedule*, i.e., sequence of events, with the minimum makespan.

An important observation in this example is that the non-tick events for given numbers NA and NB of products are always the same; only their order can be changed. Therefore, any shortest path that ends with the done event must be a schedule with the smallest number of ticks and thus minimum makespan.

3. OPTIMAL SOLUTIONS FOR FIXED-SIZE BATCHES

The model of the flexible manufacturing system has been entered in the discrete event systems tool *Supremica* (Åkesson et al., 2006; Malik et al., 2017), which supports EFSMs in the form described in the previous section (Malik et al., 2011). All the plants and supervisors in Figs. 2 and 3 have been entered, with timing information for the plants as per Fig. 4, and the additional plants $Product_A$ and $Product_B$ as per Fig. 5.

Finally, the *Goal* in Fig. 5 has been added as a *property*, which was checked for *language inclusion*. *Supremica's* language inclusion check determines whether the behaviour of the system is contained in the behaviour of the property. In this case, the property *Goal* allows all behaviour that does not include the event *done*, so that, if *done* is ever possible in the system, it will be flagged as a violation of language inclusion. If *Supremica* determines that language inclusion is violated, it produces a counterexample that shows how the property is violated. In this case, the counterexample is a sequence of events including *done*, which can be interpreted as a schedule for production of the requested products.

There are various model checking algorithms to check language inclusion. Two of *Supremica*'s algorithms use breadth-first search and guarantee that the computed counterexamples are of minimum length. Given the observation in section 2.4 that a shortest counterexample also means minimum makespan, these algorithms produce counterexamples that represent optimal schedules.

Table 2. Optimisation results

NA	NB	State space	Runtime	Makespan
1	1	$2.452 \cdot 10^{10}$	0.6 min	238
2	2	$1.512 \cdot 10^{11}$	2.7 min	395
3	3	$3.569 \cdot 10^{11}$	5.0 min	552
4	4	$6.221 \cdot 10^{11}$	7.5 min	709
5	5	$9.463 \cdot 10^{11}$	10.2 min	866
6	6	$1.329 \cdot 10^{12}$	14.0 min	1023
7	7	$1.771 \cdot 10^{12}$	17.5 min	1180
8	8	$2.272 \cdot 10^{12}$	21.5 min	1337
9	9	$2.832 \cdot 10^{12}$	24.5 min	1494
10	10	$3.451 \cdot 10^{12}$	30.2 min	1651
11	11	$4.129 \cdot 10^{12}$	36.4 min	1808
12	12	$4.866 \cdot 10^{12}$	42.0 min	1965
13	13	$5.661 \cdot 10^{12}$	45.1 min	2122
14	14	$6.516 \cdot 10^{12}$	53.4 min	2279
15	15	$7.429 \cdot 10^{12}$	61.7 min	2436

The first is an *explicit* algorithm, which remembers every visited state. It is limited by the amount of available memory and typically handles up to 10^8 states (Malik, 2016). While the untimed model of the flexible manufacturing system only has 812,544 reachable states, the timing constraints lead to state-space explosion. The explicit algorithm can only compute counterexamples for up to three products (one type A and two type B products or vice versa).

The second algorithm uses *Binary Decision Diagrams (BDD)* (Clarke et al., 1999) to represent state sets symbolically, which greatly reduces memory consumption. The BDD-based algorithm can compute schedules for up to 15 products of each type, and Table 2 gives an overview of its results. For the numbers NA and NB of products, the table shows the number of states in the explored state space, the time taken to compute, and the makespan of the computed optimal schedule. The computation was done on a standard PC with a 2.8 GHz CPU and 16 GiB of RAM. The algorithm was breadth-first search with a disjunctively partitioned transition relation (Clarke et al., 1999), using the BDD package CUDD (Somenzi, 2005) with an initial node table size of 1,000,000.

Supremica produces a single counterexample in the form of an event sequence. By interpreting the sequence appropriately, it can be presented as a *Gantt chart* (Clark et al., 1922) such as Fig. 6. The figure shows an optimal schedule for one product of each type. The operations are identified by the controllable events that start them, with start times from the counterexample. The end times are computed based on the anticipated durations of the operations.

4. OPTIMAL CYCLE

In practice, the desired number of products is often too big to permit computation of a fixed-length schedule, or it is unknown a-priori. In this case, it is common to identify the cyclic behaviour of the system to derive a production plan of arbitrary length. This amounts to the identification of a cycle with optimum throughput, which is a difficult problem in general due to the infinite number of possible cycles. Yet for the flexible manufacturing system studied

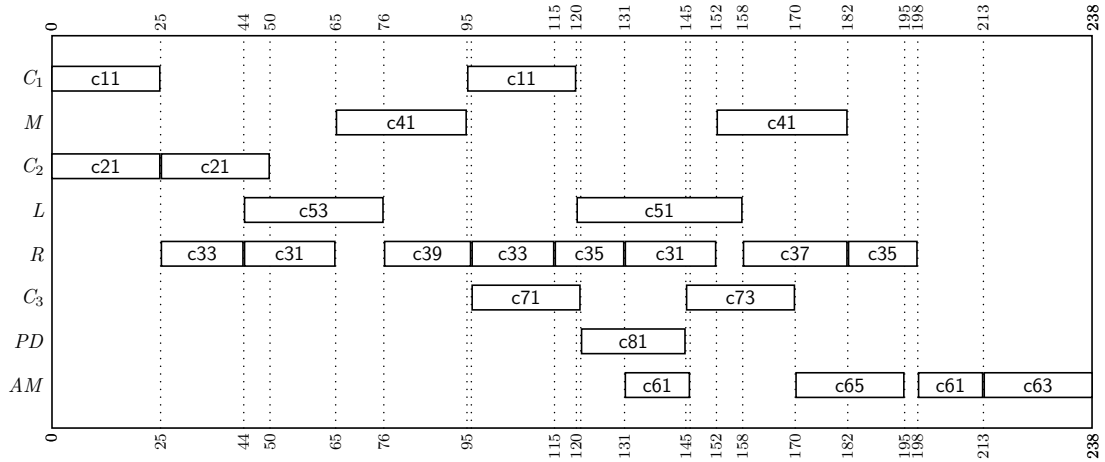

Fig. 6. Optimal schedule for $\mathsf{NA} = 1$ and $\mathsf{NB} = 1$.

in this paper, an optimal cycle can be found using methods similar to those described above.

The key to finding a cycle is to determine a suitable *entry state*, i.e., a system state that can be visited repeatedly while producing. Inspection of Fig. 6 suggests the configuration at step 95, where there is a new peg in buffer B_2, a milled block in buffer B_3, and an unpainted cylinder in buffer B_7. However, this entry state leads to a cycle with idle time waiting for the mill to complete. With some trial and error, a better entry state was determined such that:

- there are new workpieces in buffers B_1 and B_2, waiting to be picked up by the robot R;
- there is an unpainted cylinder in buffer B_7, waiting to enter conveyor C_3;
- there is a block being processed in the mill M, ten time steps before completion.

To compute a cycle for a given entry state, the model is changed so that the initial state of all components is the entry state, and the event *done* indicating completion is only possible in that state. This amounts to changing the initial states of the plants and specifications and the initial value of the mill timer variable t41, adding selfloops labelled *done* to the initial states of all plant components except the *Mill*, and adding a condition that *done* is only possible when $\mathsf{t41} = 10$. Then *Supremica*'s language inclusion check produces a counterexample that starts and ends in the entry state while producing the number of products given by NA and NB.

Fig. 7 shows the cycle obtained for $\mathsf{NA} = \mathsf{NB} = 1$, which has a duration of 157 ticks. The Gantt chart shows that the robot R is busy for the entire cycle except for one tick. As its eight operations are required to produce one product of each type, an optimal cycle must have at least 156 ticks. More careful analysis shows that a cycle of 156 ticks is not possible: the c51 operation of the lathe L takes 38 ticks, and while it is running, the robot can only perform two operations servicing the mill, c35 and c31, with a combined duration of 37 ticks. This means that every c51 operation entails one tick with the robot idle, and since one such operation is required to produce a type A product, it can be concluded that the cycle in Fig. 7 is optimal.

To complete the production plan, two further models are created to compute an *initialisation* and a *shutdown* sequence. The initialisation sequence starts from the initial state of the model and ends in the cycle entry state, and the shutdown sequence starts from the cycle entry state and ends in the initial state of the model while producing one further product of each type. The production plan to produce N products of each type then starts with the initialisation sequence, followed by $N-1$ repetitions of the optimal cycle, and the shutdown sequence that produces the last two products.

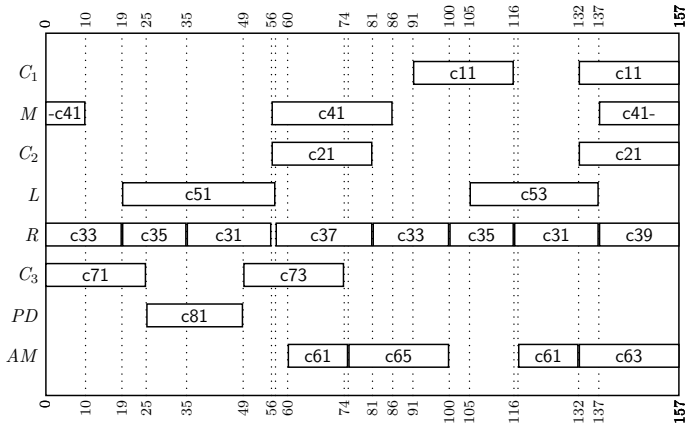

Fig. 7. Optimal cycle.

The initialisation and shutdown sequences computed by *Supremica* have durations of 96 and 142 ticks respectively. Thus, the time to produce N products of each type is

$$T(N) = 96 + 157(N-1) + 142 = 157N + 81 \ .$$

For $N \leq 15$, this value $T(N)$ is exactly the optimal makespan as computed in Table 2. For larger values of N, the linear factor, 157, cannot be improved because the cycle is optimal as explained above. The same cannot be said for the constant, 81, and it is conceivable (but unlikely) that a smaller value can be achieved for some large value of N. Even if such a solution exists, the improvement will be insignificant compared to the linear factor for sufficiently large N. Therefore, the solution proposed here is asymptotically close to optimal.

5. CONCLUSIONS

A discrete event system model of the flexible manufacturing system of de Queiroz et al. (2005) has been devel-

oped, and the model checking tool *Supremica* has been used to compute optimal task schedules. While previously only solved heuristically, *Supremica* can compute optimal schedules to manufacture up to 30 products, and optimal production cycles that solve the problem for an arbitrary number of products.

The modelling approach works for manufacturing systems with operations started by controllable events and finished by uncontrollable events. Breadth-first search is used to compute optimal schedules, which requires that the number of operations needed for a desired product is always the same. Supervisor synthesis ensures robustness of the computed schedules in case of variations in operation times, while optimisation is based on a timed model with fixed operation durations. This suggests that the method works well for similar models, where there are small variations in operation times but no major alternatives for production.

Future work may improve the search for the optimal cycle, which was found by trial and error, and develop algorithms to find cycles automatically from the fixed-size solutions.

REFERENCES

Åkesson, K., Fabian, M., Flordal, H., and Malik, R. (2006). Supremica—an integrated environment for verification, synthesis and simulation of discrete event systems. In *8th Int. Workshop on Discrete Event Systems*, 384–385. IEEE. doi:10.1109/WODES.2006.382401.

Aytug, H., Lawley, M.A., McKay, K., Mohan, S., and Uzsoy, R. (2005). Executing production schedules in the face of uncertainties: A review and some future directions. *Eur. J. Oper. Res.*, 161, 86–110. doi:10.1016/j.ejor.2003.08.027.

Chen, Y.L. and Lin, F. (2000). Modeling of Discrete Event Systems Using Finite State Machines with Parameters. In *2000 IEEE Int. Conf. Control Applications*, 941–946. doi:10.1109/CCA.2000.897591.

Clark, W., Polakov, W.N., and Trabold, F.W. (1922). *The Gantt chart, a working tool of management*. The Ronald Press Company, New York, NY, USA.

Clarke, Jr., E.M., Grumberg, O., and Peled, D.A. (1999). *Model Checking*. MIT Press.

de Queiroz, M.H. and Cury, J.E.R. (2000). Modular supervisory control of large scale discrete event systems. In R. Boel and G. Stremersch (eds.), *Discrete Event Systems: Analysis and Control*, SECS 569, 103–118. Kluwer.

de Queiroz, M.H., Cury, J.E.R., and Wonham, W.M. (2005). Multitasking supervisory control of discrete-event systems. *Discrete Event Dyn. Syst.*, 15, 375–395. doi:10.1007/s10626-005-4058-y.

Gontijo, R.L. (2015). Aplicação da ferramenta de verificação formal UPPAAL na obtenção de planos de produção para sistemas de manufatura modelados como sistemas a eventos discretos. Trabalho de Conclusão de Curso, Universidade Federal de Minas Gerais, MG, Brazil.

Hoare, C.A.R. (1985). *Communicating Sequential Processes*. Prentice-Hall.

Huang, J. and Kumar, R. (2008). Optimal nonblocking directed control of discrete event systems. *IEEE Trans. Autom. Control*, 53(7), 1592–1603. doi:10.1109/TAC.2008.927800.

Malik, R. (2016). Programming a fast explicit conflict checker. In *13th Int. Workshop on Discrete Event Systems*, 464–469. IEEE. doi:10.1109/WODES.2016.7497885.

Malik, R., Åkesson, K., Flordal, H., and Fabian, M. (2017). Supremica—an efficient tool for large-scale discrete event systems. *IFAC PapersOnLine*, 50(1), 5794–5799. doi:10.1016/j.ifacol.2017.08.427.

Malik, R., Fabian, M., and Åkesson, K. (2011). Modelling large-scale discrete-event systems using modules, aliases, and extended finite-state automata. In *18th IFAC World Congress*, 7000–7005. doi:10.3182/20110828-6-IT-1002.00593.

Mohajerani, S., Malik, R., and Fabian, M. (2014). A framework for compositional synthesis of modular nonblocking supervisors. *IEEE Trans. Autom. Control*, 59(1), 150–162. doi:10.1109/TAC.2013.2283109.

Niebert, P. and Yovine, S. (2000). Computing optimal operation schemes for chemical plants in multi-batch mode. In N. Lynch and B.H. Krogh (eds.), *Hybrid Systems: Computation and Control, HSSC 2000*, volume 1790 of *LNCS*, 338–351. Springer. doi:10.1007/3-540-46430-1_29.

Pena, P.N., Costa, T.A., Silva, R.S., and Takahashi, R.H.C. (2016). Control of flexible manufacturing systems under model uncertainty using supervisory control theory and evolutionary computation schedule synthesis. *Inform. Sciences*, 329, 491–502. doi:10.1016/j.ins.2015.08.056.

Pinha, D.C., de Queiroz, M.H., and Cury, J.E.R. (2011). Optimal scheduling of a repair shipyard based on supervisory control theory. In *2011 IEEE Conf. Automation Science and Engineering*, 39–44. doi:10.1109/CASE.2011.6042515.

Ramadge, P.J.G. and Wonham, W.M. (1989). The control of discrete event systems. *Proc. IEEE, Special Issue on Discrete Event Dynamic Systems*, 77, 81–98. doi:10.1109/5.21072.

Saygin, C. and Kilic, S.E. (1999). Integrating flexible process plans with scheduling in flexible manufacturing systems. *Int. J. Adv. Manuf. Technol.*, 15, 268–280. doi:10.1007/s001700050066.

Sköldstam, M., Åkesson, K., and Fabian, M. (2007). Modeling of discrete event systems using finite automata with variables. In *46th IEEE Conf. Decision and Control*, 3387–3392. doi:10.1109/CDC.2007.4434894.

Somenzi, F. (2005). CUDD: CU decision diagram package, release 2.4.1. Technical report, Dept. Electrical and Computer Engineering, Univ. Colorado at Boulder, USA. URL `http://vlsi.colorado.edu/~fabio/CUDD`.

Su, R. and Wonham, W.M. (2004). Supervisor reduction for discrete-event systems. *Discrete Event Dyn. Syst.*, 14, 31–53. doi:10.1023/B:DISC.0000005009.40749.b6.

Ware, S. and 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. doi:10.1109/TASE.2016.2613911.

Wu, N. and Zhou, M. (2012). Schedulability analysis and optimal scheduling of dual-arm cluster tools with residency time constraint and activity time variation. *IEEE Trans. Autom. Sci. Eng.*, 9(1), 203–209. doi:10.1109/TASE.2011.2160452.