Ann Oper Res (2008) 164: 63–78
DOI 10.1007/s10479-007-0258-1

A Petri Net based algorithm for minimizing total tardiness in flexible manufacturing systems

Gonzalo Mejía · Carlos Montoya

Published online: 14 November 2007

Abstract Petri Nets have been extensively used for modeling and simulating of the dynamics of flexible manufacturing systems. Petri Nets can capture features such as parallel machines, alternative routings, batch sizes, multiplicity of resources, to name but a few. However, Petri Nets have not been very popular for scheduling in manufacturing due to the Petri Net "state explosion" combined with the NP-hard nature of many of such problems. A promising approach for scheduling consists of generating only portions of the Petri Net state space with heuristic search methods. Thus far, most of this scheduling work with Petri Nets has been oriented to minimize makespan. The problem of minimizing total tardiness and other due date-related criteria has received little attention. In this paper, we extend the Beam A* Search algorithm presented in a previous work with capability to handle the total tardiness criterion. Computational tests were conducted on Petri Net models of both flexible job shop and flexible manufacturing systems. The results suggest that the Petri Net approach is also valid to minimize due date related criteria in flexible systems.

Keywords Petri Nets · Heuristic search · Beam search · Scheduling · Flexible manufacturing systems

Introduction

Petri Nets have been extensively used for modeling manufacturing systems due to their capability to represent the complex features of such asynchronous, concurrent, discrete-event dynamic systems (Moore and Gupta 1996). Petri Nets, however, suffer from "state explosion" which has prevented their application for scheduling problems in the manufacturing systems field. Some attempts have been made to reduce the enormous effort to search the state space of the Petri Nets. Perhaps the most widely used approach

G. Mejía (✉) · C. Montoya
Department of Industrial Engineering, Universidad de los Andes, PYLO Research Group, Bogotá, Colombia
e-mail: gmejia@uniandes.edu.co

has been the application of modified versions of the widely known A* search algorithm. The A* search algorithm is a branch and bound-like algorithm which expands only the most promising nodes of the Petri Net reachability graph (the graph that contains all states of the net). The first papers date from 1994 (Lee and DiCesare 1994; Sun et al. 1994). Since then, other authors have presented improved versions of the A* search algorithm. For example, Sun et al. (1994), limited the expansion of the net reachability graph by selectively pruning non-promising branches; Xiong and Zhou (1998) implemented the A* search with limited back-tracking capability; Jeng and Chen (1998) used the Petri Net state equations to calculate lower bounds for completion times; Reyes-Moro et al. (2002) presented a staged search approach combined with a pruning strategy to reduce the search space; Mejía (2003) and Mejía and Odrey (2005) introduced an algorithm which combines the A* and Beam Search algorithms for Petri Nets.

These algorithms have focused on minimizing makespan. Problems related to tardiness have received very little attention from the Petri Net approach. In this paper, we present a version of the Beam A* Search (BAS) (Mejía and Odrey 2005) that incorporates features to handle due dates. The BAS algorithm presented here also includes features as selective pruning (within a beam), the non-delay branching scheme and a bounding scheme based on dispatching rules. In this paper, we focus on the total tardiness criterion.

Definition

$$\text{Total Tardiness} = \sum_{j=1}^{n} T_j,$$

where T_i is the tardiness of the jth job. T_j is defined as $\max(C_j - d_j, 0)$ where C_j and d_j are respectively the completion time and the due date of the jth job.

Incorporating due dates implies several difficulties that will be explained later in this document. These relate basically to the fact that the total tardiness can only be determined when a full schedule is calculated. Thus lower bounds are difficult to obtain for partial schedules. The later difficulty is associated with the constructive nature of the search algorithms in which the search is directed along partial schedules.

This paper is organized as follows: First, concepts of modeling with Petri Nets are introduced. Section 1 presents the modeling strategy. Time-space state equations to track the net evolution are introduced in Sect. 2. The BAS algorithm for total tardiness is described in detail in Sect. 3. The results of the experiments are presented in Sect. 4. Conclusions and further research are discussed in the last section.

1 Modeling with Petri Nets

A Petri Net is a bipartite directed graph having two kinds of nodes (places and transitions), along with an initial state called the initial marking. Arcs are always directed from a transition to a place and vice versa. In general, places represent actions or conditions and transitions represent events. Entities called tokens reside in places and represent the truth of the conditions or actions associated with the corresponding places. Tokens move throughout the net by the effect of the transition firings. When a transition fires, it removes one token from

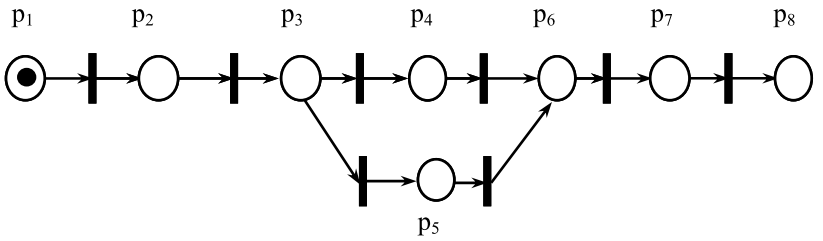

Fig. 1 An example of a Petri subnet for the routing of a job

each of its input places and puts one token on each of its output places.[1] In order to fire, a transition must be enabled. This means that there must be sufficient tokens at the transition input places (i.e. all pre-conditions must be met). See Murata (1989) for details.

In a Timed Petri Net, time can be associated with either places or transitions. In this paper, time is associated with places. In this particular case, a token must wait in a place at least a time elapse corresponding to the time associated with such a place, before moving onto the next place.

Definition Timed Petri Net (TPN): A TPN is a 6-tuple (P, T, I, O, M_0, τ) where $P =$ set of n places, $T =$ set of m transitions, $P \cap T = \{\}$, $I =$ a set of input arcs $(P \times T)$, $O =$ a set of output arcs $(T \times P)$, $M_0 =$ initial marking, and $\tau =$ set of time delays associated with places.

In this paper, we assume that all jobs are available at the beginning and no new jobs arrive to the system. For these assumptions, we follow the modeling methodology presented in previous work (Mejía 2003). First, we model the route of each job with a Petri subnet. Specifically, each place represents either an operation (e.g. job being processed by machine) or a condition (e.g. job ready). Transitions in all cases represent the beginning or the end of an operation.

Table 1 Example of Job routing

Operation	1	2	3
Resources	M1	M2 or M3	M4

The Petri subnet for this job is represented as shown in Fig. 1. Table 2 shows the description for each place. This example was taken from Mejía and Odrey (2005). Notice that there is one unique source place (no input transitions) and one unique sink place (no output transitions). These represent respectively the conditions "job ready" and "job finished". Transitions represent the beginning and the end of each operation.

The same modeling method is followed for all jobs allocated to the system. When all jobs are modeled, the next step is the addition of resources. This implies the addition of "resource places" which represent the availability of a resource, and arcs from/to the resource places to/from the transitions of the nets representing the job routings. The number of resource

[1] This is the definition of Ordinary Petri Nets. In the general case, the number of tokens removed/placed depends on the arc weights.

Table 2 Description of places for the net in Fig. 1

p_1	Job ready	p_5	Alternative 2 for the second operation of a Job being processed by machine M3
p_2	First operation of a Job being processed by machine M1	p_6	Operation 2 finished
p_3	Operation 1 finished	p_7	Last operation of a Job being processed by machine M3
p_4	Alternative 1 for the second operation of a Job being processed by machine M2	p_8	Job finished

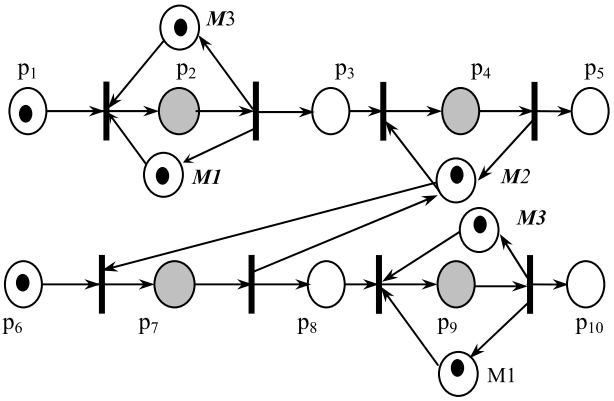

Fig. 2 Petri Net model of a system with two jobs and three machines

places corresponds in this case to the total number of machines. New transitions are not added. The resource allocation occurs when a transition, having one or more resource places as input places, fires. Resources become unavailable when allocated. Similarly, resources are released and become available when a transition, having one or more resource places as output places, fires.

The construction of the net must ensure that (i) a resource cannot be allocated twice before being released, and (ii) that for any outgoing arc from a resource place, there must be an incoming arc to the resource place, as well. A similar methodology for modeling with Petri Nets was proposed in Lee and DiCesare (1994), Mejía and Odrey (2005). Figure 2 illustrates a full Petri Net model of a manufacturing system having two jobs to be processed and three machines (M1, M2 and M3). Job 1 is first processed simultaneously by machines 1 (M1) and 3 (M3) and then by machine 2 (M2). Job 2 is processed first by machine 2 (M2) and then by both machines 1 (M1) and 3 (M3). Highlighted places (p_2, p_4, p_7 and p_9) represent timed places and correspond to the physical operations.

2 State equations

The state equations presented are intended to track both the token evolution and timing of a Petri Net constructed with the modeling methods presented in Sect. 1. These equations will be used later for optimization purposes. The main feature is the augmentation of the marking vector with the remaining processing time vector. Consider a Timed Petri Net having n places and m transitions. The remaining time vector contains in its ith position the remaining

Table 3 Description of places for the net Fig. 2

Place description for Petri Net of Fig. 2			
p_1	Job 1 ready	p_6	Job 2 ready
p_2	Job 1 being processed by machines 1 and 3	p_7	Job 2 being processed by machine 2
p_3	Operation 1 of Job 1 finished	p_8	Operation 1 of Job 2 finished
p_4	Job 1 being processed by machine 2	p_9	Job 2 being processed by machines 1 and 3
p_5	Operation 2 of Job 1 finished	p_{10}	Operation 2 of Job 2 finished
M1	Machine 1 available	M2	Machine 2 available
M3	Machine 3 available		

time required to enable the output transition(s) of the ith place ($i = 1$ to n). The augmented state space representation can be written as:

$$X(k+1) = A(k)X(k) + B(k)u(k). \tag{1}$$

Where: $X(k)$ is the state vector

$$X(k) = \begin{vmatrix} M(k) \\ M_r(k) \end{vmatrix}.$$

$M(k)$ and $M_r(k)$ are $n \times 1$ vectors. $M(k)$ and $M_r(k)$ are respectively the marking vector and the remaining processing time vector after k transition firings (events).

$A(k)$ is the system matrix. This matrix is partitioned as follows:

$$A(k) = \begin{vmatrix} I_n & 0_n \\ -I_n\delta(k) & I_n \end{vmatrix}.$$

Where:

I_n: Identity $n \times n$ matrix.
0_n: Zero $n \times n$ matrix.
$\delta(k)$: Time elapse between two consecutive transition firings. $\delta(k) = \text{Max}(M_{ri}(k))\ \forall i$ such that place p_i is input to the transition firing at the kth event.
$u(k)$: $m \times 1$ Control vector that determines which transition fires after k firings. m is the number of transitions in the net. $u_j(k)$ is the jth position of u at time k. $u_j(k) = 1$ if transition j fires, 0 otherwise.
$B(k)$: Distribution matrix that transforms the control action $u(k)$ into addition or removal of tokens when firing a transition represented in vector $u(k)$.

$$B(k) = \begin{vmatrix} C \\ TC^+ \end{vmatrix}.$$

$C : n \times m$ Incidence matrix. $C = C^+ - C^-$.
C^+ and C^- are, respectively, the incidence input and output matrices. See the definition of incidence matrix in Murata (1989).
$T = \{t_{ij}\}$: Processing time $n \times n$ diagonal matrix for operational places $t_{ii} = \{\tau_i\}$. Where τ_i is the value of the ith position of the vector of time delays associated with places as defined in Sect. 1.

Intuitively, the remaining process time is calculated as follows: After k events we denote the remaining process time vector just before a transition firing as $M_r(k)$. Suppose that jth transition fires at the kth event. The jth transition will fire after $\delta(k)$ units. Thus, the remaining process time vector is recalculated by subtracting $\delta(k)$ time units from the vector $M_r(k)$ (for the places that contain tokens). We denote this temporary remaining process time vector as $M'_r(k)$. In the state space equations this is represented by $M'_r(k) = M_r(k) - I_n\delta(k)M(k)$. Negative values of the $M'_r(k)$ vector are set to 0 (i.e. the remaining times have been exhausted). Then the corresponding tokens are moved from the input places to the jth transition output places. The remaining process time for the incoming tokens is the time delay of such output places since the corresponding actions are ready to start but have not started yet. The time delay of the incoming tokens is represented by the vector $TC^+u(k)$. Thus, the remaining process time vector $M_r(k)$ is updated with the addition of the process time of the just-arrived tokens. This yields $M_r(k+1) = M'_r(k) + TC^+u(k)$. Here, we constrain the timed places to accept at most one token to avoid duplicated remaining process times in one single place. This could happen if more than one token is present in a single place. This constraint is easily met by initially putting only one token in each resource place.

3 Petri Net scheduling using heuristic search for total tardiness

Perhaps one of the most promising approaches for optimization with Petri Nets is to selectively search the net reachability graph with the A* search algorithm as suggested by Lee and DiCesare (1994), Sun et al. (1994), Xiong and Zhou (1998). The algorithm A* search expands the most promising branches of the net reachability graph according to a criterion established with the heuristic function $f(M) = g(M) + h(M)$. The function $g(M)$ is the actual cost (time in the makespan case) from the initial marking M_0 to the marking M and $h(M)$ is an estimate of the cost from marking M to the desired final marking M_f. Those markings having lower values of $f(M)$ will have priority for expansion. The A* search algorithm (Lee and DiCesare 1994) uses two lists: OPEN and CLOSE. The list OPEN contains markings generated but not yet expanded. The list CLOSE contains the markings that have been selected for further expansion. The OPEN list is sorted in ascending order according to the heuristic function $f(M)$.

The reader is referred to Russell and Norvig (1995) for further details of the A* search algorithm.

Condition 1 In order to guarantee that A* Search finds the optimal solution, $h(M)$ must be greater than 0 and less than or equal to the actual value $h^*(M)$ for all markings M (Russell and Norvig 1995).

Additionally, if the final marking is reached for the first time using a heuristic function $h(M)$ that meets Condition 1, then the path from the initial marking to the final marking is an optimal one (Russell and Norvig 1995). In this research, the generation of the successor markings M and, the calculation of the corresponding remaining processing time vector M_r are accomplished with the state equations described in Sect. 2. The cost $g(M)$ is calculated as:

$$g(M) = g(\text{parent of } M) + \delta(M), \tag{2}$$

$\delta(M)$ is the time elapse to reach M from its parent.

Despite the fact that A* search guarantees optimality, it also presents several drawbacks:

(i) The complexity of the algorithm is exponential in both time and memory requirements
(ii) The generation of a large number of non-promising markings,
(iii) The difficulties in the calculation of the heuristic function $h(M)$.

The first two drawbacks were overcome by the BAS algorithm for makespan as shown in Mejía and Odrey (2005). The BAS algorithm expands a maximum fixed number of markings (within a beam of width bw) at each level of the reachability graph, and filters out bad markings with an evaluation scheme. In this way, the algorithm converges towards a final solution at the desired speed set by the beam width. The larger the beam, the better the solution is expected to be at the expense of longer computational time. In addition to those improvements, a filtering scheme was provided (see below). The features of filtered beam search were already implemented by Sabuncuoglu and Bayiz (1999) and Ow and Morton (1988) with good results on classical scheduling problems. The main difference of the classical beam search and BAS is the way the graph is expanded. In the beam search, lower level nodes are expanded first; In BAS, nodes with lower values of the heuristic function $f(M)$ have priority for expansion. This results in faster convergence of the algorithm towards the final marking.

For the tardiness case two additional difficulties arise: The first one relates to the tracking of the jobs' completion times while the second concerns the calculations of both the actual cost $g(M)$ and the estimated remaining cost $h(M)$. In the makespan case, the actual cost $g(M)$ is calculated as the time elapse from the initial marking to the marking M. $g(M)$ is clearly a lower bound of a solution that includes the marking M. The estimated remaining cost $h(M)$ is calculated based on the time of the remaining job operations. For the total tardiness case these calculations are not obvious. For instance, suppose that marking M represents a state in which no jobs have been finished. Clearly in this case, the total tardiness cannot be determined. In addition, in the makespan case, the costs (times) accumulate as the search moves along a path, but this is not the case for total tardiness. Likewise the remaining cost $h(M)$ is not obvious to evaluate. The following sections explain how these difficulties were solved.

3.1 Filtering non-promising markings

The filtering scheme implemented here consists of pre-evaluating each children node before performing the expensive (in terms of computational time) calculation of $h(M)$. The completion time of each job is calculated as the sum of the actual job time plus the total time of its remaining operations. Having the jobs' completion times, the calculation of total tardiness is straight forward. Only the α best children nodes will be further evaluated with the $h(M)$ function. In practice the value of α was variable throughout the algorithm execution, being a greater value at the beginning to diversify the search and a lower value towards the end to intensify the search. The values of α were selected according to the size of the problem. In the remaining of the document the pre-evaluation filtering function will be denoted as filter(M).

3.2 Tracking the jobs' completion times

Since the route of each job is modeled with a Petri subnet, all that is required is tracking the time in which a token reaches the sink place of such a Petri subnet and compare it to the due date of the corresponding job.

3.3 Calculations of the $f(M)$ function

Since the $g(M)$ and $h(M)$ functions do not contribute much in the context of total tardiness, we developed a single cost function $f(M)$ that combines the two. This function is calculated using the common Apparent Tardiness Cost (ATC) or Earliest Due Date (EDD) dispatching rules. This heuristic function denoted here as $f(M)$ is given by the expression:

$$f(M) = f_{\text{rule}}(M), \tag{3}$$

where "rule" can be either ATC or EDD. Starting from the marking M, the function $f_{\text{rule}}(M)$ assigns firing priorities to the transitions according to the selected rule. For instance, the ATC rule selects the job j with the maximum $I_j(t)$ defined as follows:

$$I_j(t) = \frac{w_j}{p_j} \exp\left(\frac{-\max(d_j - p_j - t, 0)}{Kp} \right), \tag{4}$$

where w_j is the weight or priority; p_j is the processing time; d_j is the due date; t is the current time; K is a scaling factor and p is the average processing time of the remaining jobs.

Again, as every transition corresponds to a job (i.e. a given transition belongs to a unique job subnet), it is not difficult to implement the rule. An important point here is that to any marking M reached from the initial marking, two pieces of information are required in order to calculate the total tardiness: The number of jobs already finished at marking M and their completion times (if any jobs have been finished). The completion times had to be calculated previously along the best path to the marking M. This information must be stored along with the remaining time for each marking.

Initially any problem is evaluated with the two rules (ATC and EDD) and the best rule is selected as the rule for the $f_{\text{rule}}(M)$ function. In about the 80% of all cases, the ATC rule was chosen over the EDD. Notice that the Condition 1 described above cannot be guaranteed for all markings but at least provides an upper bound for the value of total tardiness.

3.4 The beam A* search algorithm for total tardiness

The inputs for the BAS algorithm are the Timed Petri Net, the initial marking, the set of due dates, the beam width (bw), the value of the filtering constant α.

The BAS algorithm follows next:

1. Select the rule for expansion (ATC or EDD). Try several values of the scaling factor K for the ATC rule.
2. Place the initial marking M_0 on the list OPEN.
3. Initialize current_depth $= 0$ and count_markings $= 0$ (counter).
4. If OPEN is empty, terminate with failure.
5. Remove the first marking M on OPEN whose depth equals the current_depth. If no marking is on the current depth, select the first marking on OPEN whose depth is greater than current_depth. If no marking on OPEN has a greater depth than current_depth then select the marking with the greatest depth. Put M on CLOSE.
6. If M is the final marking M_f, construct the path from M_0 to M and terminate.
7. Find the set of enabled transitions $\{t_j\}$ for a given marking. Generate the children markings M'' along with the corresponding remaining processing time vector M''_r that would result from firing each enabled transition t_j. Also calculate filter(M'') for each M''. Store these markings on a temporary list (TEMPLIST).

8. (Filtering). Sort TEMPLIST according to filter(M''). Keep only the best α markings on TEMPLIST.
9. For each of the markings M'' remaining on TEMPLIST do the following:
   (a) Calculate $f_{\text{rule}}(M'')$ according to the selected dispatching rule from step 1.
   (b) If M'' is equal to some marking M^{O} already on OPEN, verify if $f(M'') < f(M^{\text{O}})$. If that is the case redirect the path to M^{O}. Otherwise insert M'' on OPEN.
   (c) If M'' is equal to a marking M^{C} already on CLOSE, verify if $f(M'') < f(M^{\text{C}})$. If that is the case delete M^{C} from CLOSE and all its children that reside on OPEN. Redirect the path to M^{C}. Otherwise, insert M'' on CLOSE. This step follows the same logic as step 9(b).
   (d) If M'' is not on either list, then insert M'' on OPEN.
10. If count_markings $<$ bw then add 1 to count_markings. Otherwise, set count_markings $= 0$ and current_depth $=$ current_depth $+ 1$.
11. Go to step 4.

The following section illustrates the performance of the BAS algorithm for total tardiness.

4 Computational experiments and results

In order to test the validity of our approach, we conducted a number of computational experiments on Petri Net models. The primary idea was testing the BAS algorithm for total tardiness on a variety of problems. BAS algorithm was coded entirely in C++. All the experiments were run on a personal computer having a 1.8 GHz Pentium III microprocessor and 512 MB RAM memory. The first part of this section shows the performance of the algorithm on randomly generated problems of flexible manufacturing systems (definition below); the second part shows results on standard and randomly generated Flexible Job Shop Scheduling Problems (FJSSP). A FJSSP is an extension of the Classical Job Shop Scheduling Problem (JSSP) in which there are n jobs, m workstations, no recirculation, no alternative routings and deterministic processing times. Unlike the in the JSSP, in which there is only one machine per workstation, in the FJSSP there can be several identical parallel machines at each station.

4.1 Results on randomly generated problems

A first set of problems consisted of instances having between 5 and 10 jobs, 5 and 10 machines, re-circulation, multiplicity of resources (one or more resources perform simultaneously the same operation on a job) and approximately 25% of the operations could be performed by more than one machine. The due dates were set as 1.5 times the work time of each job for both sets of problems. This due date assignment was proposed by Baker (1974). In the case of alternative routings, the due date was calculated considering the shortest route. The Petri Net model for each of these set of problems was built with the modelling methodology described in Sect. 1.

The literature has not reported results for this kind of problems (to the best of our knowledge). In addition, as pointed out by Sabuncuoglu and Bayiz (1999) optimal solutions for tardiness related criteria are rarely known. For this reason, we had no recourse but to compare our BAS algorithm against dispatching rules.

The purpose of the experiments was testing how the beam width affected the performance of the algorithm. For each run, the average relative difference against the EDD and the ATC

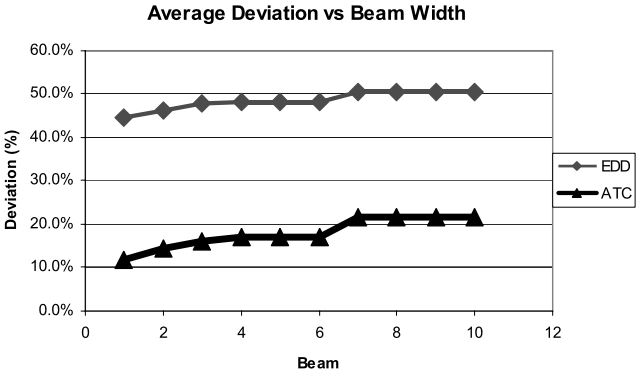

Fig. 3 Average deviation of results. BAS algorithm vs. EDD and ATC rules

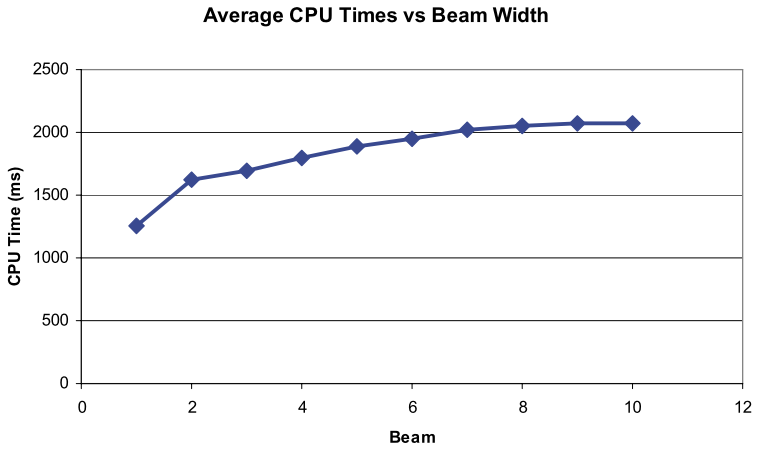

Fig. 4 Average CPU time vs. Beam Width

rules was calculated as follows:

$$\text{deviation}(\%) = \frac{Z(\text{rule}) - Z(\text{BAS})}{Z(\text{rule})} \times 100\%, \tag{5}$$

where $Z(\text{rule})$ is the objective function (total tardiness) obtained with the selected dispatching rule and $Z(\text{BAS})$ is the objective function obtained with the BAS algorithm.

The results were plotted against the beam width as shown in Fig. 3. The tests also show the CPU (computational) time varies as the beam width increases. The averages were taken of the number of explored markings and CPU times on all 10 problems for every beam width. The results are plotted in Fig. 4. In all cases the constant α was set to 5 after a number of preliminary tests.

The results show that even with small beams, the BAS algorithm clearly outperforms the EDD and ATC rules. For a beam width of 1, BAS outperforms the EDD rule on average by 45% and the ATC rule by 10%. For greater beams, BAS achieves an improvement of approximately a 50% compared to EDD and 20% compared to ATC. BAS exhibits the typical behavior of "diminishing returns" (i.e. no significant gains were obtained with in-

creasing computer effort). On the other hand, the CPU times do increase as the beam width is increased as shown in Fig. 4. Hence, if required, small beam widths can be used with no significant loss in the solution quality.

4.2 Results on standard flexible job shop scheduling problems

The flexible job shop scheduling problem (FJSSP) is an extension of the classical job shop scheduling problem. In a FJSSP, there are n jobs and m workstations and jobs follow a predetermined route. A workstation consists of m_i identical parallel machines and a job operation can be performed on any of the machines of the workstation. We tested our algorithm with 12 instances proposed by Brandimarte (1993). These instances are modified versions of a classical job shop problems having between 1 to 3 identical parallel machines in each workstation. The instances mt10xx are 10×10 (jobs $\times$ machines) scheduling problems; the setbxxx instances are 15×10 and the seti5xxx instances are 15×15. Due dates were set with the same Baker (1974) rule as in the previous section. The filtering constant α was set as 5 as in the previous section.

The results were compared with those of the General Shifting Bottleneck (SB) algorithm for total weighted tardiness. Implemented in the software Lekin® (Pinedo and Chao 1999) and the ATC, SPT (Shortest Processing Time) and EDD rules. Table 4 shows a summary of the obtained results. The computational times for the dispatching rules were less than 0.1 seconds and are not reported here. The results reveal the advantages of the BAS approach for total tardiness. In both sets (Brandimarte 1993 and Randomly Generated) of problems, the proposed algorithm outperformed all the dispatching rules and the Shifting Bottleneck (SB) algorithm in all FJSSP instances. On the other hand, the computational times favor our algorithm when compared against the SB method. Notice that the computational times for

Table 4 Results for standard problems for total tardiness

Problem	BAS(2)	BAS(2) (CPU)	BAS(5)	BAS(5) (CPU)	SB	SB (CPU)	ATC	SPT	EDD
mt10c1	818	3	518	6	563	3	1071	1082	1345
mt10cc	548	4	378	7	376	5	793	1080	1173
mt10x	561	4	434	7	523	3	880	1313	854
mt10xyz	260	4	129	7	326	6	732	841	412
setbc9	1391	9	1273	13	2297	10	2358	2550	2201
setb4cc	1666	9	1211	13	1279	13	2528	2959	2921
setb4xx	1256	9	1101	12	2181	9	2262	3124	3253
setb4xyz	1074	10	689	15	1400	14	1908	3173	2168
setb4xy	1292	8	882	12	2123	10	1972	2563	3045
seti5c12	255	17	123	50	298	18	1478	2507	2439
seti5xxx	284	20	142	56	1332	19	1510	2566	2288
seti5xy	163	18	33	51	441	18	1318	2363	1925
seti5xyz	55	19	0	52	464	17	1071	1675	1570

BAS(x): Beam A* Search (x = beam width)

SB: General shifting bottleneck method for total tardiness

CPU: Computational time in seconds

Table 5 Results for randomly generated FJJSPs for total tardiness

Problem	BAS(2)	BAS(2) (CPU)	BAS(5)	BAS(5) (CPU)	SB	SB (CPU)	ATC	EDD	SPT
1 (25 × 6)	143	13	135	26	162	119	221	248	225
2 (30 × 8)	883	23	876	40	850	517	1099	1063	1072
3 (25 × 10)	61	26	43	45	46	301	222	324	278
4 (25 × 10)	844	20	841	31	1028	205	1066	1138	950
5 (30 × 10)	1560	22	1560	38	1661	255	1747	1744	1702
6 (35 × 4)	918	9	918	20	1232	237	1088	1123	1116
7 (30 × 5)	1335	10	1313	15	1624	169	1454	1575	1473
8 (25 × 6)	752	10	752	18	838	99	882	896	981
9 (25 × 6)	821	10	821	18	898	92	957	923	984
10 (25 × 7)	700	12	700	21	665	135	874	784	874

BAS(x): Beam A* Search (x = beam width)

SB: General shifting bottleneck method for total tardiness

CPU: Computational time in seconds

BAS are a fraction of those of the SB method running under the same conditions (same machine). Also notice the improvement when running larger beam widths (bw = 2 vs. bw = 5) at the expense of longer computational times.

4.3 Results on randomly generated flexible job shop scheduling problems

The second set of problems consisted of randomly generated FJSSP instances. The number of jobs ranged from 25 to 35 and the number of workstations between 4 and 10; the number of machines per workstation varied from 1 to 3. The assignment of due dates was performed with the Baker rule as above. The comparison was made against the SB routine and the aforementioned dispatching rules. The filtering constant α was set between 5 and 10 depending on the size of the problem (5 for the smaller problems and 10 for the larger problems). Table 5 shows the results.

Similar conclusions as in Sect. 4.2 can be drawn here. BAS performed better than the SB algorithm and all the dispatching rules. Again smaller beams produced quicker results but at the expense of the solution quality.

It should be noticed that this Petri Net-based approach is very adaptable to many environments such as (flexible) job shops, flow shops, parallel machines, etc. Furthermore, more complex systems can be modeled as presented in Sect. 4.1. This is very appealing in practice since the BAS algorithm is a general purpose algorithm as opposed to many tailor-made algorithms which are suited for only a particular framework (job shop, flow shop) and/or for only a specific objective function.

5 Conclusions

A Petri Net-based algorithm for minimizing total tardiness has been presented in this research. This paper has shown how Petri Nets can be effectively used not only for modeling and simulation of manufacturing systems but also as a powerful optimization tool. In this

paper we presented an approach that exploits the modeling capability of the Petri Nets combined with an intelligent search method for scheduling a wide variety of manufacturing systems. The BAS algorithm for total tardiness features an intelligent pruning of the search space, a controlled search deepening to avoid marking explosion and the development of new heuristic functions to estimate and minimize total tardiness. The BAS algorithm described here was tested on a number of randomly generated problems and showed significant improvement upon the common dispatching rules used to sequence jobs in flexible manufacturing systems. The BAS algorithm was also tested on standard FJSSPs in order to have a benchmark for comparison. BAS performed very well in all instances both in terms of solution quality and in computational times.

Further research is needed on (i) improving the heuristic functions $f(M)$, (ii) the implementation of the algorithm on large problems, and (iii) the development of multi-criteria heuristic functions.

Other areas that are the subject of further research are the utilization of the BAS algorithm for cyclic scheduling, assembly operations where parts are incorporated into subassemblies, re-entrant manufacturing, and re-scheduling of flexible manufacturing systems.

Appendix

Numerical example: Let the following Petri Net representation of two jobs processed by two different machines. The description of places follows is Table 6.

Places p_2 and p_5 are timed places. The corresponding time delays are $\tau_2 = 4$ and $\tau_5 = 6$.

Assume the following firing sequence $\sigma = (t_1, t_3, t_2, t_4)$. All transitions fire as soon as they are enabled. The initial marking is shown in Fig. 5. Notice that both t_1 and t_3 fire at time $t = 0$ since they are both enabled and the corresponding input places are not timed.

Definition Let t_{clock} be the cumulative time elapse.

Fig. 5 Petri Net example

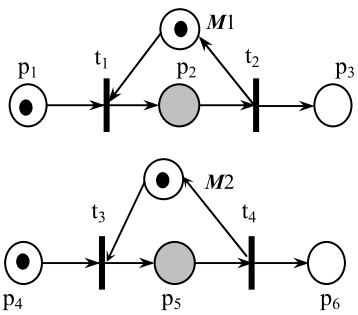

Table 6 Description of places

Place description for Petri Net of Fig. 2			
p_1	Job 1 ready	p_4	Job 2 ready
p_2	Job 1 being processed by machine 1	p_5	Job 1 being processed by machine 2
p_3	Job 1 finished	p_6	Job 2 finished
M1	Machine 1 available	M2	Machine 2 available

Notice that the (1) $X(k+1) = A(k)X(k) + B(k)u(k)$ can be rewritten as:

$$M(k+1) = M(k) + Cu(k), \tag{6}$$

$$M_r(k+1) = -I_n\delta(k)M(k) + M_r(k) + TC^+u(k). \tag{7}$$

Following the definitions of Sect. 2, the corresponding matrix C for the net is:

$C =$

	t_1	t_2	t_3	t_4
p_1	−1	0	0	0
p_2	1	−1	0	0
p_3	0	1	0	0
p_4	0	0	−1	0
p_5	0	0	1	−1
p_6	0	0	0	1
M1	−1	1	0	0
M2	0	0	−1	1

The time delay matrix $T = \text{diag}(0,\ 4,\ 0,\ 0,\ 6,\ 0,\ 0,\ 0)$.
$M(0) = M_0 = [1\ 0\ 0\ 1\ 0\ 0\ 1\ 1]^{\mathrm{T}}$; $M_r(0) = [0\ 0\ 0\ 0\ 0\ 0\ 0\ 0]^{\mathrm{T}}$ and $t_{\text{clock}} = 0$ or similarly

$$M(k+1) = [M(k) - C^-u(k)] + C^+u(k), \tag{8}$$

$$M_r(k+1) = [-I_n\delta(k)M(k) + M_r(k)] + TC^+u(k). \tag{9}$$

The terms in brackets correspond to the removal of tokens and must be calculated first. Denote the terms in brackets in (8) and (9) as $M'(k)$ and $M'_r(k)$ respectively. The terms outside the brackets correspond to the update of the marking and the remaining process time vectors. At $k = 0$ and $t = 0$, transition t_1 fires. Therefore the vector $u(0)$ is $[1\ 0\ 0\ 0]^{\mathrm{T}}$. Recall that $u(k)$ is a unit vector in which all positions are 0 except the position corresponding to the transition firing at the event k.

The resulting marking and remaining process time vectors are: Calculation of $M'(0)$ and $M'_r(0)$:

$$M'(0) = [0\ 0\ 0\ 1\ 0\ 0\ 0\ 1]^{\mathrm{T}}.$$

According to the definition given in Sect. 2, $\delta(0) = \max(M_{r\,p1}(0), M_{r\,M1}(0)) = \max(0, 0) = 0$.

$$t_{\text{clock}} = t_{\text{clock}} + \delta(0) = 0.$$

Hence the term $-I_n\delta(0)M(0)$ is a 0 vector and $M'_r(0) = [0\ 0\ 0\ 0\ 0\ 0\ 0\ 0]^{\mathrm{T}}$.

Adding the terms outside the brackets in (8) and (9) results in:

$$M(1) = [0\ 1\ 0\ 1\ 0\ 0\ 0\ 1]^{\mathrm{T}}.$$

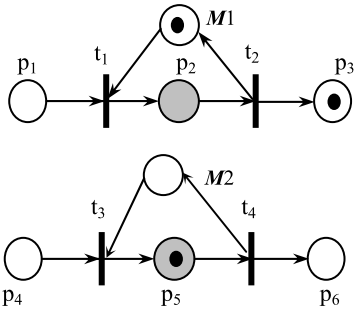

Fig. 6 Marking after firing transitions t_1, t_3 and t_2

The term $TC^+u(k)$ is calculated as follows:

$$\underset{T}{\text{diag}[0\ 4\ 0\ 0\ 6\ 0\ 0]} \underset{C^+}{\begin{vmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}} \underset{u(k)}{\begin{vmatrix} 1 \\ 0 \\ 0 \\ 0 \end{vmatrix}}$$

Hence $M_r(1) = [0\ 4\ 0\ 0\ 0\ 0\ 0\ 0]^{\mathrm{T}}$.

Following the same logic, firing t_3 yields:

$$u(1) = [0\ 0\ 1\ 0]^{\mathrm{T}}, \qquad \delta(1) = 0,$$
$$M(2) = [0\ 1\ 0\ 0\ 1\ 0\ 0\ 0]^{\mathrm{T}}, \qquad M_r(2) = [0\ 4\ 0\ 0\ 6\ 0\ 0\ 0]^{\mathrm{T}},$$
$$\delta(1) = 0 \quad \text{and} \quad t_{\text{clock}} = t_{\text{clock}} + \delta(1) = 0.$$

Firing t_2 results in:

$$u(2) = [0\ 1\ 0\ 0]^{\mathrm{T}}, \qquad M'(2) = [0\ 0\ 0\ 0\ 1\ 0\ 0\ 0]^{\mathrm{T}}, \qquad \delta(2) = \max(M_{r\,p2}(2)) = 4.$$

Hence the term $-I_n\delta(2)M(2)$ results in $[0\ 4\ 0\ 0\ 4\ 0\ 0\ 0]^{\mathrm{T}}$ and $M'_r(2) = [0\ 0\ 0\ 0\ 2\ 0\ 0\ 0]^{\mathrm{T}}$.

Adding the terms outside the brackets in (8) and (9) results in:

$$M(3) = [0\ 0\ 1\ 0\ 1\ 0\ 1\ 0]^{\mathrm{T}} \quad \text{and} \quad M_r(3) = [0\ 0\ 0\ 0\ 2\ 0\ 0\ 0]^{\mathrm{T}},$$
$$t_{\text{clock}} = t_{\text{clock}} + \delta(2) = 4.$$

Notice that job 1 finishes at this time. See Fig. 6.

Finally firing t_4 results in:

$$u(3) = [0\ 0\ 0\ 1]^{\mathrm{T}}, \qquad M'(3) = [0\ 0\ 1\ 0\ 1\ 0\ 1\ 0]^{\mathrm{T}}, \qquad \delta(3) = \max(M_{r\,p5}(3)) = 2.$$

Hence the term $-I_n\delta(3)M(3)$ results in $[0\ 0\ 2\ 0\ 2\ 0\ 2\ 0]^{\mathrm{T}}$ and $M'_r(3) = [0\ 0\ -2\ 0\ 0\ 0\ -2\ 0]^{\mathrm{T}}$. Since negative remaining process times mean that the times have been exhausted, these terms are converted into 0 s.

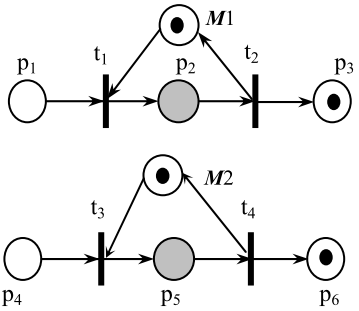

Fig. 7 Marking of the net after all transitions have fired

Adding the terms outside the brackets in (8) and (9) results in:

$$M(4) = [0\,0\,1\,0\,0\,1\,1\,1]^{\mathrm{T}} \quad \text{and} \quad M_r(4) = [0\,0\,0\,0\,0\,0\,0\,1]^{\mathrm{T}},$$
$$t_{\mathrm{clock}} = t_{\mathrm{clock}} + \delta(3) = 6.$$

Notice that the time elapse (makespan) to complete both jobs is 6 time units since both jobs were processed simultaneously. Job 1 finished at 4 time units and Job 2 at 6 time units and this is consistent with the Petri Net equations. See the state of the net in Fig. 7.

References

Baker, K. R. (1974). *Introduction to sequencing and scheduling*. New York: Wiley.

Brandimarte, P. (1993). Routing and scheduling in a flexible job shop by tabu search. *Annals of Operations Research*, *22*, 158–183.

Jeng, M. D., & Chen, S. C. (1998). A heuristic search approach using approximate solutions to Petri Net state equations for scheduling flexible manufacturing systems. *The International Journal of Flexible Manufacturing Systems*, *10*, 139–162.

Lee, D. Y., & DiCesare, F. (1994). Scheduling flexible manufacturing systems using Petri Nets and heuristic search. *IEEE Transactions on Robotics and Automation*, *10*(2), 123–131.

Mejía, G. (2003). Timed Petri Net modeling and optimization with heuristic search for flexible manufacturing workstations. In *Proceedings of the 2003 IEEE emerging technologies and factory automation (ETFA)*, Lisbon, Portugal, September 16–19, 2003.

Mejía, G., & Odrey, N. (2005). An approach using Petri Nets and improved heuristic search for manufacturing system scheduling. *Journal of Manufacturing Systems*, *34*(2), 79–92.

Moore, K. E., & Gupta, S. M. (1996). Petri Net models of flexible and automated manufacturing systems: A survey. *International Journal of Production Research*, *34*(11), 3001–3035.

Murata, T. (1989). Petri Nets: Properties, analysis and applications. *Proceedings of the IEEE*, *77*(4), 541–580.

Ow, P. S., & Morton, T. E. (1988). Filtered beam search in scheduling. *International Journal of Production Research*, *26*, 35–62.

Pinedo, M., & Chao, X. (1999). *Operations scheduling with applications in manufacturing and services*. New York: McGraw-Hill.

Reyes-Moro, A., Yu, H., Kelleher, G., & Lloyd, S. (2002). Integrating Petri Nets and hybrid heuristic search for the scheduling of FMS. *Computers in Industry*, *47*, 123–138.

Russell, S., & Norvig, P. (1995). *Artificial intelligence: A modern approach*. Englewood Cliffs: Prentice-Hall.

Sabuncuoglu, I., & Bayiz, M. (1999). Job shop scheduling with Beam search. *European Journal of Operational Research*, *118*(2), 390–412.

Sun, T., Cheng, W., & Fu, L. (1994). A Petri Net based approach to modeling and scheduling for an FMS and a case study. *IEEE Transactions on Industrial Electronics*, *41*(6), 593–601.

Xiong, H. H., & Zhou, M. C. (1998). Scheduling of semi-conductor test facility via Petri Nets and hybrid heuristic search. *IEEE Transactions on Semiconductor Manufacturing*, *11*(3), 384–393.