Proceedings of the 2000 IEEE
International Conference on Robotics & Automation
San Francisco, CA • April 2000

Advanced Scheduling Methodologies for Flexible Manufacturing Systems using Petri Nets and Heuristic Search

Antonio Reyes Moro[†] Hongnian Yu[*] Gerry Kelleher[†].

[†] School of Computing and Mathematical Sciences
Liverpool John Moores University
Liverpool L3 3AF
England

[*] School of Engineering and Computer Science
Exeter University
Exeter EX4 4QF, England
Email: H.Yu@Exeter.ac.uk

Abstract ---*The combination of Petri net (PN) and AI to solve flexible manufacturing systems (FMS) scheduling problems has been proven to be a promising approach. However, the NP-hard nature of the problem prevents the PN capability of reasoning about the behavior of a practical system. To overcome this drawback, we propose two techniques: a systematic method to avoid the generation of unpromising paths within the search graph and a stage-search based algorithm. The algorithm developed is based in the application of the A* algorithm and the PN-based heuristics. The search is performed within a limited local search window where an optimization policy is applied to evaluate the most promising paths. For each state, the algorithm is able to decide whether an enabled operation is applied, and to maintain the decision until new system information makes the reconsideration meaningful. Comparison with previous work is presented to show the superiority of the proposed approach.*

1. Introduction.

An *FMS* usually consists of several numerically controlled manufacturing machines, tool systems and other resources, capable of producing several products using alternate routing and in changing demands. A high-level control system must decide what resources assign to what product and at what certain instant, so as to optimise some criteria, makespan in our case. The purpose of scheduling is then to determine when to process which job by which resources so that production constraints are satisfied and production objectives are met.

*PN*s are an ideal tool for modelling *FMS* since they naturally capture the characteristics and constraints of *FMS* (concurrency, synchronisation, mutual exclusion and conflict) both systematically and in a single coherent formulation. In addition, the *PN* formalism defines the scheduling problem in terms of a state-space reachability analysis that allows the direct application of AI search algorithms. Perhaps most important, a *PN* model provides information (by well-understood mathematical analyses) that can be used to guide the scheduling process.

The integration of *PN* modelling and heuristic search has attracted numerous researches Branch & Bound search is employed in (Lloyd *et al.*, 1995; Chen *et al.*, 1994; Abdallah *et al.*, 1998). The $A*$ algorithm has been applied in (Lee and Dicesare, 1994; Yim and Lee, 1996).

Sun et al., (1994) and more recently Inaba *et al.*, (1998); Jeng *et al.*, (1998) and Reyes *et al.*, (1998), implement $A*$ but limit the backtracking capability of the algorithm by introducing irrevocable decisions. A hybrid algorithm combining *Best-First* and *B&B* search methodologies has been considered in (Xiong and Zhou, 1998). *Beam Search* as on-line decision support is implemented in (Shih and Sekiguchi, 1991).

The main obstacle to consolidate the power of this combination is the very large search space for large problems. To reduce this combinatorial explosion we propose a *stage search* algorithm that combines predictive heuristic information based on the *PN*, heuristic dispatching rules and optimisation criteria. Besides, we propose an approach for exploring search spaces that can avoid the generation of duplicated paths and obvious non-optimum paths

2. Scheduling of *FMS* based on *PN* structures: Representation and Reasoning.

We assume the reader is familiar with basic *PN* formulations and theory, (see Murata 1989) for background) and also with *PN* models for *FMS* descriptions (see Lee and Dicesare (1994); Jeng et al (1998)).

If a *PN* is a valid model of an *FMS*, the scheduling problem may be translated into a search problem of finding a desired path with the lowest cost (makespan) in a graph structure that is the *PN* reachability tree (Murata 1989). Unfortunately, it is well known that the generation of the reachability tree takes exponential time for the general case.

Lee and Dicesare (1994) adapt the well-known $A*$

0-7803-5886-4/00/$10.00

algorithm to the scheduling of *FMS* based on *PN* structures. *A** tries to reduce the search space by exploring only the promising markings using a heuristic function $f(m) = g(m) + h(m)$. $g(m)$ is the current cost (makespan) of a marking m, while $h(m)$ is an estimation of the remaining makespan to reach the final marking from m. To guarantee that *A** finds an optimum solution $h(m)$ must be *admissible* (Pearl 1984) i.e. $h(m)$ is always lower than the actual minimum makespan that can be obtained from m to the final marking. However, the employment of an admissible heuristic in large problems makes *A** sinks into a *breath-search* sooner or later.

To overcome the above drawback while maintaining an admissible *PN* based heuristic function, we propose two methods: a systematic method to avoid the generation of unpromising paths within the search graph and a stage-search based algorithm. The former is called intelligent generator of successors (*IGS*) and the latter is called dynamic window search (*DWS*).

3. Intelligent Generator of Successor

The *IGS* attempts to reduce the search effort by avoiding the generation of intermediate schedules that are known not to lead to an optimum solution.

To present the method, let us suppose that an operation (transition) is enabled at a state (marking) m. This has two meanings: a) all the resources are ready (or will be ready in a finite and known time); and b) the application of the operation in the system does not violate the buffer constraints. At this point, the scheduler must make a decision about this operation by either a) applying the operation now; or b) delaying the operation. As a result, there will be two subsets of possible new succeeding states of the current state: those where the operation *op* has not been applied in m, and those where it actually has been applied.

Hence, when a marking m is under evaluation, we can define an associated list of operations that are potentially applicable at this state. We will call this list *Agenda* because of its parallelism with a rule-based production system.

For the first operation *op* in the list, the scheduler may generate two new successors:

- A new state m' resulting by applying *op*, which has a new $Agenda(m')$ formed by the remaining operations of m and all the newly instanced operations.
- A copy m'' of the same state m but with *op* removed from $Agenda(m)$.

Notice that only m' needs to be stored for further exploration since m'' is just a copy of m and no operation has been applied.

In our *FMS* scenario, the decision of not immediately perform an operation now may be motivated because its delay may allow the allocation of resources that eventually can lead us to a more promising situation. As a general rule, an operation in the system expressly decided not to be applied, will be delayed until a meaningful change in the system is observed for that operation. The simple event is the reactivation of the operation which produces a new entry in the agenda.

The representation paradigm used in our scheduling environment is a timed-rule based-*PN* (Konstans et al 1998), where we only consider two types of constraints: a) resource constraints such as machines, buffers, etc.; and b) time constraints such as minimum and maximum waiting times at buffers. Operations are modelled by transitions in the *PN* model. The input places model the availability of resources. If a transition is decided not to be fired, it will be delayed until new tokens representing resources become available (buffers, machines). On the other hand, if time constraints are involved, the operation should be reconsidered at any instant in the future. Intuitively, we can express the rule within our *FMS* context as:

OPERATOR: *Fire transition t.*
PRECONDITON: *PN enabling rule for t.*
Rule-based PN time constraints
SENSITIVE: *New resources available for t.*
||Time increased.
ACTION: *PN firing Rule for t.*

Figure 1 shows the pseudo code for *IGS*. The parameter **SENSITIVE** is used in the algorithm (step (3)) to determine if an already enabled transition not included in the list *Agenda* must be re-asserted in this list. The list $Agenda(m)$ is formed by a subset of the potentially enabled transitions t in m and it is ordered in the increasing magnitude of $c(m,t)$ which is the remaining time for t to be fired at m, due to delays affecting tokens.

m is the current marking under evaluation
$Agenda_m$ is a list of enabled transitions at m
t is the transition selected for firing

1. obtain new state m' after applying t.
2. create $Agenda(m')$ as the transitions of $Agenda(m)$ after t which still are enabled in m'.
3. add **any newly enabled** transition t' in m' to $Agenda(m')$
4. Reorder $Agenda(m')$ in the increasing magnitude of $c(t,m')$

Figure 1: *IGS* algorithm.

This methodology is integrated in the hybrid search algorithm described in the next section.

4. Hybrid Heuristic search: Dynamic Window Search algorithm *(DWS)*.

The application of an admissible A^* algorithm is only affordable for small problems and consequently the goal of optimality must be sacrificed in favour of computation complexity reduction. A typical procedure is to implement hybrid algorithms that attack the two dimensions of A^* namely *scope of selection and scope of recovery* (see Pearl 1984).

The search algorithm developed (*DWS*) follows the basic schema of *stage search* and A^* by considering the following three ideas with the objective of reducing the size of the search tree under consideration while increasing the chance of finding an optimum solution:

- a reduction of the *scope of selection*, by considering only the most promising transitions at each marking.
- a reduction of the *scope of recovering*, by limiting the backtracking capability of A^*.
- A truncation of the number of candidate markings based on an optimisation policy.

The search space can be now partially seen through a window or frame where the bottom edge represents the *PN* markings to which we still have the possibility to backtrack. The upper edge contains the deepest markings, and the window itself represents a reduced sub-tree where the search algorithm is applied. To prevent exponential grown, the size of the window is limited. The window provides us with a safety frame that guides the search in a promising direction.

4.1. Defining the limits of the search window.

The depth of a marking *m* in the reachability tree (*depth(m)*) can be defined by the number of transitions that has been fired from the initial marking m_0 in order to reach *m*.

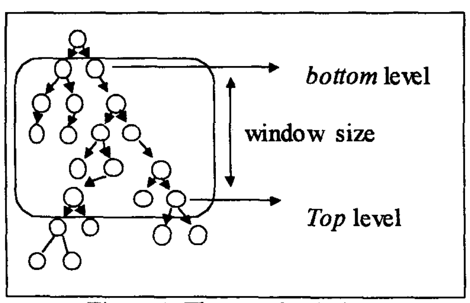

Figure 2. The search window.

Figure 2 shows the geometry of the search window defined in terms of two integers: *bottom-depth* and *top-depth* which defines the minimum and maximum depth of the markings that it can contain. Hence, the search window only contains markings whose *depth(m)* is included in *[top - bottom]*. Markings at a certain depth *l* define a level in the search window. The number of markings at level (depth) *l* is expressed as *size(l-depth)*

4.2 Introducing irrevocable decisions.

While the search window remains static, markings at *top-depth* can be created but not explored. However, it is a must that the search window moves forward to deeper markings in the reachability tree in order to reach leaf nodes that represent complete schedules. A pair of rules determines this advance as follows:

Rule 1: if *size(top-depth)* ≥ *max-top* then
discard markings at *bottom-depth*
top = *top* + 1;
bottom = *bottom* + 1;

Rule 2: if *size(bottom-depth)* = *0* then
top = *top* + 1;
bottom = *bottom* + 1;

The first rule models the amount of information about the quality of the markings contained in the search window that we want to consider before deciding to introduce further irrevocable decisions.
Since the search is mainly guided by the heuristic function *f(m)*, we propose to reject markings at *bottom* level (depth) when a certain number *max-top* of markings at the *top-depth* is found. Figure 3 shows this. Notice that since the window size must be a constant, the *bottom-depth* must also be increased by one.

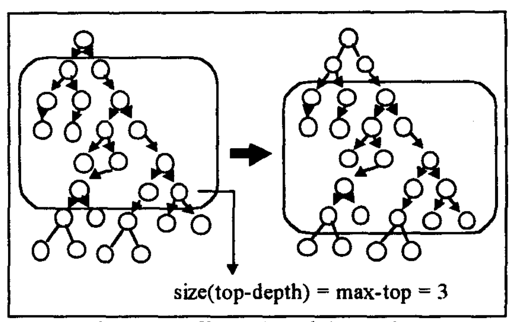

Figure 3. Effect of applying *rule 1*.

Obviously, when the window advances, the level that was the *top* is not constrained by *max-top* anymore, thus more markings of depth *top* can be generated.

If the heuristic function $h(m)$ guides the search satisfactorily, promising markings at *top-depth* will be found quickly, thus activating *rule 1*. However, if more search effort (backtracking) is needed, *bottom* level may be emptied. The second rule detects when all markings at *bottom depth* has been explored and hence *size(bottom-depth)=0.* In this case, the window is advanced as well as shown in figure 4, allowing the search to move forward.

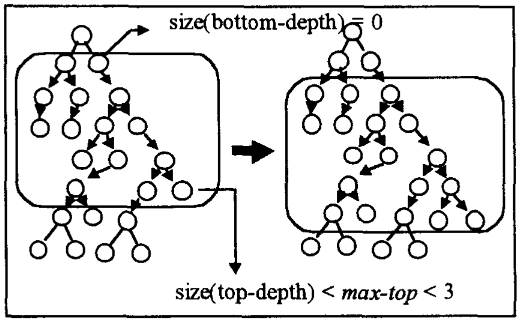

Figure 4. Effect of applying *rule 2.*

4.3. Avoiding exponential grown.

Even for a small distance between *top* and *bottom* levels of the search window, the number of markings will grow exponentially as the window advances. To allow larger distances (to increase backtracking capability) and avoid the exponential explosion, a maximum number of markings is allowed at each level. Because the number of markings that the window can contain is limited, it is critical to include the most promising ones and reject the others. The argument for including a marking is given by the following rule:

Rule 3: if *a new marking* m *is created ot depth l* then
if *size(l-depth) < max-size* then
include *marking*
else if *f(m)< WORST (l)*
include *m*
else reject *m*.

At each depth, nodes (markings) are ordered in the increasing magnitude of $f(m)$. Any new marking can be included into the level if the level is not full yet. If the level is full, the node will be included only if a marking with a bigger value of $f(m)$ is included in the level. The marking with the worst value of $f(m)$ $(WORST(l))$ will be rejected for further exploration. It is noted that this represents a dramatic decision in terms of both avoiding backtracking and limiting the number of paths for further exploration. It is important that markings that are known not to be optimum ones are not included in the search. This justifies the integration of *IGS* within *DWS*.
Because the number of markings in the search window is limited, we expect that the performance of the algorithm should reflect a polynomial cost against the size of the problem. Preliminary results seem to confirm this.

The approach presented is a simpler one, but produced interesting results. Further empirical analysis can be made. For example, consider markings not only because it improves the quality of other markings at the same level, but also because the marking represents an improvement over the markings in upper and/or lower levels.

4.4. PN based heuristic functions.

Three heuristic functions obtained by means of analysis of the current marking m and the *PN* structures are employed to guide the search process.

The first one is the heuristic function employed to guide A^*, which is obtained as $f(m)=g(m) + h(m)$ where $g(m)$ is the actual makespan of the marking m. $h(m)$ is based in the solving of an alternate problem. This alternate problem may be represented by a *PN* that is obtained by eliminating those places representing resources in the original *PN*. The delay associated with each transition is also changed in the following way: the new cost is obtained by multiplying the original cost by the number of resources used by the transition. Within this model, it is possible to determine the minimum cost path in terms of resource utilisation for a token to progress from place p to p'. $h(m)$ is finally obtained as $h(m) = U(m) / R$, where $U(m)$ is the optimal solution to the minimum machine utilisation scheduling problem and R is the total number of resources. Hence, we are assuming that parts can always be achieved following the less resource consuming routes and that no contention for the use of resources is produced. This is a lower bound for the makespan and consequently an admissible heuristic function.

The second heuristic function is used as a kind of dispatching rule to reduce the number of potentially fireable transitions at each marking m. We defined the following heuristic estimation for the firing of a transition t under the current marking m that produces a new marking m' in the search graph: $k(t,m) = c(t,m) + r(m')$ where $c(t,m)$ is the cost of reaching m' from m. $r(m')$ is a lower bound for the cost $c(t', m')$ for any t' enabled at m'. $r(m)$ tries to model how soon we can apply the next operation. The methodology is integrated in *IGS* so it truncates *Agenda* considering only a number N of transitions with the lower value of $k(t,m)$.

Finally, a third function $j(m)$ is computed to heuristically determine if a path to the marking m might lead to a better solution than a previous reached one. It is obtained as $j(m) = g(m) + d(m)$, where $d(m)$ is the average delay for tokens to become available at places representing resources. The complete heuristic search algorithm *DWS* is presented in Fig. 5.

1. *OPEN* = $\{m_0\}$ *CLOSED* = ∅
2. **if** *OPEN* = ∅ **exit with failure**
3. **apply Rule 2**
4. remove *m* in *OPEN* such as *f(m)* is minimal *and* ***depth(m)*< *Top-depth*.** Put *m* in *CLOSED*
5. **if** *m* is the goal marking **then exit with success**
6. Obtain *E* as the *N* transitions of *Agenda(m)* yielding a lower value of *k(m,t)*
7. **while** *E* is not empty
   - 7.1. remove *t* from *E*.
   - 7.2. obtain *m2* and *Agenda(m2)* after firing *t* in *m* following *IGS*
   - 7.3. **if** *m2* is already in *OPEN* redirect pointers to the marking yielding the smallest value of *j(m2)*.
   - 7.4. **if** *m2* is already in *CLOSED* and *j(m2)* does not indicates a more promising path then **goto** 7
   - 7.5. **apply Rule 3** to *m2* to be included in *OPEN*
   - 7.6. **apply Rule 1**
   - 7.7. **goto** 7
8. **goto** 2

Figure 5. *DWS* algorithm

5. Experimental results.

This section presents several experimental tests to demonstrate the proposed algorithm. *DWS* settings are given as *DWS(high,max-size,N)* where *high* stands for the distance between the *bottom-depth* and the *top-depth*, *max-size* stands for the maximum number of markings per level which is also equal to *max-top*, and *N* is the maximum number of transitions to consider for each marking.

All the algorithms were implemented in C++ and run in a 300 Mhz PentiumII PC with 64 Mb RAM.

5.1. IGS performance.

In order to study the amount of search space pruned by the *IGS* algorithm we have performed two experimental tests: *Test a)* consists of 1000 *FMS* where five jobs have to be scheduled. Each job has four tasks and an average of 75% of tasks has alternate routing. *Test b)* represents 1000 pure *JSS* with five jobs to schedule and no alternate routing. *Test b)* represents a set of problems where the *PN-based* heuristic *h(m)* is more optimistic than for *Test b)*.

We solved each problem twice: 1) using the *A** algorithm of (Lee and Dicesare 1994) and 2) including *IGS* in the same approach. The relative difference of the number of markings explored between *1)* and *2)* follows a normal distribution of mean 35.0% and std. div 9.58% for problem set *a)*. For the problem set *b)* the relative difference of markings explored now follows a normal distribution of mean 33.6% and std. div 8.47%. This confirms that *IGS* can reduce the search effort.

5.2. DWS performance.

A set of 1000 *FMS* scheduling problems was generated with a random problem generator. The generator was set with the objective of increasing the chances to obtain solutions where the concurrence between machines is maximal. Hence, the proposed heuristic function *h(m)* is a good approximation to the optimum solution. Basically, the typical *FMS* generated with these setting corresponds to a medium size problem (210 operations to schedule approx.), where 75% of the tasks have alternate routing. Each problem was solved using *DWS(10,5,15)* which pursues to keep the computational effort affordable.

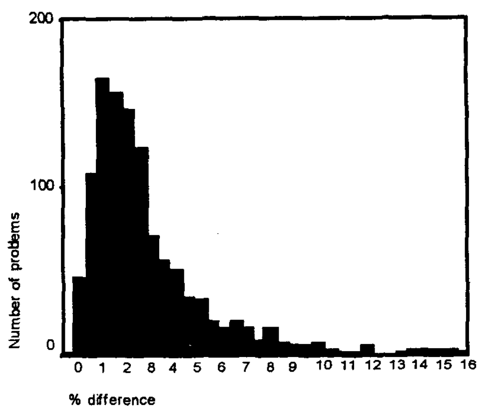

Figure 6. Frequency histogram for the relative distance to the lower bound.

Figure 6 shows the frequency histogram for the relative difference between a lower bound for the makespan and the actual makespan obtained with *DWS* for each problem. The lower bound is obtained as $h(m_0)$ being m_0 the initial marking. It is clearly seen that the figure represents a normal distribution disturbed on its right side. Again we must state that the problems were randomly generated, and the lower bound is a hypothetical expression that will never be reached in most of cases. Solving the same set with *DWS(15,10,25)* reduces the average relative difference in 0.9%, but takes an average of 5.77 times the execution time of *DWS(10,5,15)*.

5.3. Comparison with previous works.

Lee and Dicesare (1994) implement *A** and fight intractability by proposing a heuristic function that makes the algorithm prefer markings that are deeper in the reachability graph. *Table 1* shows the comparison results for three problems proposed. $L\&D^2$ stands for our implementation of their algorithm. *DWS** represents the results obtained when the number of markings explored matches with the ones reported for each problem.

Problem	L&D [11*]	L&D²	*DWS*	*DWS**
1	426	381	329	338
2	298	279	254	264
3	273	260	237	247

Table 1. Comparison results for benchmarks proposed in Lee and Dicesare (1994)

Table 2 shows the results obtained for different lot sizes for a *JSS* with four jobs proposed in (Xiong and Zhou 1998). The hybrid algorithms proposed there did not find the optimum solution. *DWS(10,10,15)* obtained the optimum result with a considerable much less effort.

Lot size	Makespan			Number of markings		
J1,2,3,4	*BF*	*BT-BF*	*DWS*	*BF*	*BT-BF*	*DWS*
5,5,2,2	58	62	58	3437	1687	431
8,8,4,4	100	104	100	9438	8045	856
10,10,6,6	134	148	134	23092	18875	1204

Table 2. Comparison results for benchmarks proposed in Xiong and Zhou (1998)

A second example from (Xiong and Zhou 1998) was adopted from a real Integrated Circuit sort and test floor in San Jose, CA. The system consists of 79 resources, and 30 jobs to schedule, each job formed by three tasks. No alternate routing is available and the total number of operations to schedule is 90. The best makespan reported was 30. *DWS* found a makespan of 28 in 12 seconds.

6. Summary.

Two techniques have been presented to solve the scheduling problem of *FMS* formulations modelled with *PN*. *IGS* allows reducing the search effort without losing optimality. On the other hand, *DWS* represents a search algorithm that effectively controls the search effort and that allows pure *PN-based* analysis information to be applied.

IGS, DWS, PN modelling and *PN-based* heuristics have been merged in a hybrid algorithm that produced very promising results. This shows the viability of the integration of the *PN* formalism with traditional AI search strategies. Besides, it opens a research frame where to compare different heuristics obtained from structural and mathematical analysis of *PN* models.

References

Abdallah, I.B., ElMaraghy H.A. and ElMekkawy: An Efficient Search Algorithm for Deadlock-free Scheduling in *FMS* using Petri Nets. Proc. IEEE Int. Conf. On Robotics and automation. Leuven, Belgium 1998 (1793-1798).

Chen, Q., Luh, J.Y.S and Shen, L. Complexity Reduction for optimization of deterministic Timed Petri-Net Scheduling by Truncation. *Cybernetic and Systems: An international Journal*. **25** (643-695), 1994.

Inaba, A., Fujiwara, F., Suzuki, T. and Okuma, S. Timed Petri Net based scheduling for Mechanical Assembly. *IEICE Transactions on Fundamentals of Electronics Communication and Computer Sciences.* **E-81A** (615-625). 1998

Jeng, M.D., Chiou W.D., and Wen, Y-L. Deadlock-Free Scheduling of Flexible Manufacturing Systems Based on Heuristic Search and Petri Net Structures. *Proc. Of the 28th Int. Conf. On Systems, Man and Cybernetics.* 1998 (26-31)

Konstans, S. Lloyd, Yu, H. *Rule based Petri Net Modelling and scheduling of an FMS.* Proc. The 14th NCMR Conference 1998

Lee D.Y., and Dicesare, F.: Scheduling Flexible Manufacturing Systems Using Petri Nets and Heuristic Search. *IEEE Transactions on Robotics and Automation* **10** (123-132) 1994.

Lloyd, S., Yu, H. and Konstants, N. *FMS* scheduling Using Petri Net Modeling and Brach & Bound Search. *Proc. IEEE International Symposium on Assembly and Task Planning*, Pittsburgh, USA, (141-146) 1995.

Murata, T: Petri Nets: Properties, Analysis and Applications. *Proceedings of the IEEE*, **77** (541-580) 1989.

Pearl, J. *Heuristics: intelligent search strategies for computer problem solving*. Addison-Wesley 1984.

Reyes, A., Yu, H and Kelleher, G. Applying new search methodologies for scheduling *FMS* using *PN. Proceedings of the sixteenth workshop of the UK Planning and Scheduling Special Interest Group.* 1998

Shih H. M. and Sekiguchi, T.: A Timed Petri Net and beam search based on-line *FMS* scheduling system with routing flexibility. *Preceedings of the 1991 IEEE International Conference on Robotics and Automation.* Sacramento California. (2548-2553) 1991.

Sun, T., Cheng, C. and Fu, L: A Petri Net Based Approach to Modeling and Scheduling for an *FMS* and a Case Study. *IEEE Transactions on Industrial Electronics*, **41** (593-601). 1994

Xiong H.H and Zhou, M.C. Scheduling of Semiconductor Test Facility via Petri Nets and Hybrid Heuristic Search. *IEEE Transactions on Semiconductor Manufacturing*, **11** (384,393), 1998

Yim, S.J. and Lee, D.Y. Multiple Objective Scheduling for Flexible Manufacturing Systems Using Petri Nets and Heuristic Search. *IEEE Int. Conf. On Systems , Man, And Cybernetics. Information Intelligence and Systems.* (2984-2989) 1996.