Int J Adv Manuf Technol (2010) 48:925–933
DOI 10.1007/s00170-009-2329-8

ORIGINAL ARTICLE

A hybrid heuristic search algorithm for scheduling FMS based on Petri net model

Bo Huang · Yu Sun · Ya-Min Sun · Chun-Xia Zhao

Received: 16 May 2008 / Accepted: 18 September 2009 / Published online: 10 October 2009

Abstract This paper presents a new scheduling method for a flexible manufacturing system (FMS) in a Petri net framework. Petri nets can concisely model multiple lot sizes for each job, the strict precedence constraint, multiple kinds of resources, and concurrent activities. To decrease the likelihood of rejecting the critical markings, our algorithm adopts an improved checking method for previous generated marking. To reduce the computation complexity, an elaborate scheme is applied, which performs A* search locally and backtracking search globally in the reachability graph of the Petri net. Some numerical experiments are carried out to demonstrate usefulness of the algorithm.

Keywords Petri nets · Manufacturing strategy · Scheduling · Heuristics

1 Introduction

In a manufacturing system, scheduling is a typical combinatorial optimization problem, which decides starting times and allocations of jobs to be processed. A desirable method must include two characteristics: (1) easy formulation of the problem and (2) quick identification of semioptimal solutions (with small computational efforts). So, many industry and research communities are now focusing on developing methods for solving real-world flexible manufacturing system (FMS) scheduling problems. However, no perfect solution has been found for all problems, due primarily to the complexity of FMS scheduling. The general FMS scheduling problem belongs to one of the NP-hard combinatorial problems [1] for which the development of optimal polynomial algorithm is unlikely.

The performance of FMS has been recently studied by the Petri net community. It is appropriate to select Petri net-based algorithms for optimization purposes. Despite their popularity as a modeling tool, Petri nets have not received much attention for optimization purposes because of the intractability of their state space [2, 3]. Recent approaches have attempted to use artificial intelligence techniques [3, 4] to selectively search the Petri net reachability graph using the well-known A* search algorithm (see Russell and Norvig [5]). The A* search algorithm presented in [2, 4, 6–9] developed for minimizing the objective function of flow time of parts in the system are all derived from the original algorithm presented in [2]. However, the origin algorithm is not admissible in certain conditions, i.e., it does not always guarantee for an optimal solution even with admissible heuristic function. This problem was also mentioned by Yu et al. [4] who devised a remedy for this problem. But the improved method still cannot guarantee not rejecting markings that lead to the optimal solutions [4].

Another problem observed is the difficulty in finding an optimal or near-optimal solution in a reasonable amount of time for a sizable problem. Some improved algorithms have been introduced to reduce the search effort. Xiong and Zhou [8] have studied hybrid algorithms combining best-first (BF) and backtracking (BT)

B. Huang (✉) · Y.-M. Sun · C.-X. Zhao
School of Computer Science and Technology,
Nanjing University of Science and Technology,
Nanjing, People's Republic of China
e-mail: star_njust@yahoo.com.cn

Y. Sun
School of Mechanical Engineering,
Nanjing University of Science and Technology,
Nanjing, People's Republic of China

search methodologies. The A* algorithms which limit the BT capability of the algorithm by introducing irrevocable decisions were implemented in [6, 10, 11]. Some hybrid search algorithms based on the relaxation of the evaluation scope of an A*-based algorithm have been considered in [4, 9].

In this paper, we present a hybrid scheduling strategy based on the timed Petri nets. The search scheme adopts (1) an improved checking method for previous generated marking to decrease the likelihood of rejecting the critical markings and (2) a more elaborate scheme, which performs BF search locally and BT search globally, than that of Xiong and Zhou [8]. Then, the scheduling results of the method are derived and compared with that of algorithm BF-BT and BT-BF through an example with different sets of lot sizes. The algorithm is also applied to a set of randomly generated more complex FMS with such characteristics as buffers with limited sizes, operations with multiple resources, and jobs with alternative routings.

2 Modeling and scheduling of FMS based on Petri net structures

A Petri net is defined as a bipartite directed graph containing places, transitions, and directed arcs connecting places to transitions and transitions to places. Pictorially, places are depicted by circles and transitions as bars or boxes. A place is an input (output) place of a transition if there exists a directed arc connecting this place (transition) to the transition (place). A place can contain tokens pictured by black dots. It may hold either none or a positive number of tokens. At any given time instance, the distribution of tokens on places, called Petri net marking, defines the current state of the modeled system. Thus, a marked Petri net can be used to study the dynamic behavior of the modeled discrete event system.

Formally, a Petri net [12, 13] is defined as $Z=(P, T, I, O, m_0)$ where:

- P $\{p_1, p_2, \ldots, p_n\}$, $n>0$ is a finite set of places;
- T $\{t_1, t_2, \ldots t_s\}$, $s>0$ with $P \cup T = \emptyset$ and $P \cap T = \emptyset$is a finite set of transitions;
- I $P\times T\rightarrow\{0, 1\}$ is an input function or direct arcs from P to T;
- O $P\times T\rightarrow\{0, 1\}$ is an output function or direct arcs from T to P;
- m $P\rightarrow\{0, 1, 2, \ldots\}$ is a $|P|$ dimensional vector with $m(p)$ being the token count of place p, m_0 is an initial marking.

In order to simulate the dynamic behavior of a system, a state or marking in a Petri net is changed according to the following transition (firing) rules:A transition is enabled if $m(p)\geq I(p, t)$ for any $p\in P$.An enabled transition t can fire at marking m', and its firing yields a new marking, $m(p) = m'(p) + O(p,t) - I(p,t)$, for arbitrary p from P.

The marking m is said to be reachable from m'. Given Z and its initial marking m_0, the reachability set is the set of all marking reachable from m_0 through various sequences of transition firings and is denoted by $R(Z, m_0)$. For a marking $m\in R(Z, m_0)$, if no transition is enabled in m, then m is called a deadlock marking, and the corresponding system is in a deadlock state.

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 [2, 8, 11]. A place represents a resource status or an operation, a transition represents either the start or completion of an event or operation process, and the stop transition for one activity will be the same as the start transition for the next activity following [2]. Token(s) in a resource place indicates that the resource is available. A token in an operation place represents that the operation is being executed and no token shows none being performed. A certain time may elapse between the start and the end of an operation. This is represented by associating timing information with the corresponding operation place.

An event-driven schedule is searched in a timed Petri nets framework to achieve minimum or near minimum makespan. This paper employs deterministic timed Petri nets by associating time delays with places. The transitions can be fired with a zero duration which is consistent with the definition of nontimed Petri nets. In the Petri net model of a system, firing an enabled transition changes the token distribution (marking). A sequence of firings results in a sequence of markings and all possible behaviors of the system can be completely tracked by the reachability graph of the net. The search space for the optimal event sequence is the reachability graph of the net, and the problem is to find a firing sequence of the transitions in the Petri net model from the initial marking to the final one. The A* algorithm applied to Petri nets was first introduced by Lee and Dicesare [2]. The A* search is an informed search algorithm that expands only the most promising branches of the reachability graph of a timed place Petri net. The basic algorithm [2] is as follows:

Algorithm 1 (A)*

1 Put the initial marking m_0 on the list OPEN.
2 If OPEN is empty, terminate with failure.
3 Remove the first marking m from OPEN and put m on the list CLOSED.
4 If m is the final marking, construct the optimal path from the initial marking to the final marking and terminate.

5 Find the enabled transitions of the marking m.
6 Generate the next marking, or successor, for each enabled transition, and set pointers from the next markings to m. Compute $g(m')$ for every successor m'.
7 For every successor m' of m:
   a if m' is already on OPEN, direct its pointer along the path yielding the smallest $g(m')$.
   b if m' is already on CLOSED, direct its pointer along the path yielding the smallest $g(m')$. if m' requires pointer redirection, move m' to OPEN.
   c Calculate $h(m')$ and $f(m')$, and put m' on OPEN.
8 Reorder OPEN in the increasing magnitude of f.
9 Go to step 2.

The function $f(m)$ in algorithm 1 is calculated from the following expression: $f(m)=g(m)+h(m)$. $g(m)$ represents the makespan of the partial schedule determined so far. On the other hand, $h(m)$, called the *heuristic function*, represents an estimate of the remaining cost (makespan) to reach the marking that represents the goal state m_f. The purpose of a heuristic function is to guide the search process in the most profitable direction by suggesting which transition to fire first.

If $h(m)$ is a lower bound to all complete solutions descending from the current marking, i.e.:

$$\mathrm{h(m)} \leq \mathrm{h^*(m)}, \forall \mathrm{m} \tag{1}$$

where $h^*(m)$ is the optimal cost of paths going from the current marking m to the final one, the $h(m)$ is admissible, which guarantees for an optimal solution [14].

At each step of the A* search process, the most promising of the markings generated so far is selected. This is done by applying an appropriate heuristic function to each of them. Then, it expands the chosen marking by firing all enabled transitions under this marking. If one of successor markings is a final marking, the algorithm quits. If not, all those new markings are added to the set of markings generated so far. Again, the most promising marking is selected and the process continues. Once the Petri net model of the system is constructed, given initial and final markings, an optimal schedule can be obtained using the above algorithm with admissible heuristic function [2].

3 Checking for previous generated markings

In the above search process, when a new marking is obtained, as a successor to the node currently explored, it can be compared with all the markings generated up to that point. The same markings may represent different paths to achieve the same state. A condition must be checked to decide whether the new path is more promising than the existing one. In algorithm 1, the simple test of comparing markings and the current makespan does not satisfy this, thus it may make the algorithm reject the path that leads to an optimal solution [15].

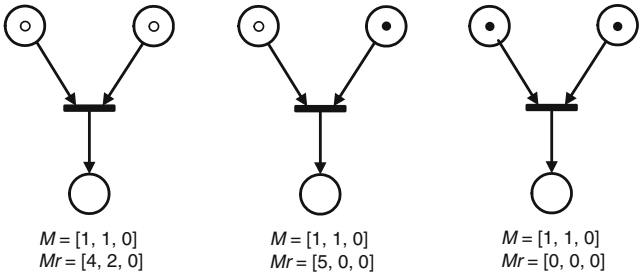

Fig. 1 Transition firing description

This problem was also recognized by Yu et al. [4] who devised a remedy for the problem using the following alternative makespan cost function to replace $g(.)$ when comparing markings:

$$j(M) = g(M) + \sum t_i \Big/ |U_M| \tag{2}$$

where $\sum t_i$ is the sum of the remaining time of the unavailable tokens and $|U_M|$ is the number of the unavailable tokens (an unavailable token in a place means that, although the token is not ready yet, it will become available after a fixed time). However, it still cannot guarantee not rejecting markings that lead to the optimum solutions [4].

To address these issues, we have to understand the process of the timed Petri net state representation. Since the concept of time delay is associated with places, tokens can have two possible states. A token in a place p may mean (1) this token can already be considered as an input token for any transition that has p as an input place or (2) although the token is not ready yet, it will become available after a fixed time (it often happens in the operation places). So, in the second condition, there exists a time left (t_l) for this kind of tokens to be available. As time passes by, t_l is decreased until it reaches zero. When t_l reaches zero, these

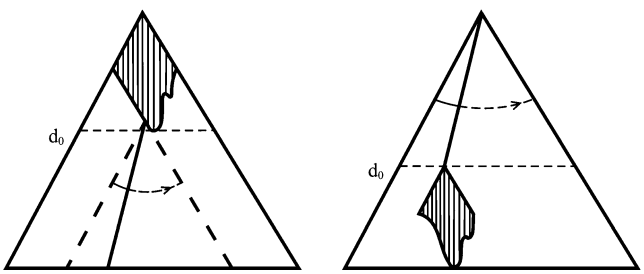

Fig. 2 **a** BF search on top (*shaded area*) followed by BT ending and **b** BT start (*left-to-right arrow*) followed by BF ending

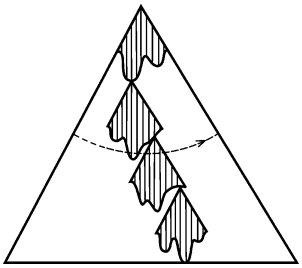

Fig. 3 Perform BF locally and BT globally

tokens become available. Thus, additional time t_l may be required to actually "reach" m. So, the conditions for path updating $m' = m^{O} \wedge g(m') < g(m^{O})$ in step 7a (or $m' = m^{C} \wedge g(m') < g(m^{C})$ in step 7b) of algorithm 1 and $m' = m^{O} \wedge j(m') < j(m^{O})$ in Yu et al. [4] do not necessarily imply a new better path from m_0 to m' (m_0 is the initial marking, m^{O} and m^{C} represent a marking on list OPEN or CLOSED, respectively). In this section, we propose an alternate checking method that considers markings and remaining process time of tokens simultaneously for selecting the better node. To express our method, we need the definition of *timed marking*.

Definition 1: timed marking of a timed Petri net A timed marking m for a timed Petri net is a three-tuple (M, Mr, g) where:

- M is the marking of both available and unavailable tokens in the net;
- Mr is a vector containing remaining process time of tokens in each place;
- g is the cost of path to reach M from M_0.

We use M_m, Mr_m, and $g(M_m)$ to represent the elements of a time-marking m. Now let us consider the three states of a timed Petri net where M is the same for these cases (Fig. 1). Then, if $g(M_m)<g(M^{O})$ and one by one the values of Mr_m are not bigger than that of Mr^{O}, it can be established that a better path between M_0 and M_m was found (the last condition is denoted as $Mr_m \leq Mr^{O}$ in this paper). This is described in algorithm 2.

Algorithm 2 (improved A)*

1 Place the initial marking m_0 on the list OPEN.
2 If OPEN is empty, terminate with failure.
3 Remove the first marking m from OPEN and put m on the list CLOSED.
4 If m is the final marking m_f, construct the optimal path from m back to m_0 and terminate.
5 Find the set of enabled transitions $\{t_j\}$ (j=1…$et(m)$). $et(m)$ is number of enabled transitions when the marking is m.
6 Generate the children markings m'_j that would result from firing each enabled transition t_j and calculate $g(m'_j)$, $h(m'_j)$, and $f(m'_j)$.
7 For each of the marking M'_j, do the following:
   a If m'_j is equal to some marking m^{O} already on OPEN, compare Mr and g. If $Mr'_j \geq Mr^{O} \wedge g(m'_j) \geq g(m^{O})$, go to step 2; If $Mr'_j \leq Mr^{O} \wedge g(m'_j) < g(m^{O})$, delete m^{O} from OPEN and insert m'_j on OPEN. Otherwise, insert m'_j on OPEN.
   b If m'_j is equal to some marking m^{C} already on CLOSED, compare Mr and g. If $Mr'_j \geq Mr^{C} \wedge$

Table 1 Scheduling results of the example for lot size (5, 5, 2, 2)

Algorithm	Depth of BF or M_{max}	Makespan	Number of markings	CPU time (s)
A*		58	3,437	14
BT		105	85	571
BF-BT	20	94	571	0.65
	40	85	1,607	4
	50	79	2,123	6
	60	74	2,775	8
	80	64	3,308	11
BT-BF	20	88	248	0.38
	40	80	484	0.8
	50	70	1,247	3.6
	60	64	1,520	6.5
	80	62	1,687	7
A* locally and BT globally	1	75	230	0.17
	5	68	358	0.31
	10	67	634	0.62
	15	65	540	0.46
	20	62	946	0.78
	25	62	921	0.93
	30	60	1,423	1.12

Table 2 Scheduling results of the example for lot size (8, 8, 4, 4)

Algorithm	Depth of BF or$M_{\max}$	Makespan	Number of markings	CPU time (s)
A*		100	9,438	112
BT		198	145	0.23
BF-BT	40	168	3,888	24
	60	154	5,234	38
	80	140	7,699	49
	100	127	8,819	90
	120	108	9,233	104
BT-BF	40	163	585	1.4
	60	140	1,590	7
	80	121	2,873	18
	100	112	4,545	36
	120	104	8,045	76
A* locally and BT globally	1	128	422	0.31
	5	116	570	0.46
	10	111	782	0.62
	15	112	1,047	1.56
	20	109	1,694	2.65
	25	105	1,585	2.03
	30	105	1,780	2.65

$g(m'_j) \geq g(m^C)$, go to step 2; If $Mr'_j \leq Mr^C \wedge g(m'_j) < g(m^C)$, delete m^C from CLOSED and insert m'_j on OPEN. Otherwise, insert m'_j on OPEN.

c If m'_j is not on either list, insert m'_j on OPEN.

8 Reorder OPEN in the increasing magnitude of f.

9 Go to step 2.

In this algorithm, when a marking is generated we compare its values of M, Mr, and g with that of markings already generated. If it is better than some marking already on OPEN or CLOSED, we insert it on OPEN and delete the marking on OPEN or CLOSED. If worse, we directly reject it. If the quality of the marking cannot be evaluated

Table 3 Scheduling results of the example for lot size (10, 10, 6, 6)

Algorithm	Depth of BF or$M_{\max}$	Makespan	Number of markings	CPU time (s)
A*		134	23,092	720
BT		274	193	0.38
BF-BT	80	206	6,281	64
	100	198	12,341	240
	120	180	16,602	480
	140	169	20,155	540
	160	153	21,797	660
BT-BF	80	209	1,254	5
	100	181	2,315	16
	120	162	8,495	139
	140	150	11,368	390
	160	148	18,875	560
A* locally and BT globally	1	170	581	0.62
	5	155	809	0.93
	10	146	1,089	1.25
	15	147	1,538	2.50
	20	144	1,889	2.81
	25	142	2,850	5.62
	30	141	2,995	7.03

temporarily, we insert it on OPEN. This method presents modifications to the basic algorithm introduced by Lee and DiCesare [2] and it will not reject the promising markings. Additional details can be seen in [15].

4 A*-BT Combinations

A* search is optimal and optimally efficient. But for sizable scheduling problems, it is very difficult to find the optimal solution in a reasonable amount of time and, in many problems, A* spends a large amount of time discriminating among paths whose costs do not vary significantly from each other [14].

To reduce the memory space and computation time required by a pure A* strategy, Xiong and Zhou [8] have proposed two hybrid search methods (BF-BT and BT-BF) which combine the heuristic A* strategy with the BT strategy based on the execution of the Petri net. BF-BT is depicted in Fig. 2a where the A* strategy is applied at the top of the search graph (represented by the shaded area with the irregular frontier) and a BT strategy at the bottom (represented by the left-to-right arrow). As soon as a depth-bound d_0 is reached, the BT search takes over from the best node on OPEN until the entire graph beneath that node is traversed. Another algorithm BT-BF is shown in Fig. 2b. Here, BT is employed at the top of the graph, whereas A* is used at the bottom. BT is applied until a depth-bound d_0 is reached. At this point, instead of backing up, the A* search is started from the node at d_0 until it returns a solution or exists with failure.

Because the performance of the BF search is usually at its best at the bottom of the search graph, BT-BF performs much better than BF-BT [8]. However, algorithm BT-BF presents two major drawbacks: (1) The important decisions with respect to the quality of a schedule may happen at the early stages of the scheduling activity [8]; this increases the likelihood of missing the critical candidates for the BT-BF search which employs BT search instead of BF search at the early stage. (2) The depth of the solution is usually not known a priori, so it is hard for the BT-BF strategy to decide where to trigger the transition from BT to BF is right.

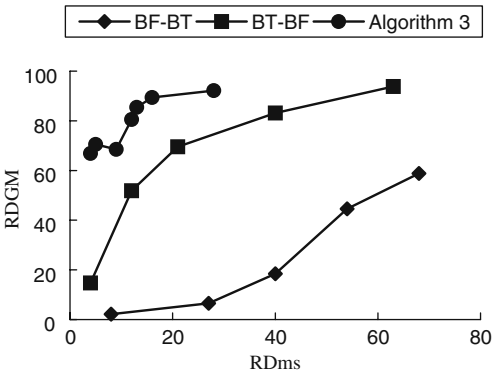

Fig. 5 RDGM versus RDms for lot size (8, 8, 4, 4)

A more elaborate scheme, which performs BF locally and BT globally, is proposed in this section. It is depicted in Fig. 3. We begin searching in an A* manner. At each loop of the search process, we check the number of nodes on OPEN after all successor nodes have been generated from the father node. Once it exceeds a threshold M_{max}, we regard all the nodes on OPEN as direct successors of the root node and submit them to a BT search. BT selects the best among these successors and "expands" it using BF search; that is, it submits the chosen node to a local A* search until the number of nodes on OPEN exceeds M_{max} again and treats the new nodes on OPEN as direct successors of the node "expanded." This strategy is implemented in algorithm 3.

Algorithm 3 (A locally and BT globally)*

1 i=0.
2 Place the initial marking m_0 on the list OPEN$_i$.
3 If i>0, remove the first marking m from list OPEN$_{i-1}$ and put m on list OPEN$_i$
4 If OPEN$_i$ is empty, check i. If i=0, terminate with failure. If i>0, go to step 3.

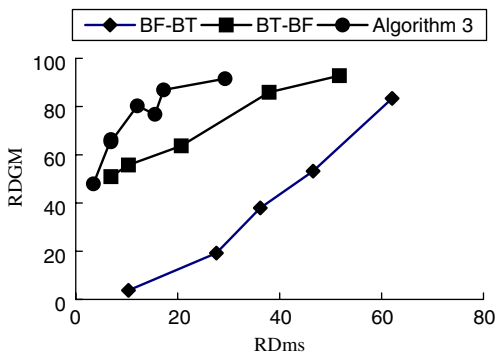

Fig. 4 RDGM versus RDms for lot size (5, 5, 2, 2)

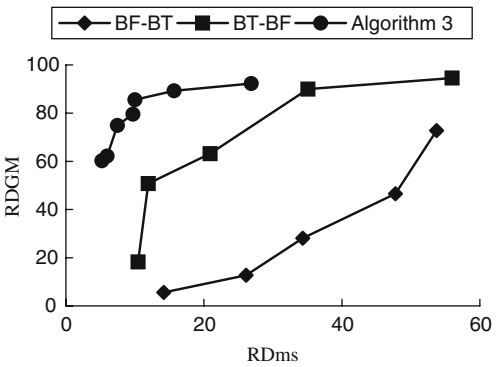

Fig. 6 RDGM versus RDms for lot size (10, 10, 6, 6)

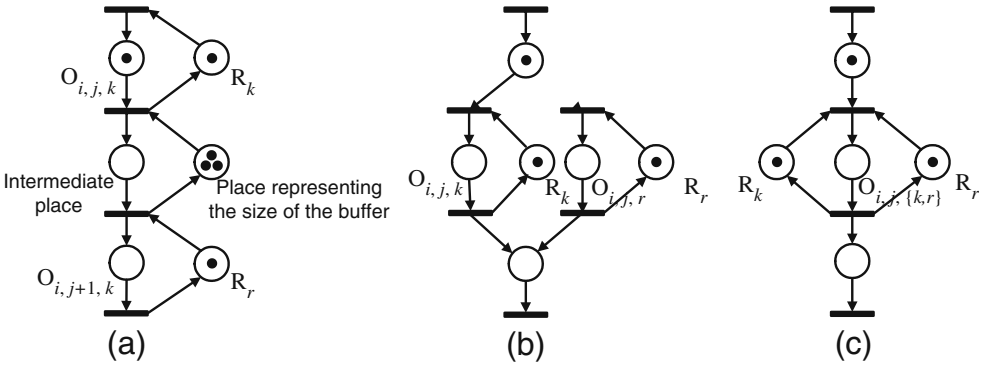

Fig. 7 **a** A model for a buffer with a finite size, **b** a model for a job having alternative routings, and **c** a model for an operation with dual resources

5 Remove the first marking m from list OPEN_i and put m on the list CLOSED_i.
6 If m is the final marking m_f, construct the path from m back to m_0 and terminate.
7 Find the set of enabled transitions $\{t_j\}$ (j=1…$et(m)$). $et(m)$ is number of enabled transitions when the marking is m.
8 Generate the children markings m'_j that would result from firing each enabled transition t_j and calculate $g(m'_j)$, $h(m'_j)$, and $f(m'_j)$.
9 For each of the marking M'_j, do the following:
   a If m'_j is equal to some marking m^{O} already on OPEN_i, compare Mr and g. If $Mr'_j \geq Mr^{\text{O}} \wedge g(m'_j) \geq g(m^{\text{O}})$, go to step 4; If $Mr'_j \leq Mr^{\text{O}} g(m'_j) < g(m^{\text{O}})$, delete m^{O} from OPEN_i and insert m'_j on OPEN_i. Otherwise, insert m'_j on OPEN_i.
   b If m'_j is equal to some marking m^{C} already on CLOSED_i, compare Mr and g. If $Mr'_j \geq Mr^{\text{C}} \wedge g(m'_j) \geq g(m^{\text{C}})$, go to step 4; If $Mr'_j \leq Mr^{\text{C}} \wedge g(m'_j) < g(m^{\text{C}})$, delete m^{C} from CLOSED_i and insert m'_j on OPEN_i; Otherwise, insert m'_j on OPEN_i.
   c If m'_j is not on either list, insert m'_j on OPEN_i.
10 Reorder OPEN_i in the increasing magnitude of f.
11 If the number of nodes on OPEN_i exceeds $M_{\max}$, i=i+1 and go to step 3; otherwise, go to step 4.

In summary, this strategy amounts to running an informed depth-first search where each node expansion is accomplished by a memory-limited A* search and the nodes on OPEN are defined as children.

For the example in Xiong and Zhou [8], the three sets of lot size (5, 5, 2, 2), (8, 8, 4, 4), and (10, 10, 6, 6) are tested. We employ algorithm 3 with the same admissible heuristic function $h(m)$ as that in Xiong and Zhou [8]. The code was written in C# in its entirety. The tests were performed on personal computers having an AMD Athlon microprocessor at a speed of 1 GHz with 512 MB of memory. The scheduling results of makespan, number of generated markings, and computation time are shown in Tables 1, 2, and 3 for lot sizes (5, 5, 2, 2), (8, 8, 4, 4), and (10, 10, 6, 6), respectively. The results for different cases obtained from A* search, BT method, and BF-BT and BT-BF algorithms in Xiong and Zhou [8] are also shown in these tables.

All the algorithms BF-BT, BT-BF, and algorithm 3 cut down the computation complexity by narrowing the evaluation scope at the expense of losing the optimality. The relations of the number of generated markings reduced versus optimality lost are shown in Figs. 4, 5, and 6 for three different sets of lot size (5, 5, 2, 2), (8, 8, 4, 4), and (10, 10, 6, 6) respectively. In these figures, the percentage of optimality lost, which is the comparison of the makespan, is equal to:

$$\text{RDms} = \frac{\text{ms(Hybrid)} - \text{ms(A*)}}{\text{ms(A*)}} \times 100\% \tag{3}$$

and the percentage of computation complexity reduced, which is the comparison of the storage (number of generated markings), is equal to:

$$\text{RDGM} = \frac{\text{GM}\left(\text{A}^*\right) - \text{GM(Hybrid)}}{\text{GM}\left(\text{A}^*\right)} \times 100\%. \tag{4}$$

From the testing results, the following conclusions are drawn.

The hybrid heuristic search which employs A* locally and BT globally in the Petri net reachability graph

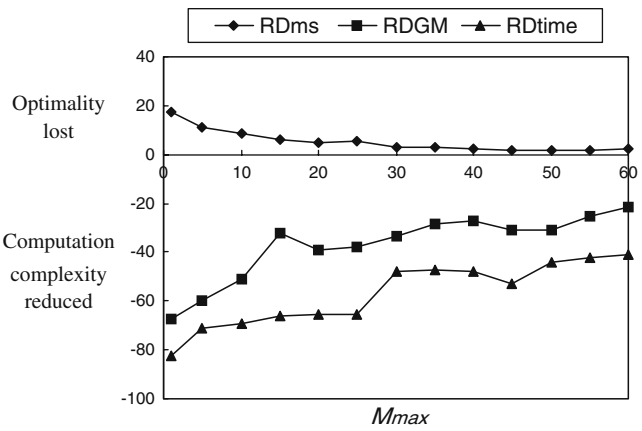

Fig. 8 Performance of algorithm 3

(algorithm 3) performs much better than algorithms BF-BT and BT-BF. This is due to three reasons. Firstly, the performance of heuristic BF search is at its best when its guiding heuristic is more informed, and this usually happens at the bottom of the search graph [14]. Thus, algorithm 3 which employs the A* search (BF search) locally in all stages of the search process greatly reduces the computation complexity compared with the BF-BT search which only employs the A* search at the top of the search graph. Secondly, the important decisions with respect to the quality of a schedule may happen at the early stages of the scheduling activity [8]. So algorithm 3 can more greatly decrease the likelihood of missing the critical candidates than the BT-BF search that employs the BT search instead of the A* search at the early stage. Thirdly, algorithm 3 adopts an improved checking method for previous generated markings and it further decreases the likelihood of rejecting the critical markings.When $M_{\max}=1$, algorithm 3 not only will not degenerate to the BT method, but can still outperform the BT method. This is due to the fact that the two algorithms adopt different policies to node generation. The BT method adopts the last-in-first-out policy to node generation. When a marking is first selected for expansion, only one of its enabled transitions is chosen to fire and thus only one of its successor markings is generated. This newly generated marking is again submitted for expansion. However, in algorithm 3 with $M_{\max}=1$, after a marking has been expanded, the best marking of list OPEN is chosen for the next expansion. Thus, the quality of markings (even the final marking represents the solution found) expanded in each step of algorithm 3 with $M_{\max}=1$ is better than that of the BT method.

5 Application to more complex cases

In this section, we test algorithm 3 with more complex problems which have such characteristics as (1) buffers with limited sizes, (2) jobs with alternative routings, and (3) operations with multiple resources. These characteristics are illustrated in Fig. 7 where $O_{i,\,j,\,k}$ represents the *j*th operation of the *i*th job type being processed with the *k*th resource. When the buffer size is finite, the model can be modified as shown in Fig. 7a. The buffer size is represented by the number of tokens, e.g., the buffer size of the model shown in Fig. 7a is three. In Fig. 7b, the *j*th operation of job *i* can be performed by alternative routings, i.e., by using either resource *k* or resource *r*. In Fig. 7c, the performance of the operation needs multiple resources, resource *k* and resource *r*.

We generated a set of 40 random problems to test the algorithm. These problems, which were generated by the method of randomly selecting and linking predefined Petri net modules [11], had the following characteristics. The system had three resources and four different jobs with three operations each. Seventy-five percent of jobs had two alternative routings and 40% of operations had dual resources. Each operation was assigned a random cost from a uniform distribution (one to 100). The size of each buffer was limited to one to three and the lot size of each job was one to three, too.

We solved the same problem set using algorithm 3 with values of $M_{\max}$ of [1, 5, 10, 15…60]. And we compare these with the results obtained when $M_{\max}$ is set to infinity (pure A* strategy). The scheduling results are summarized in Fig. 8. The uppermost curve represents the mean percentage of optimality lost (RDms), the middle curve represents the mean percentage of search effort reduced (RDGM), and the lowest curve represents the mean percentage of computational time reduced (RDtime, which is the comparison of computational time and is equal to RDtime $= \frac{\mathrm{A*-Hybrid}}{\mathrm{A*}} \times 100\%$). We can see that algorithm 3 can greatly reduce the computational complexity (number of generated markings and computational time), and the mean optimality lost is substantially low, e.g., for $M_{\max}=30$, by average, it explores 33% of nodes less and it executes two times faster than for $M_{\max}=+\infty$, but the RDms is only around 3%.

6 Conclusions

This paper investigates a hybrid scheduling strategy in a Petri net framework. Timed Petri nets provide an efficient method for representing concurrent activities, shared resources, and precedence constraints encountered frequently in FMS. We use a hybrid heuristic algorithm that performs A* locally and BT globally in the Petri net reachability graph to search for a near-optimal schedule of a simple manufacturing system with different sets of lot sizes. In order to decrease the likelihood of rejecting the critical markings, an improved checking method for previous generated marking is also applied in the algorithm. Finally, the algorithm is used for a set of more complex FMS with limited buffer sizes, alternative routings, and dual resources.

Further work will be conducted in setting different performance indices such as minimization of tardiness. The efficiency of the heuristic functions in FMS will also be investigated.

Acknowledgements The research described in this paper is supported by a grant from the doctoral fund of the Ministry of Education of China (Grant No. 20050288015) and the science innovation fund of NUST. The authors want to thank Dr. Huanxin Henry Xiong at Lucent Technologies for his helpful suggestions in this research.

References

1. Tzafestas S, Triantafyllakis A (1993) Deterministic scheduling in computing and manufacturing systems: a survey of models and algorithms. Math Comput Simul 35:397–434
2. Lee DY, Dicesare F (1994) Scheduling FMS using Petri nets and heuristic search. IEEE Trans Robot Autom 10:123–132
3. Tuncel G, Bayhan GM (2007) Applications of Petri nets in production scheduling: a review. Int J Adv Manuf Technol 34(7–8):762–773
4. Yu H, Reyes A, Cang S, Lloyd S (2003) Combined Petri net modelling and AI based heuristic hybrid search for flexible manufacturing systems—part I: Petri net modelling and heuristic search. Comput Ind Eng 44:527–543
5. Russell S, Norvig P (1995) Artificial intelligence: a modern approach. Prentice-Hall, Upper Saddle River
6. Jeng MD, Chen SC (1998) A heuristic search approach using approximate solutions to Petri net state equations for scheduling flexible manufacturing systems. Int J Flex Manuf Syst 10:139–162
7. Yim SJ, Lee DY (1996) Multiple objective scheduling for flexible manufacturing systems using Petri nets and heuristic search. In: Proceeding of the IEEE International Conference on Systems, Man and Cybernetics: Information Intelligence and Systems, Beijing, China, pp 2984–2989
8. Xiong HH, Zhou MC (1998) Scheduling of semiconductor test facility via Petri nets and hybrid heuristic search. IEEE Trans Semicond Manuf 11(3):384–393
9. Reyes A, Yu H, Kelleher G, Lloyd S (2002) Integrating Petri nets and hybrid heuristic search for the scheduling of FMS. Comput Ind 47 (1):123–138
10. Inaba A, Fujiwara F, Suzuki T, Okuma S (1998) Timed Petri net-based scheduling for mechanical assembly-integration of planning and scheduling. IEICE Trans Fundam Electron Commun Comput Sci E81-A(4):615–625
11. Mejia G (2002) An intelligent agent-based architecture for flexible manufacturing systems having error recovery capability. Ph.D. thesis, University of Lehigh, USA
12. Huang B, Sun Y, Sun YM (2008) Scheduling of flexible manufacturing systems based on Petri nets and hybrid heuristic search. Int J Prod Res 46(16):4553–4565
13. Zeng QL, Wan LR (2007) Modeling and analysis for a kind of flexible manufacturing system involving time factors. Int J Adv Manuf Technol 34(3–4):346–352
14. Pearl J (1984) Heuristics: intelligent search strategies for computer problem solving. Addison-Wesley, USA
15. Huang B, Sun YM (2005) Improved methods for scheduling flexible manufacturing systems based on Petri nets and heuristic search. J Contr Theor Appl 2:139–144