TIMED PETRI-NET BASED FORMULATION AND AN ALGORITHM FOR THE OPTIMAL SCHEDULING OF BATCH PLANTS

TIANLONG GU[*], PARISA A. BAHRI[**], GUOYONG CAI[*]

[*] School of Computer Science
Guilin University of Electronic Technology
Guilin 541004, China
e-mail: {cctlgu, ccgycai}@gliet.edu.cn

[**] School of Engineering, Murdoch University
Murdoch, WA 6150, Australia
e-mail: parisa@eng.murdoch.edu.au

The effective scheduling of operations in batch plants has a great potential for high economic returns, in which the formulation and an optimal solution algorithm are the main issues of study. Petri nets have proven to be a promising technique for solving many difficult problems associated with the modelling, formal analysis, design and coordination control of discrete-event systems. One of the major advantages of using a Petri-net model is that the same model can be used for the analysis of behavioural properties and performance evaluation, as well as for the systematic construction of discrete-event simulators and controllers. This paper aims at presenting a Petri-net based approach to the scheduling of operations in batch plants. Firstly, the short term of the 'scheduling of batch plants' is formulated by means of a timed Petri net which can accommodate various intermediate storage policies, such as unlimited intermediate storage (UIS), no intermediate storage (NIS), finite intermediate storage (FIS), and mixed intermediate storage (MIS). Secondly, a heuristic search algorithm for the optimal scheduling of batch plants is given, which is based on generating and checking the markings in the reachability tree of the Petri-net model. Finally, the novel formulation and algorithm are tested with several simulation case studies.

Keywords: timed Petri nets, scheduling, batch plants, discrete event, algorithm, heuristic

1. Introduction

The problem of production scheduling constitutes an important topic in industrial plant operations, especially when multipurpose/multiproduct batch plants are involved (Rippin, 1993). The effective scheduling of operations in batch plants has a great potential for high economic returns, in which the formulation and the optimal solution algorithm are the main issues of study. Traditionally, the optimal scheduling of batch plants has been formulated as Mixed Integer Linear or Nonlinear Programming (MILP or MINLP) problems (Kondili *et al.*, 1996; Papageorgaki and Reklaitis, 1990). The methodologies allow the incorporation of almost any constraint in the problem formulation, but the solution algorithms suffer from high combinatorial complexity. The inclusion of nonlinearities and integer variables creates a particularly difficult problem. To improve the computational efficiency, heuristic search algorithms are often adopted (Błażewicz *et al.*, 1996), which tend towards finding good solutions. A major problem with these algorithms is that they usually end up in local optima. As a remedy, stochastic global optimisation techniques (simulated annealing and genetic algorithms) were proposed (Graells *et al.*, 1996; Kobayashi *et al.*, 1995). These algorithms turn out to be efficient in the optimal problems that are dominated by a combinatorial element. Furthermore, they are not as much affected by nonlinearities and complex objective functions as mathematical programming algorithms. However, it is quite difficult to include complex constraints into their internal representation to ensure the feasibility of the generated schedules. More recently, to easily formulate the scheduling problem, a combinatorial technique using the graph representation was proposed (Adams *et al.*, 1988; Carlier and Pinson, 1988). This technique was then extended to the short term scheduling of multipurpose batch plants (Sanmarti, 1998). However, the main problem with this method is that it can only handle unlimited or non-intermediate storage cases.

Petri nets constitute a promising technique to solve many difficult problems associated with the modelling, formal analysis, and design of discrete-event systems (Murata, 1989). One of the major advantages of using a

Petri net model is that the same model can be used for the analysis of behavioural properties and performance evaluation, as well as for systematic construction of discrete-event simulators and controllers. Petri nets have been used extensively in discrete manufacturing systems (Zhou and Kurapati, 1999; YoungWoo *et al.*, 2001a), through representing simple production lines with buffers, flexible manufacturing systems, intelligent machines and robotics, to implementing supervisory control of their logical behaviours. The timed Petri-net-based optimal scheduling of discrete manufacturing systems was also reported (YoungWoo *et al.*, 2001b; Moro *et al.*, 2002; Lee and Dicesare, 1994). Petri-net based techniques for batch chemical industries were started by Yamalidou and Kantor (1991) and Hanisch (1992), where Petri nets were used for the logic modelling and coordination control of discrete batch operations. Gonnet and Chiotti presented a Petri-net supervisory control technique for multipurpose plants (Gonnet and Chiotti, 1997). Gu and Bahri (2002) reviewed Petri-net techniques in chemical batch plants. However, the Petri-net based optimal scheduling of batch plants has not been given much attention (Gu and Bahri, 1999).

The aim of this paper is to present a timed Petri-net-based novel approach to the optimal scheduling of batch plants. In Section 2, some background on timed Petri nets is given. Section 3 is devoted to the formulation of the short-term scheduling of batch plants. In order to obtain an optimal schedule, a heuristic search algorithm is presented in Section 4. The applicability of the proposed approach is illustrated through some simulation case studies in Section 5.

2. Background for Timed Petri Nets

A Petri net is a directed, weighted, bipartite graph consisting of two kinds of nodes called places and transitions. Graphically, the places are represented by circles and the transitions by bars. An arc connects either a place to a transition, or a transition to a place. Formally, an ordinary Petri net is represented as $\mathrm{PN} = (P, T, R)$, where $P = \{p_1, p_2, \dots, p_n\}$ is a finite set of places; $T = \{t_1, t_2, \dots, t_m\}$ is a finite set of transitions; $R \subseteq (P \times T) \cup (T \times P)$ is a binary relation corresponding to the set of directed arcs from P to T or from T to P (Murata, 1989). Here, P and T are disjoint sets, i.e., $P \cap T = \Phi$. The set of the places (or transitions) connected to a transition t (or the place p) is called the set of input places (or transitions) of t (or p), denoted by $I^t(t)$ (or $I^p(p)$), i.e.,

$$I^t(t) = \{p' \mid (\forall\, p' \in P)\ (p', t) \in R\}, \quad \forall\, t \in T,$$

$$I^p(p) = \{t' \mid (\forall\, t' \in T)\ (t', p) \in R\}, \quad \forall\, p \in P.$$

The set of the places (transitions) connected to a transition t (or a place p) is called the set of output places (or transitions) of t (or p), denoted by $O^t(t)$ (or $O^p(p)$), i.e.,

$$O^t(t) = \{p' \mid (\forall\, p' \in P)\ (t, p') \in R\}, \quad \forall\, t \in T,$$

$$O^p(p) = \{t' \mid (\forall\, t' \in T)\ (p, t') \in R\}, \quad \forall\, p \in P.$$

Furthermore, $I^t : T \to 2^P$ (or $I^p : P \to 2^T$) is defined as a transition (or place) input function, and $O^t : T \to 2^P$ (or $O^p : P \to 2^T$) as a transition (or place) output function. Then an ordinary Petri net can also be formally represented by $\mathrm{PN} = (P, T, I^t, O^t)$ or $\mathrm{PN} = (P, T, I^p, O^p)$.

In addition to the elements described above, a marked Petri net contains tokens. Tokens remain in places, travel along arcs, and their flow through the net is regulated by transition. Tokens are graphically represented by dots. The marking of a place p in ordinary Petri nets is a mapping of the place to a non-negative integer representing the number of tokens residing in this place, denoted by $m(p)$. The markings of all the places in an ordinary Petri net constitute the marking vector $m : P \to \mathbb{N}_+$ (the non-negative integer set), where the i-th component $m(p_i)$ represents the number of the tokens in the i-th place p_i. The initial marking is denoted by m_0. Formally, a marked Petri net is represented as $\mathrm{PN} = (P, T, R, m_0)$, or $\mathrm{PN} = (P, T, I^t, O^t, m_0)$, or $\mathrm{PN} = (P, T, I^p, O^p, m_0)$.

In a marked Petri net, the places are static and the transitions are dynamic. The latter can be triggered in terms of the enabled conditions and firing rules:

Enabled conditions: For some $t \in T$ and marking m_k, if any $p \in I^t(t)$ satisfies $m_k(p) \geq 1$, then t is called the enabled transition under this marking;

Firing rules: The occurrence of a transition t under the marking m_k results in a new marking m_{k+1}:

$$m_{k+1}(p) = \begin{cases} m_k(p), & \forall p \in \big(I^t(t) \cap O^t(t)\big), \\ m_k(p) - 1, & \forall p \in \big(I^t(t) - O^t(t)\big), \\ m_k(p) + 1, & \forall p \in \big(O^t(t) - I^t(t)\big). \end{cases}$$

Obviously, the firing of transition results in a changeable distribution of tokens in places, or changeable markings. Alternative occurrences of transitions produce a firing sequence $\sigma = m_0 t_{i0} m_1 t_{i1} m_2 t_{i2} \dots t_{ik} m_{k+1}$, where the occurrence of transitions t_{i0}, t_{i1}, t_{i2} and t_{ik} has a partial relation $t_{i0} < t_{i1} < t_{i2} < t_{ik}$.

Ordinary Petri nets constitute a powerful tool for modelling and analysing the logical behaviour of complex discrete event systems. However, in many cases time plays an important role in system evolution and cannot be ignored. Time constants have been included into a Petri net by associating them either with transitions (called a timed transition Petri net, TTPN), or with places (called a timed place Petri net, TPPN). In a TTPN, there are two modes of firing. In one case, tokens are removed from the input places when a transition becomes enabled. The transition fires after some period of time (time delay), depositing tokens in the output places. In the other mode, tokens remain in the input places of an enabled transition. After a time delay, the transition fires by removing tokens from the input places, and deposits tokens in the output places. In a TPPN, a token deposited in places becomes available only after some period of time (a time delay). Only the available tokens can enable transitions. A TPPN can be formally represented as the six-tuple:

$$\mathrm{PN} = (P, T, I^t, O^t, m_0, \gamma),$$

or

$$\mathrm{PN} = (P, T, I^p, O^p, m_0, \gamma),$$

where P, T, I^t, O^t, I^p, O^p and m_0 have the same meanings as those in ordinary Petri nets, $\gamma : P \to \mathbb{R}^+$ (the set of nonnegative real numbers) is the time delay function of places.

3. Formulating the Scheduling Problems of Batch Plants

In a batch plant, there are n ($n > 1$) products to be produced in m ($m > 1$) processing units. For each product, the sequence of visiting the processing units is pre-specified, referred to as the product (or job) routing or processing recipes. Normally, the processing time τ_{ij} for product (or job) i ($i = 1, 2, \dots, n$) in unit j ($j = 1, 2, \dots, m$) is usually given.

In a TPPN, places can be used to model activities or resources, and time constants associated with places represent the processing duration of activities. A token residing in places means either the availability of a resource, or an intermediate product being processed. Transitions can be used to model the discrete events of the involved activities, such as opening a valve, starting an operation, finishing a reaction, etc.

Let O_{ij} represent the operation (or the processing activity) of product i at unit j. This operation O_{ij} can be represented by two transitions t_{sij} and t_{fij} for the starting and terminating of this operation, respectively, and one place p_{ij} with time duration τ_{ij} for the processing activity (see Fig. 1(a)). For product i, assuming that O_{ij} is the upward activity of operation O_{ik}, these two connective activities (operations) are related by the intermediate place p_{Iij}, which is introduced to represent the readiness of the intermediate product for the next processing activity (see Fig. 1(b)). All the activities of the same product can be linked by additional intermediate places (see Fig. 1(c)).

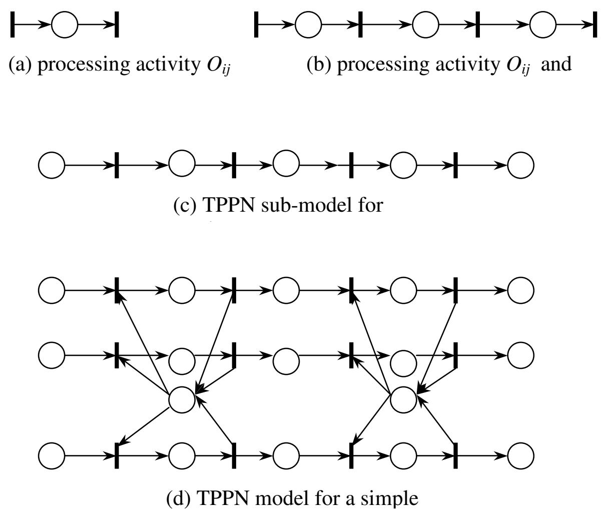

(a) processing activity O_{ij}
(b) processing activity O_{ij} and
(c) TPPN sub-model for
(d) TPPN model for a simple

Fig. 1. Modelling a simple batch plant via TPPNs.

In a batch plant, there exist activities of processing different products in the same unit. This kind of resource sharing can be modelled by introducing a resource place p_j ($j = 1, 2, \dots, m$) for each processing unit. Assuming that operations O_{ij} ($j = 1, 2, \dots, m$) share the same unit j, place p_j is both the input of the starting events and the output of the terminating events for all these activities.

Now all the operations in a batch plant can be easily formulated by the following procedures:

S1: For product i, each processing activity is represented by two transitions t_{sij}, t_{fij}, and one place p_{ij}.

S2: For product i, introduce the i-th initial marked place p_{si} representing the beginning of the job (e.g., the raw materials are ready), and the i-th final place p_{fi}, representing the end of the i-th job.

S3: For the processing activity O_{ij}, introduce the intermediate place p_{Iij}, representing the readiness of the intermediate product for the next processing activity, $O_{i(j+1)}$.

S4: For the processing unit j, introduce the resource place p_j representing the unit's availability.

S5: In terms of job routing or product recipe, all the activities involved in product i are linked, and modelled as a TPPN sub-model.

S6: All the activities that share the same resource places p_j ($j = 1, 2, \dots, m$) are interconnected, and a complete TPPN model for the scheduling of batch plants is created.

Figure 1(d) shows the TPPN for a simple multiproduct batch plant consisting of three products (p_1 to p_3) and two processing units (u_1 to u_2), where the processing times are given in Table 1.

Table 1. Processing times (h) of products.

Units	Products		
	p_1	p_2	p_3
u_1	3.0	4.0	3.0
u_2	4.0	5.0	7.0

So far, the intermediate storage policies have not been included into the modelling procedures. In fact, they can handle only an unlimited intermediate storage policy (UIS) (i.e., buffers with infinite sizes). For the UIS policy, every intermediate product of the i-th job after unit j does not need to occupy unit j until the next unit (unit $j+1$) is ready to process the i-th job. If the next unit (unit $j+1$) is busy, the intermediate product of the i-th job can be transferred from unit j to any available storage at any time. Essentially, the UIS does not impose any constraints on the operations in a batch plant. However, there also exist other complicated intermediate storage policies, such as no intermediate storage (NIS), finite intermediate storage (FIS), zero wait (ZW), and mixed intermediate storage (MIS) (Ku *et al.*, 1987; Ku and Karimi, 1990). In the following, we shall discuss how to deal with these intermediate storage policies.

Finite Intermediate Storage (FIS) Policy. Under the FIS policy, finite storage tanks (i.e., buffers with finite sizes) are available between batch units. If there is at least one intermediate storage tank available between units j and $j+1$, the intermediate product of the i-th job finished at unit j does not need to occupy this unit until the next unit (unit $j+1$) is ready to implement product i. If there are b_j intermediate storage tanks between unit j and unit $j+1$, an intermediate storage place $p_{sj(j+1)}$ can be introduced, which contains initially s_j tokens. The intermediate storage place $p_{sj(j+1)}$ will function as both the input of the finishing events of upward activities and the output of starting events of downward activities. In order to formulate the FIS policy, Steps 4 and 6 in the previous procedures should be modified:

S4': For the processing unit j, introduce a processing unit resource place p_j ($j = 1, 2, \dots, m$), and an intermediate storage resource place $p_{sj(j+1)}$ ($j < m$) initially marked by s_j tokens, where b_j is the number of intermediate storage tanks between units j and $j+1$.

S6': All the activities that share the same resource places p_j ($j = 1, 2, \dots, m$) (processing unit) are interconnected, and so are all the activities that share the same sharing resource places $p_{sj(j+1)}$ ($j < m$) (intermediate storage tank), which results in a complete TPPN for the scheduling of batch plants.

No Intermediate Storage (NIS) Policy. Under the NIS policy, there exists no intermediate storage in the batch plant. Then the intermediate product of the i-th job finished at unit j has to be held in unit j until the next unit (unit $j+1$) is ready to implement the i-th job. This imposes additional restrictions on the operation, which can be represented by introducing a virtual resource place $p_{dj(j+1)}$ between units j and $j+1$. This virtual resource place $p_{dj(j+1)}$ is initially marked by one token, and serves as both the input of the finishing events of upward activities and the output of the finishing events of downward activities. In order to formulate the NIS policy, Steps 4 and 6 in the previous procedures should also be modified:

S4: For the processing unit j, introduce a processing unit resource place p_j ($j = 1, 2, \dots, m$), and an intermediate storage imaginary resource place $p_{dj(j+1)}$ ($j < m$), initially marked by one token.

S6: All the activities that share the same resource places p_j ($j = 1, 2, \dots, m$) (processing unit) are interconnected, and so are all the activities that share the same virtual resource places $p_{dj(j+1)}$, which results in a complete TPPN for the scheduling of batch plants.

Mixed Intermediate Storage (MIS) Policy. Under the MIS policy, each of UIS, FIS and NIS may be used between units j and $j+1$. This only brings some difference of either intermediate storage resource places or virtual resource places. Thus, the modelling procedures can be modified as follows:

S4: For the processing unit j, introduce an processing unit resource place p_j ($j = 1, 2, \dots, m$), an intermediate storage resource place $p_{sj(j+1)}$ ($j < m$), marked initially by b_j tokens, if there are b_j intermediate storage tanks between units j and $j+1$, and a virtual intermediate storage resource place $p_{dj(j+1)}$ ($j < m$), initially marked by one token if there is no intermediate storage between units j and $j+1$.

S6: All the activities that share the same resource places p_j ($j = 1, 2, \dots, m$) (processing unit) are interconnected, and so do all the activities that share the same virtual resource places $p_{dj(j+1)}$ and the intermediate storage resource places $p_{sj(j+1)}$. Then a complete TPPN for the scheduling of batch plants is created.

4. Heuristic Algorithm

In the TPPN formulation of a batch plant, all the possible operations of different sequences can be completely tracked by their reachability tree. Theoretically, an optimal schedule of the batch plant can be obtained by generating the reachability tree, and finding an optimal path from the initial to the final marking. The path is a firing sequence of the transitions of the TPPN model. However, even for a simple Petri net, the reachability tree may be too large to generate in its entirety. Instead of generating the entire reachablity tree, a heuristic algorithm is developed to generate only a necessary portion of the reachability tree for the optimal searches.

The heuristic algorithm that finds an optimal path is constructed based on the well-known graph search algorithm A^* (Nilsson, 1980). Given a TPPN model, the algorithm expands the reachability tree from the initial marking until the generated portion of the reachability tree touches the final marking. Once the final marking is found, the optimal path is constructed by tracing the pointers that denote the parenthood of the markings, from the final to the initial marking. Then the transition sequence of the path provides the order of the activities or operations, i.e., the schedule. In the algorithm, *expanding* and *checking* are two important aspects:

(1) *Expanding*: For the current node or marking m_k, find all possible enabled transitions and compute their new marking m_{k+1} in terms of the enabled conditions and firing rules. These new markings $\{m_{k+1}\}$ are the descendant or new nodes expanded from the current node m_k, and stored in the OPEN list. Here, $\{m_{k+1}\}$ is used to represent the set of all possible new markings, and the list OPEN maintains all the markings generated but not explored yet. The current node m_k is put in the list CLOSED, which maintains all the markings generated and explored so far. In this expanding process, some important information is recorded, such as the time during which the new marking is produced, denoted by $g(m_{k+1})$, the time during each processing unit has been utilised, denoted by $\mathrm{UT}_j(m_{k+1})$ ($j = 1, 2, \ldots, m$), the time during each product has been processed, denoted by $\mathrm{PT}_i(m_{k+1})$ ($i = 1, 2, \ldots, n$), and the return pointer of $\{m_{k+1}\}$ as the current node m_k.

(2) *Checking*: When the expanding reaches the final marking, we say that a feasible schedule is obtained, and then determine whether or not the current upper bound of the objective function should be updated. If the current node m_k is not a final marking, determine whether a branch should be expanded (forward search) or discarded (backward search). The checking is based on the estimated cost $f(m_k)$ and the upper bound UB. The upper bound (UB) of the cost can be initialised as a large number M so as to ensure that a reasonably optimal makespan is not left out.

Under the assumption that the optimal scheduling is to satisfy the minimal makespan, the estimated cost can be calculated as follows:

$$f(m_k) = h(m_k) + g(m_k),$$
$$h(m_k) = \max\Big(\max_j\Big\{\sum_i(\tau_{ij}) - \mathrm{UT}_j(m_k)\Big\}, \max_i\Big\{\sum_j(\tau_{ij}) - \mathrm{PT}_i(m_k)\Big\}\Big),$$

where $(\sum_i(\tau_{ij}) - \mathrm{UT}_j(m_k))$ is the total remaining duration of the products which are assumed to be processed consecutively in unit j, and $(\sum_j(\tau_{ij}) - \mathrm{PT}_i(m_k))$ is the total remaining duration of the batch units which are assumed to be utilised consecutively for product i; $f(m_k)$ is an estimate of the cost, i.e., the makespan from the initial to the final marking along an optimal path which goes through the current marking m_k; $g(m_k)$ is the current lowest cost obtained from the initial to the current marking m_k; $h(m_k)$ is the heuristic function, or an estimate of the cost from the current marking m_k to the final marking along an optimal path which goes through the current marking m_k. The estimated cost $f(m_k)$ always gives the lowest bound of the makespan since any idle times that may occur afterwards are not included in $h(m_k)$, i.e., it satisfies the following condition:

$$0 \leq h(m_k) \leq h^*(m_k).$$

Therefore the scheduling algorithm is admissible (Nilsson, 1980), i.e., it always finds an optimal path or an optimal schedule.

The heuristic algorithm for the short-term scheduling of batch plants is implemented in the following steps:

s1: Initialize the upper bound (UB) of the makespan as a large number M, and $e(m_0)$, $\mathrm{UT}_j(m_0)$ ($j = 1, 2, \ldots, m$) and $\mathrm{PT}_i(m_0)$ ($i = 1, 2, \ldots, n$) equal to 0. Store the initial marking m_0 in the OPEN list.

s2: If OPEN is empty, the search is complete, and the firing sequence and the time constants of the optimal schedule are found, then Stop.

s3: Retrieve marking m_k from OPEN, and set m_k as the current node (marking).

s4: If m_k is not the final marking, go to s6.

s5: If $g(m_k) <$ UB, update $\mathrm{UB} = g(m_k)$. Go to s2.

s6: For the current marking m_k, compute its estimated cost $f(m_k) = h(m_k) + g(m_k)$.

s7: If $f(m_k) \geq$ UB, go to s2.

s8: For the current marking m_k, find all the possible enabled transitions, and compute the new marking $\{m_{k+1}\}$. Put $\{m_{k+1}\}$ into the OPEN list and m_k into the CLOSED list.

s9: Reset the return pointer of m_{k+1}. Update $g(m_{k+1})$, $\mathrm{UT}_j(m_{k+1})$ $(j = 1, 2, \dots, m)$ and $\mathrm{PT}_i(m_{k+1})$ $(i = 1, 2, \dots, n)$. Go to s2.

The simple batch plant in Fig. 1 is revisited to illustrate the heuristic algorithm. In the TPPN model, the initial marking is $m_0 = (1\ 1\ 1\ 1\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0)$ and the final marking is set as $m_f = (0\ 0\ 0\ 1\ 1\ 1\ 1\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0)$. The search path of a partial reachability tree is shown in Fig. 2, and the corresponding decision information and enabled transitions are presented in Table 2 and 3. When the search path reaches the marking m_{19}, the upper bound of the makespan is updated, and UB is reset as 19. Under the marking m_{21}, we have $h(m_{21}) = 19 \geq \mathrm{UB}$, hence we search backward (B). Since there is no entry in OPEN, the search is finished successfully, and the optimal schedules are obtained (cf. Fig. 3).

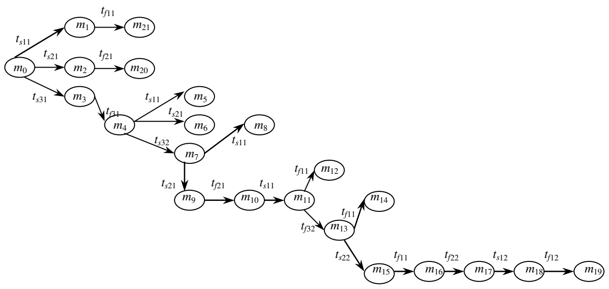


Fig. 2. Optimal searches of the reachability tree.

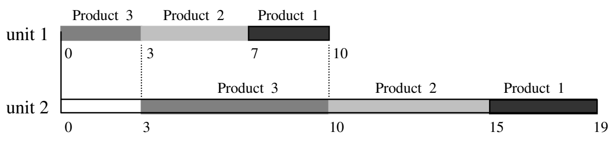


Fig. 3. Gantt chart for optimal schedules.

5. Case Studies

The heuristic algorithm was implemented in Matlab on a PC/Pentium 4, and several case studies were simulated to test it. The results show that the algorithm works well.

Case Study 1. The formulation and the algorithm are tested by considering a multiproduct batch plant consisting of four products (p_1 to p_4) and three processing units (u_1 to u_3). The processing times of each product in each unit are given in Table 4. Different types of intermediate storage policies, such as UIS, NIS, FIS (two buffers between the units 1 and 2, and one buffer between the units 2 and 3) and MIS (two 2 buffers between the units 1 and 2, and no buffer between the units 2 and 3) are assumed. The TPPN models for each intermediate storage policy are shown in Fig. 4. By the heuristic algorithm, optimal schedules are solved, and the results are presented in Tab. 5 and Fig. 5.

Case Study 2. A multi-purpose batch plant with three products (p_1 to p_3) and three processing units (u_1 to u_3) is considered as the second case study, whose product recipes are given in Table 6. In this batch plant, the MIS (Mixed Intermediate Storage) policy with $b_1 = 1$, $b_2 = 0$ and $b_3 = \infty$ is adopted. The TPPN model is shown in Fig. 6. The simulation results from the heuristic algorithm are presented in Fig. 7.

6. Conclusions

A Timed-Placed Petri-Net (TPPN) formulation for the scheduling of batch plants has been proposed. It was shown that the changes in the markings in a TPPN model completely describe the evolution of different operations, even in the presence of complicated intermediate storage policies. An optimal schedule can be obtained by searching the reachability tree of the TPPN model. The heuristic algorithm proposed is admissible, and can always find the optimal schedule. The scheduling of several simple batch plants was simulated to show the applicability of the proposed techniques.

The great benefit of the Petri-net based approach is to graphically and concisely represent activities, resources and constraints of a batch plant in a single coherent formulation, which is very helpful for the designers to better understand and formulate the scheduling problems. There also exist two main aspects of the formulation and algorithms for this novel approach. Improving the computational efficiency of the algorithm for optimal scheduling and giving performance comparisons of TPPN based algorithms with traditional MILP and CLP are under way. Moreover, it is understood from this research that the Petri-net based approach has a great potential for solving a variety of complicated scheduling problems in batch plants. In this regard, further research is also being undertaken to accommodate complicated constraints such as a variable batch size, a variable cycle time and mixed batch/continuous plants, and to implement the integrated design of the supervisory control and scheduling of batch plants.

Acknowledgments

The authors are very grateful to the anonymous reviewers for their helpful comments. This work was supported by the National Natural Science Foundation of China (grant no. 60243002).

Table 2. Marking vectors and enabled transitions.

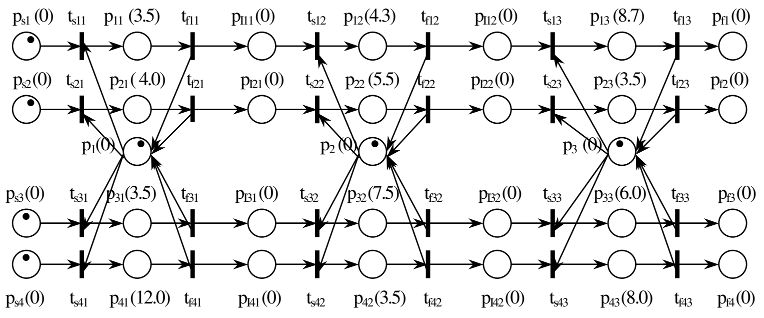
Transition enabled	$e(m_k)$	Marking $m_k(p)$																	m_k
		p_{s1}	p_{s2}	p_{s3}	p_1	p_2	p_{f1}	p_{f2}	p_{f3}	p_{11}	p_{I11}	p_{12}	p_{21}	p_{I21}	p_{22}	p_{31}	p_{I31}	p_{32}	
/	/	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	m_0
t_{s11}	0.0	0	1	1	0	1	0	0	0	1	0	0	0	0	0	0	0	0	m_1
t_{s21}	0.0	1	0	1	0	1	0	0	0	0	0	0	1	0	0	0	0	0	m_2
t_{s31}	0.0	1	1	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	m_3
t_{f31}	3.0	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	m_4
t_{s11}	3.0	0	1	0	0	1	0	0	0	1	0	0	0	0	0	0	1	0	m_5
t_{s21}	3.0	1	0	0	0	1	0	0	0	0	0	0	1	0	0	0	1	0	m_6
t_{s32}	3.0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	m_7
t_{s11}	3.0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	m_8
t_{s21}	3.0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	m_9
t_{f21}	7.0	1	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	1	m_{10}
t_{s11}	7.0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	1	m_{11}
t_{f11}	10.0	0	0	0	1	0	0	0	0	0	1	0	0	1	0	0	0	1	m_{12}
t_{f32}	10.0	0	0	0	0	1	0	0	1	1	0	0	0	1	0	0	0	0	m_{13}
t_{f11}	10.0	0	0	0	1	1	0	0	1	0	1	0	0	1	0	0	0	0	m_{14}
t_{s22}	10.0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	0	0	0	m_{15}
t_{f11}	10.0	0	0	0	1	0	0	0	1	0	1	0	0	0	1	0	0	0	m_{16}
t_{f22}	15.0	0	0	0	1	1	0	1	1	0	1	0	0	0	0	0	0	0	m_{17}
t_{s12}	15.0	0	0	0	1	0	0	1	1	0	0	1	0	0	0	0	0	0	m_{18}
t_{f12}	19.0	0	0	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0	m_{19}
t_{f21}	4.0	1	0	1	1	1	0	0	0	0	0	0	0	1	0	0	0	0	m_{20}
t_{f11}	3.0	0	1	1	1	1	0	0	0	0	1	0	0	0	0	0	0	0	m_{21}

Table 3. Estimated makespan and decisions.

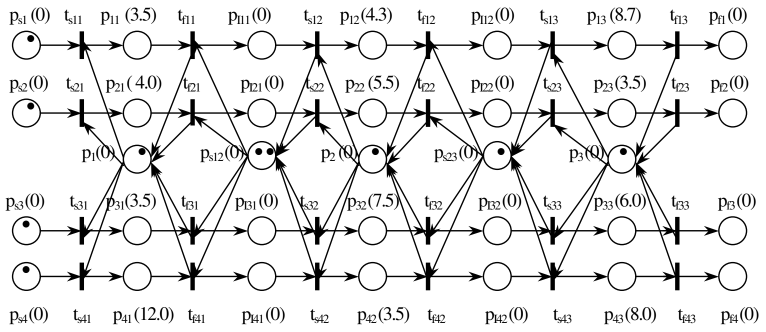
m_k	m_0	m_3	m_4	m_7	m_9	m_{10}	m_{11}	m_{13}	m_{15}	m_{16}	m_{17}	m_{18}	m_{19}	m_{14}	m_{12}	m_8	m_6	m_5	m_2	m_{20}	m_1	m_{21}
$e(m_k)$	0	0	3	3	3	7	7	10	10	10	15	15	19	10	10	3	3	3	0	4	0	3
$g(m_k)$	16	16	16	16	16	12	12	9	9	9	4	4	0	9	9	16	16	16	16	16	16	16
$h(m_k)$	16	16	19	19	19	19	19	19	19	19	19	19	19	19	19	19	19	19	16	20	16	19
UB	M	M	M	M	M	M	M	M	M	M	M	M	M	19	19	19	19	19	19	19	19	19
B/F	F	F	F	F	F	F	F	F	F	F	F	F	B	B	B	B	B	B	F	B	F	/

Table 4. Processing times (h) of products.

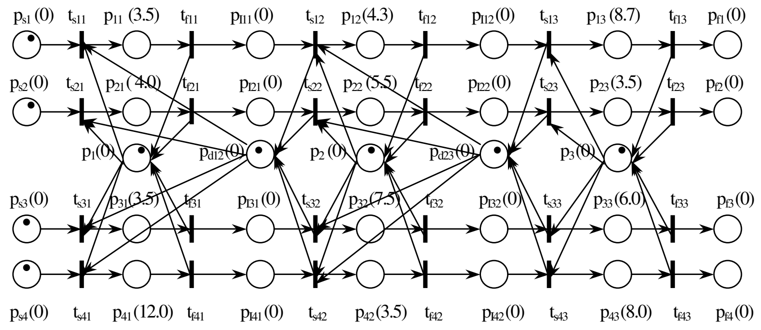
Units	Products			
	p_1	p_2	p_3	p_4
u_1	3.5	4.0	3.5	12.0
u_2	4.3	5.5	7.5	3.5
u_3	8.7	3.5	6.0	8.0



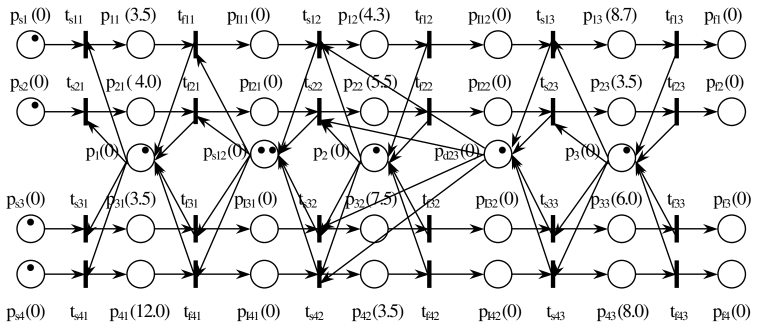
(a) TPPN model under UIS policy



(b) TPPN model under FIS policy



(c) TPPN model under the NIS policy

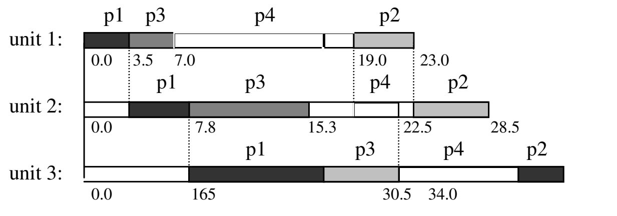


(d) TPPN model under the MIS policy

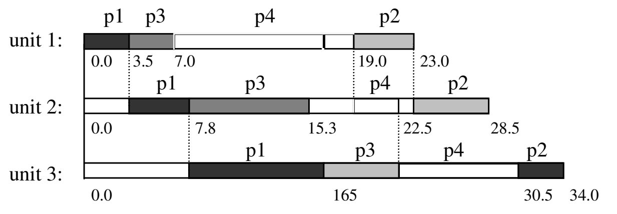
Fig. 4. Modelling Case Study 1 via TPPNs.

Table 5. Simulation results of Case Study 1.

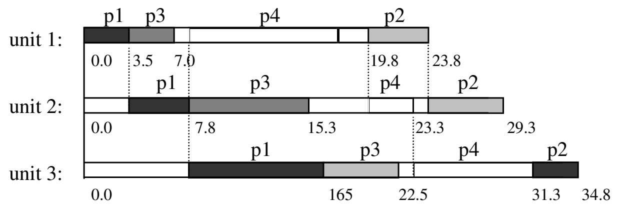
Storage policies	Optimal schedule	Makespan
UIS	product 1 → product 3 → product 4 → product 2	34.0
FIS	product 1 → product 3 → product 4 → product 2	34.0
NIS	product 1 → product 3 → product 4 → product 2	34.8
MIS	product 1 → product 3 → product 4 → product 2	34.0



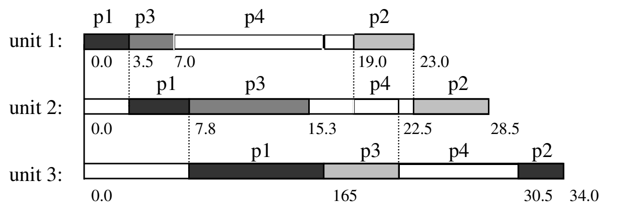
(a) UIS policy



(b) FIS policy



(c) NIS policy



(d) MIS policy

Fig. 5. Gantt chart of optimal schedules.

Tab. 6. Product recipes.

Product 1		Product 2		Product 3	
Unit	Time	Unit	Time	Unit	Time
U1	6	U2	9	U1	14
U3	9	U3	15	U2	16

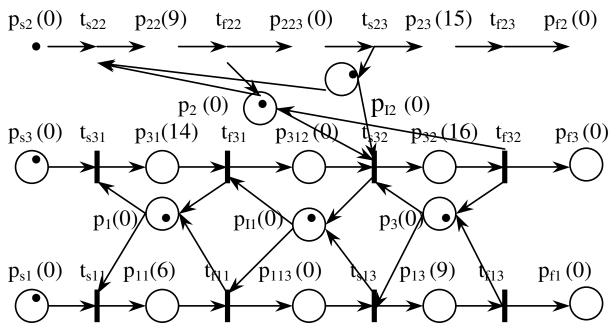


Fig. 6. TPPNs model for Case Study 2.

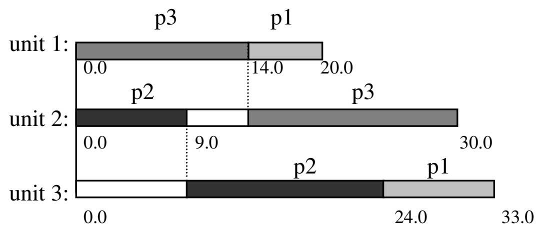


Fig. 7. Gantt chart of optimal schedules for Case Study 2.

References

Adams J., Balas E. and Zawack D. (1988): *The shifting bottleneck procedure for job shop scheduling*. — Manag. Sci., Vol. 34, No. 3, pp. 391–410.

Błażewicz J., Ecker L., Schmidt G. and Węglarz J. (1996): *Scheduling Computer and Manufacturing Processes*. — Berlin: Springer.

Carlier J. and Pinson E. (1988): *An algorithm for solving the job shop problem*. — Manag. Sci., Vol. 35, No. 2 , pp. 164–176.

Gonnet S. and Chiotti O. (1997): *Modelling of the supervisory control system of a multipurpose batch plant*. — Comp. Chem. Eng., Vol. 21S, pp. S691–S696.

Graells M., Espuna A. and Puigjaner L. (1996): *Sequencing intermediate products: A practical solution for multipurpose production scheduling*. — Comp. Chem. Eng., Vol. 20S, pp. S1137–S1142.

Gu T. and Bahri P.A. (1999): *Timed Petri-net representation for short term scheduling of multiproduct batch plants*. — Proc. Amer. Contr. Conf., San Diego, USA, pp. 4092–4096.

Gu T. and Bahri P.A. (2002): *A survey of Petri-net applications in batch processes*. — Comp. Ind., Vol. 47, No. 1, pp. 99–111.

Hanisch H.M. (1992): *Coordination control modelling in batch production systems by means of Petri nets*. — Comp. Chem. Eng., Vol. 16, No. 1, pp. 1–10.

Jung T.H., Kim M. and Lee I. (1996): *Optimal scheduling of multi-product batch processes for various intermediate storage policies*. — Ind. Eng. Chem. Res., Vol. 35, No. 11, pp. 4058–4066.

Kobayashi S., Ono I. and Yamamura M. (1995): *An efficient genetic algorithm for job shop scheduling problems*. — Proc. 6-th Int. Conf. *Genetic Algorithms*, Tokyo, Japan, pp. 506–511.

Ku H.M., Rajagopalan D. and Karimi I.A. (1987): *Scheduling in batch processes*. — Chem. Eng. Prog., Vol. 83, No. 8, pp. 35–45.

Ku H.M. and Karimi I.A. (1990): *Completion time algorithms for serial multi-product batch processes with shared storage*. — Comp. Chem. Eng., Vol. 14, No. 1, pp. 49–56.

Lee D.Y. and Dicesare F. (1994): *Scheduling flexible manufacturing systems using Petri nets and heuristic search*. — IEEE Trans. Robot. Automat., Vol. 10, No. 2, pp. 123–132.

Moro A.R., Yu H. and Kelleher G. (2002): *Hybrid heuristic search for the scheduling of flexible manufacturing systems using Petri nets*. — IEEE Trans. *Robotics and Automation*, Vol. 18, No. 2, pp. 240–245.

Murata T. (1989): *Petri nets: Properties, analysis and applications*. — Proc. IEEE, Vol. 77, No. 4, pp. 541–580.

Nilsson N. (1980): *Principles of artificial intelligence*. — Palo Alto, CA: Tioga.

Papageorgaki S. and Reklaitis G.V. (1990): *Optimal design of multi-purpose batch plants, I: Problem formulation*. — Ind. Eng. Chem. Res., Vol. 29, No. 5, pp. 2054–2062.

Rippin D.W.T. (1993): *Batch process systems engineering: A retrospective and prospective review*. — Comp. Chem. Eng., Vol. 17S, pp. s1–s13.

Sanmarti E., Friedler F. and Puigjaner L. (1998): *Combinatorial technique for short-term scheduling of multi-purpose batch plants based on schedule-graph representation*. — Comp. Chem. Eng., Vol. 22S, pp. S847–S850.

Yamalidou E.C. and Kantor J.C. (1991): *Modelling and optimal control of discrete-event chemical processes using Petri nets*. — Comp. Chem. Eng., Vol. 15, No. 7, pp. 503–519.

YoungWoo K., Inaba A., Suzuki T. and Okuma S. (2001a): *FMS scheduling based on Petri net model*. — Proc. IEEE Int. Symp. *Assembly and Task Planning*, Fukuoka, Japan, pp. 238–243.

YoungWoo K., Inaba A., Suzuki T. and Okuma S. (2001b): *FMS scheduling based on timed Petri net model and RTA^* algorithm*. — Proc. IEEE Int. Symp. *Assembly and Task Planning*, pp. 848–853.

Zhou M.C. and Kurapati V. (1999): *Modeling, Simulation, and Control of Flexible Manufacturing Systems: A Petri Net Approach*. — New Jersey: World Scientific.

Received: 5 March 2003
Revised: 5 August 2003