Job Shop Scheduling In Manufacturing: An Approach Using Petri Nets and Heuristic Search

Gonzalo Mejía
Universidad de los Andes
Bagota, Columbia

Nicholas G. Odrey
Lehigh University

Report No. 04T-007

JOB SHOP SCHEDULING IN MANUFACTURING: AN APPROACH USING PETRI NETS AND HEURISTIC SEARCH

by

GONZALO MEJÍA, Assistant Professor, Department of Industrial Engineering, Universidad de los Andes, Bagota, Columbia

NICHOLAS G. ODREY, Professor, Department of Industrial and Systems Engineering, Lehigh University , Bethlehem, PA, 18015

ABSTRACT. Petri Nets have been extensively used to model different types of manufacturing systems due to their power to capture the complex characteristics of many of such systems. However, the "state explosion" and the NP-hard nature of many scheduling problems have prevented their use for optimization. In this paper, we propose an algorithm that combines the A* Search with an aggressive node pruning to intelligently search the reachability graph of a Petri Net. The primary purpose of the proposed algorithm is to reduce the search on the state space without compromising much the solution quality. Computational tests were conducted on both randomly generated and classical job shop problems. In addition, the performance of our algorithm is compared against the Beam Search and the Shifting Bottleneck algorithms a benchmark comparison with other methods,. The obtained results suggest that the Petri Net approach can be a powerful tool for scheduling and control of manufacturing systems.

Keywords: Petri Nets; A* Search; Beam Search; Job Shop Scheduling; Manufacturing Systems.

1. Introduction

Petri Nets have been extensively used to model a number of manufacturing systems due to their capability to model the asynchronous, concurrent non-deterministic nature of such systems. A very good survey on the topic modeling with Petri Nets was carried out by Moore and Gupta (1996). Petri Nets, however, suffer from "state explosion" which has prevented their application for optimization (i.e. planning and scheduling in the manufacturing systems field). Recall that a Petri Net is a simplified representation of a graph (the reachability graph) and such a graph can be huge even for small nets. Hence, performing an exhaustive search on the reachability graph is only feasible in very limited instances.

In order to reduce the effort required to search the reachability graph, past approaches have used an adaptation of the common A* search algorithm. The A* search algorithm for Petri Nets is a branch and bound-like algorithm which expands only the most promising nodes of the Petri Net reachability graph. Seminal papers date from 1994 (Lee and DiCesare, 1994). Since then, other authors have presented improved versions of the 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 makespan and Reyes-Moro et al (2002) presented a staged search approach combined with a pruning strategy to reduce the search space. These newer versions of the algorithm have shown good results when compared against the performance of the original algorithm developed by Lee and DiCesare (1994). However, there are two major issues that are have not received much attention: (i) How the A* search algorithm performs on standard test problems with known optimal solutions, and (ii) how A* search derived algorithms on Petri Nets compares against other scheduling techniques.

This paper extends the work by the authors (Mejía and Odrey, 2003). In this paper, our version of the A* algorithm (named as the Beam A* Search (BAS)) is presented. The BAS algorithm incorporates a number of features that improve both the performance and speed of the basic algorithm introduced by Lee and DiCesare (1994). Such features, described below, include selective pruning (within a beam, as explained later in this document), limited expansion, and controlled search deepening. The experiments that validate our approach were carried out in two parts: First, the performance of the BAS algorithm was compared against an exhaustive search method to investigate both the reduction of the generated state space and the behavior of the solution quality. The computational results show that our algorithm reduces drastically the search space while producing near-optimal solutions. In the second part, the BAS algorithm was tested on classical job shop problems with known optimal solutions and the results are compared against other similar algorithms. The obtained results show that the BAS algorithm performs competitively both in CPU times and solution quality against other methods. In this way, we conclude that heuristic search methods combined with Petri Nets can be a very powerful tool for scheduling and control for a wide variety of manufacturing systems. We emphasize that our approach is not intended to be another job shop scheduling algorithm. The job shop problems are only used as a benchmark. The virtue of this approach is the handling of the complex features of Flexible Manufacturing Systems. These include alternative routings, dual resources, buffers of finite capacity, material handling machines, re-circulation, batch sizes, etc.

This paper is organized as follows: First, in section 2, concepts of modeling with Petri nets are introduced. Next in section 3, time-space state equations to track the net evolution are presented. In section 4, the BAS algorithm is described in detail. Section 5 presents the results of our experiments. Conclusions and further research are discussed in the last section.

2. Modeling With Petri Nets

A Petri Net, as defined by Desrochers and Al-Jaar, (1995) is a bipartite directed graph having two types of nodes, namely places and transitions. Places are generally used to represent actions and/or conditions and transitions serve to model events. Additional entities named tokens reside in the places and represent the truth of the condition associated with the specific place. Tokens are moved between places by effect of transition "firings". Arcs between places and transitions define the direction of the token flow. Desrochers and Al-Jaar (1995) formally define a Petri Net as a 6-tuple $\{P, T, I, O, M_0, W\}$ in which P is a set of places, T is a set of transitions, $P \cap T = \varnothing$, $I = \{P \times T\}$ is a set of input arcs from places to transitions, $O = \{T \times P\}$ is a set of output arcs from transitions to places, M_0 is the initial marking which represents the number of tokens in each place at the initial state and W is a set of weights for each arc. The weight w of an input arc $(p \times t) \in \{P \times T\}$ represents the number of tokens that are required in place p to enable the transition t. Similarly, the weight w of an output arc $(t \times p) \in \{T \times P\}$ represents the number of tokens that are placed in place p after transition t fires. Without loss of generality, all weights $w \in W$ are set to 1 in this research. In the remainder of the document, the set of weights is dropped from the definitions

Definition. Timed Petri Net (TPN): A TPN is a 6-tuple $\{P, T, I, O, M_0, \tau\}$ where P, T, I, O and M_0 are defined as before and τ is the set of time delays associated with places.

Definitions. Source and Sink Places: A Source Place is a place with no incoming arcs. A Sink Place is a place with no outgoing arcs.

Definition. Path: A Path is a sequence of nodes $(x_1, x_2 \ldots x_N)$ (places or transitions) such that there exists a directed arc between two consecutive nodes (x_i, x_{i+1}) i = 1...N-1. The definition implies that the arc connects a transition to a place or vice versa. This definition was taken from (Zhou and DiCesare, 1994).

Definition. Sequence of Activities (SA). A SA is a Petri subnet that is intended to model the routing of a job or a part (the distinction depends on the subject of the representation). Here on the SA will be related to jobs. A SA accounts for precedence constraints, alternative routings and timing of operations. Each SA has a unique Source Place and a unique Sink Place. Places that belong to a SA are denoted as operational places and represent either conditions (e.g. part finished) or actions (e.g. part being processed). Places representing availability of resources are not accounted for in the definition of the SA. Transitions represent the beginning or the end of an action. Formally:

- A SA is an acyclic and connected state machine. In a state machine, every transition has only one input and one output arc.
- A SA contains no cycles. No path in the net can have duplicated nodes.
- All places have at least one input and one output arc except the Source Place (no incoming arcs) and the Sink Place (no outgoing arcs).

Generally, any job operation is modeled with a "precondition" place, a timed "action" place and a "post-condition" place. The precondition place represents (when marked) a job waiting in queue; the action place represents the execution of the operation and the post-condition place represents the successful termination of the operation. Transitions in all cases represent the start or end of an operation. More complex situations such as alternative routings can be also represented. For example, consider a job having the following routing as shown in table 1.

Table 1. Example of Job Routing

Operation	1	2	3
Resource	M_1	M_2 or M_3	M_4

The SA subnet for this job is represented as shown in figure 1. Table 2 shows the description for each place. Transitions represent the start or the end of an operation. A goal of the tokens residing at the operational places would be traversing the SA subnet from the corresponding Source Place to the Sink Place.

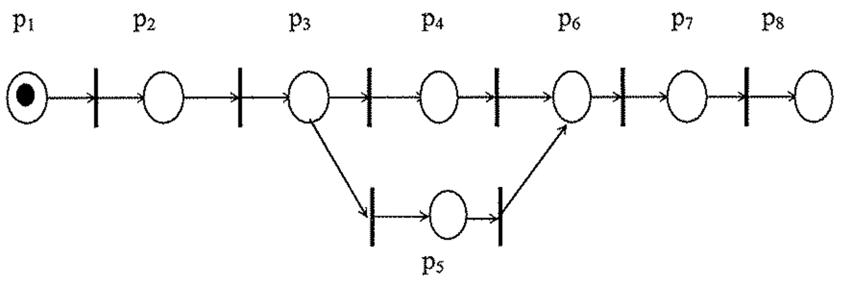


Figure 1. An example Sequence of Activities (SA) for the routing of a job.

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

p_1	Job ready	p_4	Job being processed by machine 2	p_7	Job being processed by machine 4
p_2	Job being processed by machine 1	p_5	Job being processed by machine 3	p_8	Job Finished
p_3	Operation 1 finished	p_6	Operation 2 finished		

Definition. Set of Sequences of Activities (SSA). A SSA is the Petri Net formed by as many SAs as jobs there exist in the system. A SSA consists of one or more unconnected SAs. A SSA represents the routing of the activities of all current jobs that have been allocated to a manufacturing system.

Definition. Set of Sequences of Activities with Resources (SSAR). A SSAR represents the addition of resources to SSA. The addition of resources implies addition of resource places which represent the availability of a resource, and arcs from/to the resource places to/from the transitions of the SSA. The number of resource 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" (i.e. tokens are removed from all the transition input places and put into its output places). 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. Additional provisions must be taken to ensure the "proper" allocation and release of the resources. Such provisions include (i) verifying a resource cannot be allocated twice before being released and (ii) guaranteeing that for any outgoing arc from a resource place, there must be an incoming arc to the resource place, as well. Tokens residing originally at the resource places are denoted as resource tokens. The concept of SSAR completes the modeling of a manufacturing system. Figure 2 illustrates a Petri Net model of a manufacturing system with two jobs to be processed and two machines. Job 1 is first processed by machine 1 and then by machine 2. Job 2 is processed first by machine 2 and then by machine 1. The machines are released as soon as a job finishes its processing on such machines. The highlighted places (p_2, p_4,p_7 and p_9) represent timed places and correspond to operations. The description of places follow in table 3.

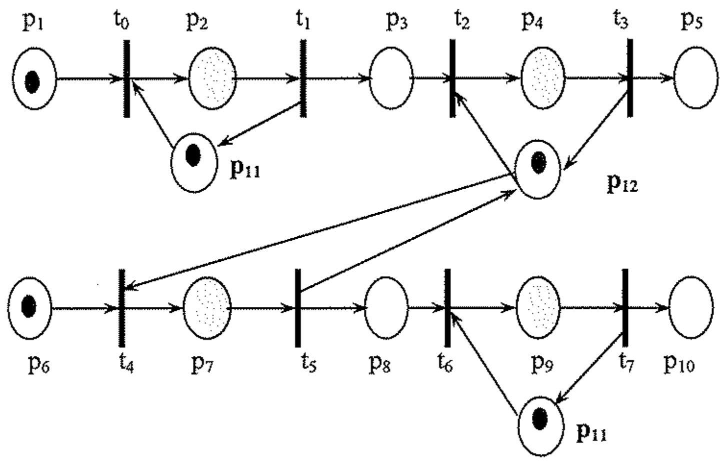


Figure 2. Petri Net model of a job shop with two jobs and two machines

Table 3. Description of places for the net in figure 2

Place Description for Petri Net of figure 2			
p_1	Job 1 ready	p_6	Job 2 ready
p_2	Job 1 being processed by machine 1	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_7	Job 2 being processed by machine 1
p_5	Operation 2 of Job 1 finished	p_8	Operation 2 of Job 2 finished
p_{11}	Machine 1 available	p_{12}	Machine 2 available

3. State Equations

The state equations presented here were the subject of the previous work (Liu et al (1997)) on Color Petri Nets. Here these equations are adapted to SSAR nets. The state equations serve to track both the token evolution and the net timing. The main feature is the augmentation of the marking vector with the remaining time vector. For the following definitions, consider a SSAR net having *n* places and *m* transitions.

Definition. Remaining Process Time vector (M_r): This *n* vector contains in its *i-th* position the remaining time required to enable output transitions of the *i-th* place. The remaining process time corresponding to condition places and to resource places is always set to zero (0). Here, we constrain the timed places to accept at most one token to avoid duplicated remaining process times in one single place. This constraint is easily met by allowing at most one token in each resource place that intervenes in an operation.

The remaining process time vector is updated with every transition firing. Here we assume that during a transition firing, tokens are removed first from input places and then placed to the output places. Here, two more definitions are required: M^+ and M_r^+ are respectively the marking and remaining processing time vectors after the token removal (from input places) but before the token placement (in the output places).

Definition. Time elapse between transition firings $\delta(k)$: This scalar is calculated as the maximum of the remaining processing times of the places that are input to the firing transition (say t^*) at the *k-th* event.

$$\delta(k) = \text{Max}\ \{ M_r(k)_i \}\ \forall i \text{ such that the place } p_i \in \bullet t^* \qquad (1)$$

$M_r(k)_i$ is the *i-th* position of the $M_r(k)$ vector.

The augmented state space representation can be written as:

$$X(k+1) = A(k)\,X(k) + B(k)u(k) \qquad (2)$$

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.

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

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

I_n: Identity $n \times n$ matrix.

0_n: Zero $n \times n$ matrix.

Q: Diagonal $n \times n$ matrix that serves to distinguish operational places from resource places.

$Q = \{q_{ij}\}$. $i = 1 \ldots n, j = 1 \ldots n$. $q_{ij} = 1$ when $i{=}j \wedge i$ is timed place. $q_{ij} = 0$ everywhere else.

u(k): $m \times 1$ Control vector that determines which transition fires after *k* firings. $u_j(k)$ is the *j-th* position of *u* at time *k*. $u_j(k) = 1$ if transition *j* fires, 0 if it does not.

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. $C = C^+ - C^-$. See the definition of incidence matrix in Murata(1989).

C^+ and C^- are, respectively, the incidence input and output matrices.

$T = \{t_{ij}\}$: Processing time $n \times n$ diagonal matrix for operational places. $t_{ij} = \tau_i$ when $i{=}j$; 0 otherwise. τ_i is the *i-th* position of vector of time delays associated with places as defined in section 2.

Separating the marking and remaining processing time vectors, the equations will be respectively:

$$M(k+1) = M(k) + Cu(k) \tag{3}$$

$$M_r(k+1) = M_r(k) - \delta(k)QM(k) + TC^+u(k) \tag{4}$$

Equations 3.1 and 3.2 require a two step calculation in order to calculate the value of $\delta(k)$:

$$M(k)^+ = M(k) - C^-u(k) \tag{5}$$

$$M_r(k)^+ = M_r(k) - \delta(k)QM(k) \tag{6}$$

and

$$M(k+1) = M(k)^{+} + C^{+} u(k) \tag{7}$$

$$M_r(k+1) = M_r(k)^{+} + TC^{+}u(k) \tag{8}$$

Intuitively, the remaining process time is calculated as follows: Suppose that the firing of the j^{th} transition corresponds to the k-th event. The j^{th} transition will fire after $\delta(k)$ units counted from the last transition firing. Thus, the remaining process time vector is recalculated by subtracting $\delta(k)$ time units from the vector $M_r(k)$ (for those marked places). In the state space equations this is represented by $M_r(k)^{+} = M_r(k) - \delta(k)QM(k)$ as in equation 6 . Negative values of the $M_r(k)^{+}$ vector are set to zero (0), meaning that the corresponding remaining process times have been exhausted. Then the j^{th} transition fires and the corresponding tokens are moved from the input places to the transition output places. The remaining process time for the output places of the firing transition are the time delays since the corresponding operations 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 $M_r(k+1) = M_r(k)^{+} + TC^{+}u(k)$ as in equation 8.

Numerical Example: Consider the Petri net shown in figure 2. The Petri Net represents two jobs to be processed by two machines. The Sequence of Activities (SA) of job 1 consists of the path = $\{p_1, t_0, p_2, t_1, p_3, t_2, p_4, t_3, p_5\}$. Likewise the SA of job2 consists of the path = $\{p_6, t_4, p_7, t_5, p_8, t_6, p_9, t_7, p_{10}\}$. In the example, the time delay vector is set to:

$$\tau^{T} = [0, 3, 0, 4, 0, 0, 2, 0, 5, 0, 0, 0]$$

In this example$\tau(p_0)$=0, $\tau(p_1)$=3, $\tau(p_2)$=0, $\tau(p_3)$= 4 and so on. Notice that time delays associated with resource places and condition places are 0.

For this net, the matrices Q and T are respectively:

$$Q=\begin{vmatrix}
0&0&0&0&0&0&0&0&0&0&0&0\\
0&1&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&1&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&1&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&1&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0
\end{vmatrix}
\qquad
T=\begin{vmatrix}
0&0&0&0&0&0&0&0&0&0&0&0\\
0&3&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&4&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&2&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&5&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0
\end{vmatrix}$$

The state at k=0 is given by: (The ^{T} indicates the transpose of the vector).

$$M(0)^T = M_0^T = [1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1]$$

$$M_r(0)^T = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]$$

The initial $M_r(0)$ vector is 0 since the time delays for all marked places is 0. Notice that for the initial marking, all marked places represent conditions (e.g. part available, machine available).

Assume that the net will be executed according to the valid transition firing sequence $\sigma = \{t_0, t_4, t_5, t_1\}$. Hence t_0 fires at k=0. The control vector $u(0)$ corresponding to the firing of t_0 would be:

$$u(0)^T = [\ 1, 0, 0, 0, 0, 0, 0, 0]$$

The token evolution changes according to the equation 5.

$$M(0)^+ = M(0) - C^- u(0) \ \text{ where } M(0) = M_0$$

$$M(0)^+ = [0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1]$$

The marking vector $M(0)^+$ is calculated with equation 4.1.

The time between transition firings (δ (0))at k = 0 is calculated as in equation 1:

$$\delta\ (k) = \text{Max}\ \{M_r(k)_i\}\ \forall i\ /\ p_i \in\ \bullet t_0$$

The set of input places of transition t_0 is: $\bullet t_0 = \{p_0, p_{11}\}$. Thus = δ (0) = max $\{M_r(0)_0\ , M_r(0)_{11}\}$. Since $M_r(0)_0$ and $M_r(0)_{11}$ are both 0, the value of δ (0) is 0.

From equation 4.2 at k = 0.

$$M_r(0)^+ = M_r(0) - \delta\ (0)\ Q\ M(0) = M_r(0)$$

Recall that negative values of M_r are set to zero (0). $M(1)$ and $M_r(1)$ are calculated with equations 4.3 and 4.4:

The full $M(1)$ and $M_r(1)$ vectors are:

$$M(1)^T = [0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1]$$

$$M_r(1)^T = [0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]$$

The firing of transition t_4 results in the following vectors:

$$M(2)^T = [0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0]$$

$$M_r(2)^T = [0, 3, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0]$$

Next transition t_5 fires. The time between transition firings (δ (2)) at k = 2 is calculated as in equation 1:

The set of input places of transition t_5 is: $\bullet t_5 = \{p_7\}$. Thus = δ (2) = $M_r(2)_7$ = 2.

$$M_r(2)^{+\,T} = [0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]$$

And $M_r(3) = M_r(2)^+$. Notice that if transition t_1 had been fired instead of t_5, $\delta(2) = 3$, $M_r(2)^+{}_7$ would have been -1 but since negative values are not allowed, the value of $M_r(2)^+{}_7$ would be 0.

The marking and remaining processing time vectors after firing transition t_5 would be:

$$M(3)^T = [0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0]$$

$$M_r(3)^T = [0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]$$

Transition t_1 fires last. The resulting value of $\delta(3)$ would be 1 and the corresponding marking and processing time vectors would be:

$$M(4)^T = [0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1]$$

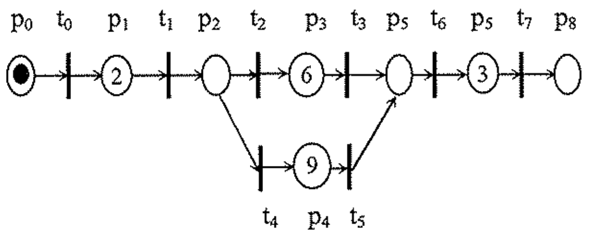
$$M_r(4)^T = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]$$

The total elapsed time is: $\sum_k \delta(k) = 3$.

Definition. Remaining Work Time (*RWT*) vector. This definition was originated in this research with the purpose of calculating estimates of the minimum time required to complete the remaining operations on a job. The *RWT* vector is a constant vector and contains in its *i-th* position the sum of time delays (associated with places), along the shortest elementary path, that a token located at *i-th* place would require to reach the sink place of the corresponding SA. The calculation of RWT_i excludes the remaining processing time at the *i-th* place itself since such remaining process time is a variable quantity. The *RWT* of all resource places is zero (0). An example of the *RWT* vector is illustrated with the figure 3. The net shows a single SA net with a token in p_0. Time delays are shown inside places (if no number is shown, the timed delay is 0 for such a place). This SA contains two possible paths containing both the start (p_0) and the Sink Place (p_7). The two paths correspond to the paths $\text{Path}_A = \{p_0, t_0, p_1, t_1, p_2, t_2, p_4, t_3, p_5, t_6, p_6, t_7, p_7\}$ and $\text{Path}_B = \{p_0, t_0, p_1, t_1, p_2, t_4, p_3, t_5, p_5, t_6, p_6, t_7, p_7\}$. We denote here $\tau(p_i)$ as the time delay associated with place i. The minimum time for a job token at p_0 to reach p_7 (the Sink Place) would be 11 ($\tau(p_1) + \tau(p_2) + \tau(p_3) + \tau(p_5) + \tau(p_6) + \tau(p_7)$) time units along P_B. RWT_0 is then 11 time units. A token at p1 would take 9 time units ($\tau(p_2) + \tau(p_3) + \tau(p_5) + \tau(p_6) + \tau(p_7)$) to reach p_7 plus the time required to enable t_1 (M_{r1}). Thus RWT_1 would be 9 since the remaining time M_{r1} is excluded from the calculation of RWT_1. Following the same reasoning, RWT_2 would be 9 time units. A token at p_3 (resp. p_4) would require 3 time units plus the remaining time at p_3 (resp. p_4). RWT_3 and RWT_4 both would be 3 time units ($\tau(p_5) + \tau(p_6) + \tau(p_7)$). The full *RWT* for the net shown in figure 3 would be:

$$RWT = [11, 9, 9, 3, 3, 3, 0, 0, 0, 0]$$

The *RWT* vector will be used to calculate lower bounds for completion times in the remainder of this paper.



p_0 t_0 p_1 t_1 p_2 t_2 p_3 t_3 p_5 t_6 p_5 t_7 p_8
2
6
3
9
t_4 p_4 t_5

Figure 3. Calculation of the *RWT* vector (See text).

4. Optimization using Heuristic Search

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 (Lee and DiCesare (1994); Xiong and Zhou (1998); Reyes-Moro et al (2002)). The A* search algorithm expands the most promising branches of a Petri 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 following is the A* search algorithm described in Lee and DiCesare (1994).

- Place the initial marking M_0 on the list OPEN
- If OPEN is empty, terminate with failure.
- Remove the first marking M from OPEN and put M on CLOSE
- If M is the final marking M_f, construct the path from M back to M_0 and terminate.
- Find the set of enabled transitions $\{t_j\}$ ($j = 1 \ldots et(M)$). $et(M)$ is number of enabled transitions for the marking M.
- Generate the successors M' that would result from firing each enabled transition t_j and calculate $g(M')$, $h(M')$, and $f(M')$.
- For each of the markings M' do the following:

- If M' is equal to some marking M^O already on OPEN, verify if $g(M') < g(M^O)$. If that is the case delete M^O from OPEN and insert M' on OPEN. The conditions $M' = M^O \wedge g(M') < g(M^O)$ indicate that a new better path was found between M_0 and M' (M^O).
- If M' is equal to a marking M^C already on CLOSE, verify if $g(M') < g(M^C)$. If that is the case delete M^C from CLOSE and all its successors that reside on OPEN and insert M' on CLOSE. The conditions $M' = M^c \wedge g(M') < g(M^c)$ indicate that a new better path was found between M_0 and M' (M^c).
- If M' is not on either list, then insert M' on OPEN.
- Go to step 2.

The reader is referred to Russell and Norvig (1995) for further details of this algorithm.

Condition 1: In order to guarantee that A* search finds an optimal solution, $h(M)$ must be greater or equal than 0 and equal or less than the actual value $h^*(M)$ for every marking M (Lee and DiCesare, 1994). 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 the optimal one (Russell and Norvig, 1995). In this research, the generation of the successor markings M' of M and, the calculation of the corresponding remaining processing time vector M_r' are accomplished with the state equations described in section 3 . The cost $g(M')$ is calculated as:

$$g(M') = g(\text{parent of } M') + \delta\ (\text{parent of } M'\text{-}M')$$

where δ (parent of M'-M') is the time elapse to reach $g(M')$ from the parent of $g(M')$. The cost $g(M_0)$ is 0.

Although, A* search guarantees optimality, four major drawbacks of the above A* search algorithm have been identified. Those relate to (i) the calculation of the heuristic function $h(M)$, (ii) the exponential complexity of the algorithm that results in marking explosion, (iii) the generation of non-promising markings, and (iv) the growth of the list OPEN (Sun et al, 1994). Each of these drawbacks and our corresponding corrections are explained next:

4.1 Calculations of the heuristic function *h(M)*

Heuristic functions $h(M)$ that meet the Condition 1 described above are generally difficult to estimate (Reyes-Moro et al, 2002) This is due in part to the lack of features of the Timed Petri Nets that facilitate the calculations.

In this research, two heuristics were developed: The first heuristic calculates a lower bound for the remaining time to process all jobs given a marking M. This first heuristic function employs the remaining process time M_r and the remaining work time RWT vectors introduced in the previous section. The idea is that the minimum time that a job (represented by a token located at place i) would require to reach the sink place of the corresponding SA is the time required to process the remaining operations. This is given by $E_i = RWT_i + Mr_i$. This calculation excludes the time a job token spends in queues waiting for resources to become available. Although unrealistic, the estimate E_i is a lower bound of the remaining time required for a job to be finished. The maximum of the E_i given that $M_i > 0$ and p_i is an operational place, is used as an estimate of $h(M)$. The estimate, denoted here as $h_a(M)$, can be expressed as:

$$h_a(M) = \max (E_i) \text{ such that } M_i > 0 \text{ and } p_i \text{ is an operational place} \qquad (10)$$

The second heuristic developed in this research to calculate the cost $h(M)$ calculates the remaining cost $h(M)$ as the cost of the path from the current marking to the final marking using the common Most Remaining Work Time (MRWT) dispatching rule. The dispatching rule MRWT was selected because it generally produces better results for the makespan case than other common dispatching rules such as Shortest Processing Time (SPT) and Longest Processing Time (LPT) (Morton and Pentico, 1996) Clearly this estimate is more realistic as compared to $h_a(M)$ because it is based on a feasible schedule. This heuristic function denoted here as $h_b(M)$ is given by the expression:

$$h_b(M) = h(\text{MRWT}, M) \qquad (11)$$

$h(\text{MRWT},M)$ is the remaining cost from the marking M to the final marking M_f employing the MRWT priority rule for transition firings. The MRWT rule is implemented here with the RWT vector. Other heuristic functions $h(M)$ based on dispatching rules are currently under investigation.

4.2 Avoiding Markings Explosion by Controlled Deepening

The second drawback mentioned earlier in this section is that using admissible heuristics $h(M) \leq h(M)^*$ (for all markings M, and $h(M)^*$ is the actual cost), will degenerate towards a Breadth-First Search strategy (Reyes-Moro et al (2002)). To overcome this drawback, the strategy followed in this research consists of forcing the search deeper in the reachability graph in a controlled manner. This is accomplished by expanding only a limited number of markings at each level of the reachability graph. The maximum number of markings allowed in a given level is denoted as the beam width (*bw*). The speed of the search will be primarily determined by the beam width *bw*: Smaller beam widths (*bw*) lead to faster solutions but the quality of the solution may be compromised because fewer markings are explored. In figure 4, a sketch of the Petri Net reachability graph is depicted showing the explored markings. Notice that only 2 markings at expanded at each level (beam width = 2).

In our algorithm, a maximum of *bw* markings on OPEN whose depth corresponds to the current depth, are moved to CLOSE. If no marking on OPEN is located at the current depth, then the first marking on OPEN, whose depth is greater than the current depth, is moved to CLOSE or, if no marking on OPEN has a greater depth than the current depth, the marking with higher depth is selected. In this way, we guarantee that the search always move towards the location of final marking.

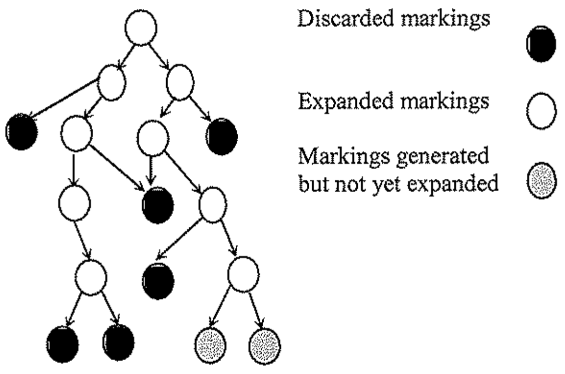


Figure 4. The Beam Width Approach for Marking Selection

4.3 Reducing the Search Space by Eliminating Non-Promising Markings

The third strategy to reduce the search space consists of selectively eliminating nodes. Here, we adapt the concept of "non-delay" scheduling (Baker (1974) to the Petri Net formalism. In a non-delay schedule, a machine is never idle when a job is waiting for processing. The idea here is to avoid sequences of transition firings that lead to schedules with delays. In the A* search terminology the non-delay scheme translates to the following: When generating the successors *M'* of a given marking *M*, only the marking(s) with the minimum value of *g(M')* would be inserted on the list OPEN. Successor markings with values of *g(M')* greater than min (*g(M')*) will be permanently deleted. In non-delay scheduling, the operations of the jobs with the earliest starting times are scheduled first. The values of *g(M)* correspond to the start of such operations.

4.4 Reducing the Length of the List OPEN

As pointed out by Sun et al (1994) the list OPEN can exponentially grow if no appropriate measures are taken. In this research, we adapt the strategy of Sun et al (1994) which limits the length of the list OPEN to a certain cutoff value. The strategy of Sun et al (1994) removes the last marking from OPEN (the marking *M* with highest value of *f(M)*). The potential drawback is that usually deeper markings will have greater values of *f(M)* compared to shallow markings. Hence "good" deeper markings can eventually be pruned off. In our strategy, when the length of the list OPEN exceeds the pre-defined cutoff value, the marking with the highest *f(M)*, whose depth is lower than the current depth is discarded.

4.5 The Beam A* Search Algorithm for Petri Net Optimization

All the proposed modifications presented in this research lead to the algorithm that we denote as Beam A* Search (BAS) algorithm. The prototype function for this algorithm is BAS (*h(M)*, *bw*, *cutoff*) where *h(M)* is the heuristic function used, *bw* is the width of the beam, and *cutoff* is the maximum length of the list OPEN as defined above. In this paper, the algorithm is sometimes termed as BAS (h_x) for simplicity and denotes the "BAS employing the heuristic h_x" regardless of the values of *bw* and *cutoff*.

The full BAS(*h(M)*, *bw*, *cutoff*) algorithm follows next:

- Place the initial marking M_0 on the list OPEN
- Initialize current_depth = 0. Also initialize the variable count_markings as 0. The variable count_markings tracks the number of markings in the current depth that have been expanded.
- If OPEN is empty, terminate with failure.
- 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.
- If *M* is the final marking M_f, construct the path from M_0 to *M* and terminate.
- Find the set of enabled transitions $\{t_j\}$ / $j \in \{1 \ldots m_e\}$ m_e is number of enabled transitions when the marking is *M*.
- Generate the children markings *M'* along with the corresponding remaining process time vectors M_r' that would result from firing each enabled transition t_j. This is accomplished with the state equations described in previous sections. Also calculate *g(M')* for each *M'*. Store these markings on a temporary list (TEMPLIST).
- (Non-delay branching). Find the minimum value of *g(M')* for all *M'*s in TEMPLIST, say min(*g(M')*). Delete those markings with values of *g(M')* greater than min(*g(M')*) from TEMPLIST. For each of the markings *M'* remaining on TEMPLIST do the following:
- Calculate *h(M')*, and *f(M')*.
- If *M'* is equal to some marking M^O already on OPEN, verify if $g(M') < g(M^O)$. If that is the case delete M^O from OPEN and insert *M'* on OPEN. If not, insert *M'* on OPEN. If the conditions $M' = M^O$, $g(M') <$

$g(M^O)$, a new better path was found from M_0 to M' ($M' = M^O$). Deleting M^O from OPEN and inserting M' redirects the path from M_0 to M'.

- If M' is equal to a marking M^C already on CLOSE, verify if $g(M') < g(M^C)$. If that is the case delete M^C from CLOSE and all its children that reside on OPEN and insert M' on CLOSE. If not, insert M' on CLOSE. This step follows the same logic as step 8a. If the conditions $M' = M^C$, $g(M') < g(M^C)$ are all true, a new better path was found from M_0 to M' ($M' = M^C$). Deleting M^C from CLOSE and inserting M' redirects the path from M_0 to M'. In addition, if children nodes of M^C were expanded, they must be deleted from OPEN.
- If M' is not on either list, then insert M' on OPEN.
- If the length of OPEN is greater than the pre-defined cutoff value, delete the marking on OPEN whose depth is smaller than current_depth and with the highest value of $h(M)$. If no marking has a depth smaller than current_depth, delete the last marking on OPEN.
- If count_markings < bw then add 1 to count_markings. Otherwise, set count_markings = 0 and current_depth = current_depth + 1.
- Go to step 3.

The following section illustrates the performance of the BAS algorithm.

5. Computational Experiments and Results

The BAS algorithm was coded in C++ and the experiments were run on a personal computer having a 1.8 GHz speed microprocessor and 256MB RAM memory. Two sets of test problems consisted were considered. The first set of problems was randomly generated and the second set of problems consisted of classical problems taken from the literature. The evaluation criterion in all cases was makespan.

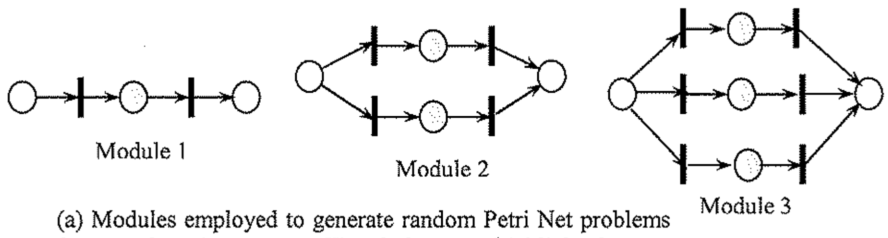
5.1 Random Problem Generation

The first set of experiments was conducted on a set of 30 randomly generated problems of approximately the same size (number of jobs and number of operations). Such tested problems consisted of 5 jobs, 3 or 4 machines, 4 or 5 operations per job, recirculation and approximately 33% of the operations had alternative routings. The purpose of these tests was to evaluate performance of the BAS algorithm on both state space reduction and solution quality. Any given problem was tested with both an exhaustive search method and the BAS algorithm. These problems were generated by randomly selecting and linking predefined Petri Net modules. A pre-defined module is intended to model a number of alternative routes to execute an operation of a job. The three predefined modules, denoted here as module1, module2 and module3, with one, two or three possible alternatives per operation respectively are shown in figure 5a. The SA of each job is constructed by linking the modules as shown in figure 5b. Next, the resources are added to each operation of the SA as shown in figure 5c with the restriction that no machine can be assigned twice to an operation. This process is repeated for the SA of each job.

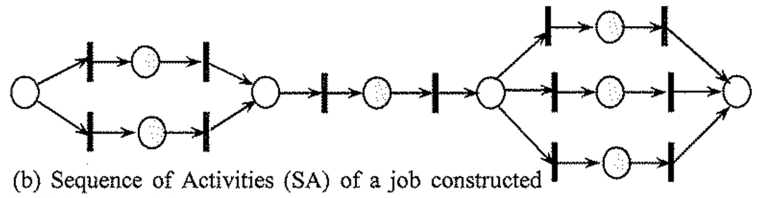
A detailed description of the random problem generation is described next:

- Set the desired number of jobs, and the number of operations per job and the number of machines (or resources) in the system.
- Create en "empty" job subnet that later will contain partial Sequences of Activities (SA) of a job.
- For a given job:
- Randomly select one of the modules (module1, module2 or module3).
- Randomly assign time delays to the timed operational places of the module (highlighted in figure 5a).
- Link the selected module to the existing job subnet (See figure 5b). If this is the first module to be selected, the module is added to the empty job subnet.
- Repeat steps 3a, 3b and 3c for as many times as the number of operations of the job.
- Repeat steps 2 and 3 *nj* times where *nj* is the number of jobs.
- Create as many resource places as resources were predefined.
- Add resources to the operations (represented as timed places) of a job as follows:
- Randomly select a resource place.
- Link the selected resource place to an "uncovered" timed place in the SA of the job. An uncovered timed place represents an operation for which no resources have been assigned. Linking a resource place to a timed place is accomplished by adding an arc from the resource place to the input transition of the timed place and an arc from the output transition of the timed place to the resource place.
- Repeat steps 6a and 6b until all timed places have been covered.
- Repeat step 6 until all jobs have been assigned resources.
- Add tokens to each resource place and to each Source Place. The initial number of tokens in the resource places is set to one (This will guarantee that no more than one token at a time can reside in a timed place which is a condition for the net construction. See figure 5c for an example.

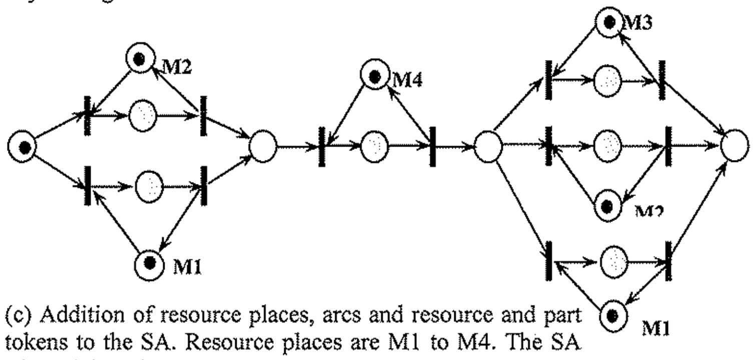
- Calculate the Petri Net incidence matrix C, the initial and final marking vectors (M_0 and M_f), the Remaining Work Time (RWT) vector, and the time delay vector τ.



(a) Modules employed to generate random Petri Net problems for manufacturing systems. Timed places are highlighted



(b) Sequence of Activities (SA) of a job constructed by linking modules 2-1-3



(c) Addition of resource places, arcs and resource and part tokens to the SA. Resource places are M1 to M4. The SA of one job is shown.

Figure 5. Steps for Random Problem Generation

5.2 Results on Randomly Generated Problems

As mentioned above, we compare the BAS solutions against the solutions resulting from an exhaustive search method on the Petri Net reachability graph (denoted here on as Exhaustive Search). The first indicator is the average relative deviation, compared to the Exhaustive Search solution, of the 30 problems.

The relative deviation for a problem is defined as:

$$\frac{\textit{BAS solution} - \textit{Exhaustive Search solution}}{\textit{Exhasutive Search solution}} \times 100\% \tag{12}$$

The second indicator is the ratio of markings expanded by BAS vs. the number of markings expanded by Exhaustive Search. The third and last indicator is the ratio of the CPU time of the BAS algorithm vs. the CPU time of Exhaustive Search. The charts in the figures 6, 7 and 8 illustrate the performance of the algorithm using both heuristic functions BAS (h_a) and BAS (h_b). Figure 6 shows the Average Relative Deviation (ARD) chart which is the average relative deviation for the 30 problems as a function of the beam width (*bw*). Figure 7 corresponds to the Average Ratio of Explored Markings (AREM) chart which shows the average ratio of markings explored by BAS and Exhaustive Search vs. the beam width. Figure 8 is the Average Ratio of CPU Times (ARCT) chart. This chart shows the average ratio of CPU times vs. the beam width. The maximum beam width was set to 7 because no more than 7 transitions were enabled at any given time for any of these problems. Beam widths greater than 7 would produce expansion of markings deeper than "current_depth" in the reachability graph and eventually large beam widths will degenerate into depth-first strategies.

Figure 6. The ARD chart

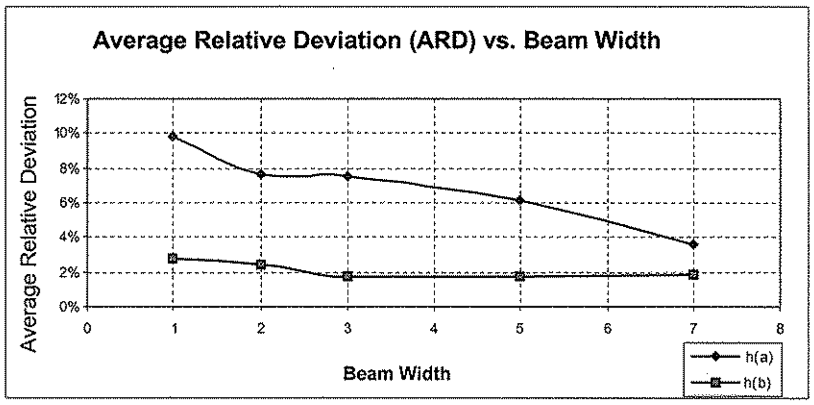


Figure 7. AREM Chart.

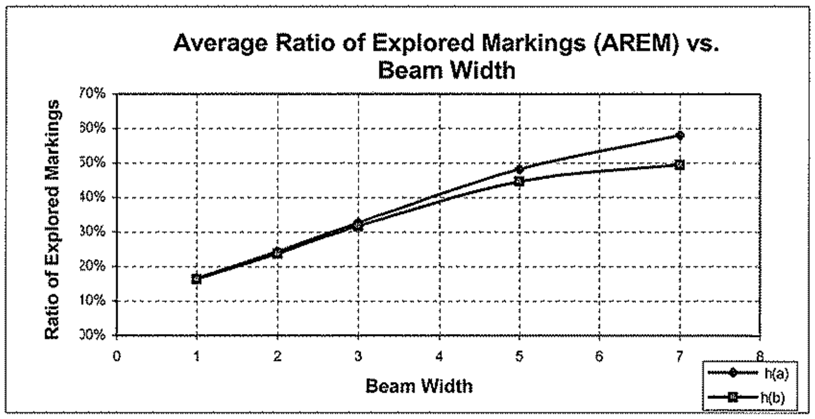
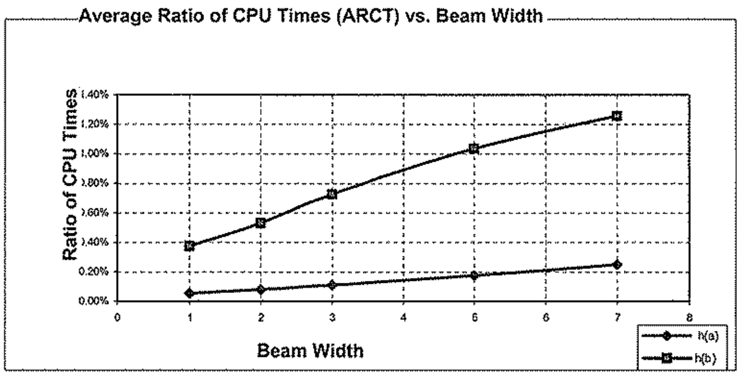


Figure 8. ARCT Chart.



Analysis. The above charts show that on average, results very close to the Exhaustive Search solution can be achieved in a very small fraction of time and exploring only a small portion of the markings. For example, the average relative deviation for BAS (h_a, *bw*= 7, *cutoff*= 100) was 4.0% (See ARD chart), employing on average approximately 0.2% of the time (this is the value of average ratio of CPU time for BAS(h_a, *bw*=7, *cutoff*= 100) in the ARCT chart) and exploring approximately 0.6% of the markings (see the AREM chart) compared to Exhaustive Search. Similar conclusions can be drawn from the curves for BAS using the h_b heuristic function.

Using the h_a heuristic with small beam widths, relatively important gains in terms of performance were achieved with changes on the beam width. For instance, the average relative deviation for BAS (h_a, *bw* = 1, *cutoff*=100) was approximately 10% whereas the average relative deviation for BAS (h_a, *bw* = 7, *cutoff=100*) was approximately 4% (See ARD chart). The curves for BAS using the h_b heuristic show that solutions close to the optimal were achieved using small beam widths. The extra-effort of exploring more markings (by using larger beam widths) did not produce significant improvements in terms of solution quality (See the ARD and AREM charts for the h_b heuristic).

On average, BAS (h_b) achieved better results than BAS (h_a) given the same value of beam width (See the ARD chart). Notice, however, that BAS (h_a) required significantly less CPU time than BAS (h_b) for the same beam width. The extra-time required by BAS (h_b) is explained by the more complex calculation of the remaining function $h_b(M)$ with the MRWT dispatching rule.

5.3 Results on Standard Job Shop Scheduling Problems

The literature has produced scores of papers on Job Shop Scheduling Problems (Pinedo and Chao, 1999) and it is not the purpose of this research to develop yet another algorithm for this type of problem. Rather, we intend to show the feasibility of Petri Nets as an optimization tool. The choice of testing our algorithm on classical job shop problems corresponds to the wide availability of known optimal solutions and the results of other algorithms that yield optimal or near-optimal solutions. These problems will provide a benchmark for the performance of the BAS algorithm.

Problem Definition.

As it is common in the literature a N x M job shop problem represents a job shop problem having N jobs and M machines The job shop scheduling problem can be defined as follows: A number of jobs (N) are to be processed by a number of machines (M). The processing of a job on a machine is called an operation. Each job follows a fixed sequence of operations and a job visits a machine only once. The processing times are deterministic and no pre-emption is allowed. The objective of a schedule is to minimize some criterion, subject to (i) precedence constraints, and (ii) resource constraints (a machine can only process a job at a time). Let *(i,j)* the operation of the job *i* on machine *j*; p_{ij} the duration of the operation *(i,j)* and y_{ij} the start time of operation *(i,j)*. In this paper, the criterion is makespan (*Cmax*) which is the maximum of the completion times of all jobs (*Cmax* = max (C_j) for all *j*).. This scheduling problem can be formulated as follows (Pinedo and Chao, 1999).

Min *Cmax*

Subject to

$y_{kj} - y_{ij} \geq p_{ij}$ for every operation *(i,j)* that precedes *(k,j)*

$Cmax - y_{ij} \geq p_{ij}$ for all operations *(i,j)*

$y_{ij} - y_{il} \geq p_{il}$ or $y_{il} - y_{ij} \geq p_{ij}$ for all operations *(i,l)* and *(i,j)* to be processed on machine i

$y_{ij} \geq 0$ for all operations *(i,j)*

Solving this problem to optimality using mathematical programming is only feasible for small instances. For bigger sized problems, most commonly used techniques include graph-based algorithms such as the Shifting Bottleneck algorithm (Adams et al, 1988) and Beam Search (Sabuncuoglu, and Bayiz, 1999); Neighborhood search algorithms such as Tabu Search (Glover, 1990), Simulated Annealing (Hussain and Joshi, 1997), and Genetic Algorithms (Della Croce et al, 1995)

In this paper, 54 standard job shops problems were tested and the results compared against the optimal solutions, and the results of the Beam Search and Shifting Bottleneck algorithms provided by Sabuncuoglu, and Bayiz (1999) and Adams et al (1998) respectively. In all cases, the proposed job shop test problems were modeled with a SSAR nets. Among these problems, MT06 and MT10 were developed by Muth and Thompson (1963), ABZ5 and ABZ6 were originated by Adams et al (1987), LA01-LA40 were presented by Lawrence (1994) and ORB1-ORB5 were also taken from Applegate and Cook (1991) Based on the results found in the previous section in which the behavior of BAS (h_a) and BAS (h_b) was investigated, we selected the range of beam widths. For BAS(h_a) algorithm the beam width was varied from 10 to 15. The BAS(h_b) algorithm was used with beam widths of 1 and 2. A summary of the results of BAS with both the heuristics h_a and h_b is shown in table 4. The comparison with the results from the Beam Search (BS) and the Shifting Bottleneck (SB) algorithms are also shown in table 4. The BS algorithm is similar to the BAS algorithm in the sense that both algorithms are graph-based. The biggest difference is that each node in BS corresponds to an operation whereas in BAS, a node in the reachability graph corresponds to a transition firing. Other differences include the branching schemes and the node filtering of BS. The SB algorithm was developed by Adams et al (1988) and since its introduction has been widely popular (Pinedo and Chao, 1999). It is based on the common disjunctive graph representation of a job shop and on finding the optimal solution of M single machine problems.

Results and Analysis

The results in table 4 illustrate the feasibility of applying A* search-based algorithms for Petri Nets to job shop scheduling problems. On the suggested problems, BAS(h_a) found solutions within 7.38% of the optimal solution on average, with 9 optimal solutions and 23 results within 10%. BAS(h_b) found solutions on average within 3.78% of the optimal, with 13 optimal solutions and 35 solutions within 10%. BAS(h_b) performed better compared to BAS(h_a) in 28 out of 54 cases (in 9 cases both algorithms found the same solution). Overall, these results suggest that using known properties (e.g. non-delay scheduling) and proved rules for job-shop scheduling (e.g. the MRWT rule to calculate estimates of remaining times in the case of BAS(h_b)) provide better results than using simple lower bounds for the estimation of the remaining cost $h(M)$ as in the case of BAS(h_a).

The results also show that BAS(h_b) performed competitively than the Beam Search and the Shifting Bottleneck algorithms as suggested by the closeness of the average, number of optimal solutions and results within 10% of the optimal solution shown in table 4.

Table 4.

		BAS (h_a)			BAS (h_b)			BS		SB	
Problem	Opt	C_{max}	CPU	Dev %	C_{max}	CPU	Dev %	C_{max}	Dev %	C_{max}	Dev %
6 jobs 6 machines											
MT06	55	62	0.8	12.7%	59	0.8	7.3%	N/A	N/A	59	7.3%
10 jobs 5 machines											
LA01	666	666	3.4	0.0%	666	1.9	0.0%	666	0.0%	666	0.0%
LA02	655	768	3.6	17.3%	670	6.8	2.3%	704	7.5%	720	9.9%
LA03	597	651	2.8	9.0%	633	5.2	6.0%	650	8.9%	623	4.4%
LA04	590	659	2.1	11.7%	639	1.8	8.3%	620	5.1%	597	1.2%
LA05	593	593	2.1	0.0%	593	1.8	0.0%	593	0.0%	593	0.0%
15 jobs 5 machines											
LA06	926	926	5.5	0.0%	926	6.3	0.0%	926	0.0%	926	0.0%
LA07	890	936	7.6	5.2%	890	7.2	0.0%	890	0.0%	890	0.0%
LA08	863	870	6.8	0.8%	863	6.8	0.0%	863	0.0%	868	0.6%
LA09	951	951	7.4	0.0%	951	6.7	0.0%	951	0.0%	951	0.0%
LA10	958	958	5.5	0.0%	958	6.5	0.0%	958	0.0%	958	0.0%
20 jobs 5 machines											
LA11	1222	1222	13.5	0.0%	1222	16.6	0.0%	1222	0.0%	1222	0.0%
LA12	1039	1039	11.8	0.0%	1039	16.4	0.0%	1039	0.0%	1039	0.0%
LA13	1150	1150	12.2	0.0%	1150	17.2	0.0%	1150	0.0%	1150	0.0%
LA14	1292	1292	24.8	0.0%	1292	16.7	0.0%	1292	0.0%	1292	0.0%
LA15	1207	1294	23.7	7.2%	1207	17.9	0.0%	1207	0.0%	1207	0.0%
10 jobs 10 machines											
ABZ5	1234	1325	8.8	7.4%	1266	20.3	2.6%	1288	4.4%	1354	9.7%
ABZ6	943	1101	1.3	16.8%	977	19.8	3.6%	980	3.9%	986	4.6%
LA16	945	1052	18.9	11.3%	983	10.5	4.0%	984	4.1%	1021	8.0%
LA17	784	822	8	4.8%	800	10.0	2.0%	827	5.5%	796	1.5%
LA18	848	880	27.1	3.8%	860	19.4	1.4%	881	3.9%	891	5.1%
LA19	842	908	18.3	7.8%	895	29.4	6.3%	882	4.8%	875	3.9%
LA20	902	948	7.8	5.1%	959	19.7	6.3%	948	5.1%	924	2.4%
MT10	930	1026	7.1	10.3%	1022	29.9	9.9%	1016	9.2%	1015	9.1%
ORB1	1059	1211	22.5	14.4%	1161	26.8	9.6%	1174	10.9%	1250	18.0%
ORB2	888	921	7.5	3.7%	923	26.7	3.9%	926	4.3%	1014	14.2%
ORB3	1005	1148	20.5	14.2%	1100	10.1	9.5%	1087	8.2%	1175	16.9%
ORB4	1005	1087	12.7	8.2%	1067	9.7	6.2%	1036	3.1%	1053	4.8%
ORB5	887	1018	21.6	14.8%	958	19.0	8.0%	968	9.1%	989	11.5%

Makespan. Dev(%). Relative Deviation from Optimal. CPU: CPU time in seconds. N/A. Not Available. *: Deviation cannot be calculated. The results for ORB1 to ORB5 in the Shifting Bottleneck case were found with the LEKIN ® software.

Problem	Opt	BAS (h_a)			BAS (h_b)			BS		SB	
		C_{max}	CPU	Dev %	C_{max}	CPU	Dev %	C_{max}	Dev %	C_{max}	Dev %
15 jobs 10 machines											
LA21	(1040, 1053)	1250	18.8	*	1150	35.1	*	1154	*	1172	*
LA22	927	1027	32.7	10.8%	999	71.0	7.8%	985	6.3%	1040	12.2%
LA23	1032	1097	30.7	6.3%	1032	72.7	0.0%	1051	1.8%	1061	2.8%
LA24	935	1036	23.6	10.8%	1000	34.6	7.0%	992	6.1%	1000	7.0%
LA25	977	1140	18.3	16.7%	1027	73.5	5.1%	1073	9.8%	1048	7.3%
20 jobs 10 machines											
LA26	1218	1361	50.9	0.1	1240	253.2	0.018	1269	0.042	1224	0.5%
LA27	(1235, 1269)	1471	38.2	*	1356	261.4	*	1316	*	1291	*
LA28	1216	1412	46.6	16.1%	1361	176.2	11.9%	1373	12.9%	1250	2.8%
LA29	(1120, 1195)	1302	48.6	*	1233	270.3	*	1252	*	1239	*
LA30	1355	1469	47.5	8.4%	1424	263.3	5.1%	1435	5.9%	1355	0.0%

		BAS (h_a)	BAS (h_b)	BS	SB
Average		7.38%	3.78%	4.26%	4.20%
Optimal solutions		9	13	12	12
Within 10%		23	35	33	31

6.CONCLUSIONS

This paper has presented a comparative study in which a Petri Net-based algorithm has been tested on a number of test problems. The major conclusion that Petri Nets but can be a powerful tool for scheduling and control of manufacturing systems and are not just a modeling and simulation tool.

In this paper a new algorithm, named Beam A* Search (BAS), was presented. The main features of the BAS algorithm presented here include the intelligent pruning of the search space, a controlled search deepening to avoid marking explosion and the development of new heuristic functions to estimate the remaining cost *h(M)* for a marking *M*. Competitive results were obtained on standard job shop problems compared against similar algorithms such as Beam Search (Sabuncuoglu, and Bayiz, 1999) and the Shifting Bottleneck algorithms (Adams et al, 1988). The BAS algorithm exhibits in addition to good solutions, very attractive computational times which, in turn, makes it attractive for real-time control of flexible manufacturing systems.

Further research is needed on developing a BAS version for minimizing other criteria such as mean tardiness or maximum lateness. Also efforts would be concentrated on developing new heuristic functions *h(M)*, improving both the marking pruning and the marking selection.

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.

REFERENCES

1. Moore, K.E. Gupta, S.M. "Petri Net Models of Flexible and Automated Manufacturing Systems: A Survey". *International Journal of Production Research,* 34, no 11, 1996, p 3001-3035
2. Lee, D. Y. DiCesare, F. "Scheduling Flexible Manufacturing Systems using Petri Nets and Heuristic Search". *IEEE Transactions on Robotics and Automation*, 10 no. 2, 1994, pp. 123-131.
3. Sun, T. Cheng, W. Fu, L. "A Petri Net Based Approach to Modeling and Scheduling for an FMS and a Case Study". *IEEE Transactions on Industrial Electronics.* 41, no 6, 1994, pp. 593-601.
4. Xiong. H. H. Zhou M.C. "Scheduling of Semi-Conductor Test Facility via Petri Nets and Hybrid Heuristic Search". *IEEE Transactions on Semiconductor Manufacturing*, 11, no 3, 1998, pp.384-393.
5. Jeng, M. D. Chen, S. C. "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, 1998, pp.139-162.
6. Reyes-Moro, A. Yu, H. Kelleher, G. Lloyd, S. "Integrating Petri Nets and Hybrid Heuristic Search for the Scheduling of FMS". *Computers in Industry,* 47, 2002, pp.123-138.
7. Sabuncuoglu, I. and Bayiz, M. "Job shop scheduling with Beam Search". *European Journal of Operational Research,* 118 no. 2 1999, pp.390-412.
8. Zhou, M. DiCesare, F. *Petri Net Synthesis for Discrete Event Control of Manufacturing Systems.* USA. Kluwer Academic Publishers. 1993.
9. Liu, C, Ma, Y. Odrey, N. "Hierarchical Petri Net Modeling for System Dynamics and Control of Manufacturing Systems". *Proceedings of the 7th FAIM International Conference.* June 25-27. 1997. Middlesbrough, UK.
10. Murata, T. "Petri Nets: Properties, Analysis and Applications". *Proceedings of the IEEE,* 77 no 4, 1989, pp. 541-580.
11. Russell, S. Norvig, P. *Artificial Intelligence: A Modern Approach.* Prentice-Hall. 1995.
12. Morton, T. Pentico, D. *Heuristic Scheduling Systems with Applications to Production Systems and Project Management.* John Wiley & Sons. New York. 1993.
13. Baker, K. R. *Introduction to Sequencing and Scheduling.* Wiley. 1974.
14. Pinedo and Chao, *Operations Scheduling with Applications in Manufacturing and Services.* Irwin-Mc Graw-Hill. 1999.
15. Applegate, D. Cook, W. "A Computational Study of the Job-Shop Scheduling Problem". *ORSA Journal on Computing,* 3, no 2, 1991, pp. 149-156.
16. Muth, J. F. Thompson, G.L. (eds.) *Industrial Scheduling*. Englewood Cliffs, NJ. Prentice Hall, 1963.
17. Adams, J., Balas, E., Zawack, D. "The Shifting Bottleneck Procedure for Job Shop Scheduling. Management Science Research". Report No. MSRR-525 (R), Carnegie Mellon University. 1987.

18. Lawrence, S. "Resource Constraint Scheduling: An experimental Investigation of Heuristic Scheduling Techniques". GSIA, Carnegie Mellon University. 1984.