[17] C. G. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems*. Norwell, MA: Kluwer, 1999.
[18] Y. Li and W. M. Wonham, "Deadlock issues in supervisory control of discrete event systems," in *Proc. Conf. Inf. Sci. Syst.*, 1988, pp. 57–63.
[19] P. J. G. Ramadge and W. M. Wonham, "The control of discrete event systems," *Proc. IEEE*, vol. 77, pp. 81–98, 1989.
[20] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. San Francisco, CA: Freeman, 1979.

Hybrid Heuristic Search for the Scheduling of Flexible Manufacturing Systems Using Petri Nets

Antonio Reyes Moro, Hongnian Yu, and Gerry Kelleher

***Abstract*—The combination of Petri nets (PNs) as an analysis tool for discrete-event dynamic systems and artificial intelligence heuristic search has been shown to be a promising way to solve flexible manufacturing systems (FMS) scheduling problems. However, the NP hard nature of the problem obscures the PN capability of reasoning about the behavior of the system. In this paper, two techniques to alleviate this drawback are presented: a systematic method to avoid the generation of futile paths within the search graph and a novel hybrid stage-search algorithm. The new algorithm is based on the application of A^* guided by a *PN-based* heuristic within a limited local search frame. An optimization policy is applied to maintain, under evaluation, only the most promising paths. For each system state, the algorithm is able to decide whether an enabled operation should be applied and to maintain this decision until new information forces reconsideration. This eliminates permutation paths and useless scheduling sequences. Experimental results show that the algorithm's cost does not grow exponentially with the size of the problem. Comparison with previous work is given to show the superiority of our approach and the potential of *PN-based* heuristic search.**

***Index Terms*—Flexible manufacturing systems, heuristic search, Petri net, scheduling.**

I. Introduction

A flexible manufacturing system (FMS) usually consists of several numerically controlled manufacturing machines and automated material handling systems that transport work pieces between machines and tool systems. In a facility with routing flexibility, each product can be manufactured via one of several available routes. The scheduling of an FMS is the process of determining the allocation of parts to machines and the sequence of operations so that the constraints of the system are met and performance criteria are optimized.

Motivated by the need to develop models that factor in the full complexity of the FMS, yet are efficient enough to obtain good solutions, the recent integration of Petri nets (PNs) [7] as a representation tool for FMS [11], [22] with artificial intelligence (AI) problem-solving methods as a reasoning paradigm appears promising.

In this paper, we extend the work begun in [12], [13], and [15] by presenting a limited-selection limited-backtracking algorithm called DLSS* which stands for PN-based dynamic local stage search A^*. The aim is to reduce the scope of evaluation of the A^* algorithm so that we may apply a more exhaustive local search and to enhance the usefulness of an admissible heuristic function [12] based on the PN model. To enhance the power of the algorithm in selecting promising paths, we also propose a branching scheme for DLSS* called controlled generator of successors (CGS) [16] that avoids the generation of both schedule permutations caused by *concurrent* transitions and certain nonactive schedules [10].

The research thus aims to provide effective decision modules integrated with knowledge-based architectures that employ a PN as a representation paradigm [7], e.g., hybrid methodologies based in random optimization such as [14].

II. Background

A. Problem Formulation

A number of jobs are to be processed in the system. Each job has several process plans and each plan is a sequence of temporarily related tasks ordered by the technological constraints. Each task can be processed in several ways. Each alternative may require one or several resources. The following assumptions are taken.

- Each machine can process at most one task at a time and no preemption is allowed.
- Each task consumes a single subpart and produces only a single subpart (there is no assembling).
- For the scope of the paper, we have limited our results to the case where an infinite buffer policy applies in the system. However, different storage models can be applied as presented in [13].
- Machine tool loading and setup are considered negligible.

Details of the problem formulation and the PN modeling are given in [12].

B. Scheduling of FMS Based on a Heuristic Search Over PN Structures

Several works have addressed the combination of PN simulation capabilities and AI-based systematic search within the PN reachability graph to solve FMS scheduling problems. The Beam search (BS) algorithm as online decision support is implemented in [17] and the Branch & Bound (B&B) search is employed in [6]. More recently, an *informed* B&B to generate deadlock-free schedules has been implemented in [1].

The L1 algorithm in [5] adapts the A^* to PN structures to perform FMS scheduling. L1 bases its strategy on maintaining a list of candidate markings for further exploration. The selection of the next marking to explore is based on a heuristic function $f(M)$ that 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, $h(M)$ represents an estimate of the remaining cost (makespan) to reach the marking that represents the goal state M_F. The performance of A^* basically lies in how good our heuristic function is.

To guarantee that A^* finds the optimum solution, the heuristic function must be admissible [9], i.e., it must be a lower bound (Lb) for the actual makespan. Unfortunately, under these conditions, A^* turns out to be impractical for even small problems since the search effort becomes unaffordable.

Manuscript received August 31, 2000; revised May 9, 2001. This paper was recommended for publication by Associate Editor M. Zhou and Editor N. Viswanadham upon evaluation of the reviewers' comments.

A. Reyes is with iSOCO Lab at iSOCO, Intelligent Software Components, Sant Cugat del Vallés 08190, Barcelona, Spain (e-mail: toni@isoco.com).

H. Yu is with the School of Engineering and Computer Science, Exeter University, Exeter EX4 4QF, U.K. (e-mail: h.yu@exeter.ac.uk).

G. Kelleher is with the School of Computing and Mathematical Sciences, Liverpool John Moores University, Liverpool L3 3AF, U.K. (e-mail: g.kelleher@livim.ac.uk).

Publisher Item Identifier S 1042-296X(02)02651-4.

Consequently, a typical problem observed is the difficulty in balancing the control of the search effort with the potential of PN to guide a heuristic search process [5]. An immediate solution is to add a depth-first (DF) search component to the heuristic function, thus making a pure A^* to quickly progress to a solution [5], [21]. The main problem is that the tuning of the balance between the makespan estimation component of $h(M)$ and the DF search component is difficult and the search might produce unexpected results, obscuring the PN capability of reasoning about the behavior of the system.

An alternate solution comes from applying admissible PN-based estimates to guide the search while controlling the search effort with an incomplete algorithm. Actually, limiting the backtracking capability of a *best-first A^*-alike* search algorithm has already been considered by some authors that combine PN modeling with heuristic search.

In [2], [19], and [21], a limitation of the candidate markings for exploration is proposed by simply rejecting those markings with the maximum cost of $f(M)$. However, these works still consider the application of weak heuristic functions that include a DF component. Nevertheless, the main problem with this approach employing an admissible heuristic function is that the backtracking capability of the algorithm is seriously limited due to the existence of a large number of equally promising markings.

An approach that may solve this can be found in [3], where, although the approach does not limit the number of candidate markings, it removes those markings beyond a maximal depth from the current marking that is being explored. A more elaborated approach is followed in [12] where a hybrid search algorithm based on relaxation of the evaluation scope of an A^* is proposed. However, this does not fully prevent the exponential explosion if an admissible heuristic function is used. This is due to the fact that, even limiting backtracking, the number of nodes grows exponentially at each level, thus considering large *maximal depths* are prohibitive.

The algorithm that we describe in the next section [dynamic look-ahead stage search (DLSS*)] aims to overcome these difficulties. DLSS* has its basis in concepts of look-ahead algorithms such as BS [8] and adaptations of well-known game search algorithms to real-time scheduling [4]. It also follows the philosophy of the staged search method [9].

III. Dynamic Look-Ahead Stage Search (DLSS*)

A. Heuristic Function h_{RCR}

The heuristic function h_{RCR} [12] employed in DLSS* is an admissible one [9] and solves an alternate problem: the minimization of the total machine utilization, which represent an Lb for the actual makespan.

B. DLSS* Description

The aim of DLSS* is to implements an A^* strategy that employs h_{RCR} but avoiding the exponential generation of heuristically equally valued markings as the search progresses. The A^* search is constrained by a number of markings that are contained in structure called the Search Frame (SF) which is conceptually identified with the OPEN list in the A^* algorithm. The number of markings that this frame can contain is limited; thus avoiding exponential grown. SF changes dynamically both in: 1) contents and 2) search limits:

1) Dynamics of SF: Each marking M in the reachability tree is associated with the number of transitions that have been fired. This value is equivalent to the depth of M in the search tree: $depth(M)$. In our modeling paradigm, since operations are represented by transitions, $depth(M)$ also matches with the number of operations that have been scheduled. We have organized SF into levels; each level is labeled with a number that represents a depth in the PN reachability graph and

Rule 1: *if Bottom level has been completely explored, ⇒ advance the window.*

if *size(**bottom**) = 0* then
    ***top** = **top** + 1;*
    ***bottom** = **bottom** + 1;*

Rule 2: *if relevant information is found, ⇒ consider nodes at bottom+1 as irreversible decisions and advance the window.*

if *size(top) ≥ **move-at*** then
Reject *markings* in *bottom-Level.*
    top = top + 1;
    bottom = bottom + 1;

Fig. 1. Rules 1 and 2.

Rule 3: *if a new marking* m *is created for level l…..*
    if *size(l) < max-nodes(l)* then
        include *marking* m *in SF*
    else if *f(m) < WORST (l)*
        include *marking* m *in SF*
        reject *WORST (l)*
    else **reject** *marking.*

Fig. 2. Rule 3: the heuristic selection of markings for exploration of SF.

therefore SF contains a total of top–bottom levels. A level l can only contain a maximum of $max\_nodes(l)$ markings of $depth(M) = l$. The number of markings currently allocated in this level is expressed as size(l).

SF limits the application of A^* in two ways:

- the backtracking capability is constrained to markings whose depth is equal to or bigger than a *bottom-level* value;
- nodes of depth equal to a *top-level* constant can be generated but not explored.

Hence, a marking M is only included in SF if $depth(M) \in [top\text{-}bottom]$ and an inclusion criterion is applied at each level of SF.

2) Introducing Irrevocable Decisions: Dynamics of SF: SF provides a bounded safety frame within which to apply a heuristic best-first search that considers the markings at the bottom level as irrevocable decisions. But the search must progress toward a final marking, representing a solution to the problem. The advance of SF will be determined by two rules. A first rule (Rule 1) advances the search windows when the bottom level is empty. A second rule (Rule 2) allows the search to progress DF when a number of markings are found at the top level. These are shown in Fig. 1.

3) Inclusion Criterion: The second dynamic aspect of SF is the heuristic selection of markings for exploration of SF. This strategy is implemented as rule 3. The idea is to use the estimate function $f(m)$ as the inclusion criterion. For each level l of SF, markings are ordered in the increasing magnitude of $f(m)$. Any newly obtained marking m can be included into a level if the level is not yet completed. If the level is completed, the marking will be included only if a marking m' with $f(m') > f(m)$ is already included. In this case, the marking m'' with the greatest value of $f(m'')$ will be rejected for exploration and discarded. This marking m'' for each level is WORST(l) shown in Fig. 2.

4) Improving the Selection Criteria: CGS: It is worth noting that DLSS* represents a dramatic decision in terms of rejection of paths by the inclusion rule. It is important that markings representing different paths to achieve the same schedule permutations and futile paths do not compete to be included in SF. In the original A^* algorithm, the test for similar markings solved this problem satisfactorily. However, DLSS* seriously affects the application of such a test since the number of markings stored is limited.

To overcome such a drawback, we propose a branching scheme for DLSS* that avoids the generation of both schedule permutations caused by concurrent transitions and certain nonactive schedules [10]. A schedule is called active if no operation can be completed earlier by

```
(1) while Agenda(M) is not empty
        select an unconsidered element (t,r) in Agenda(M) yielding the
        smallest value of r.
        Obtain the marking M' which results from firing t in M.
        Check whether M' is the goal marking and perform tests for similar
        marking...
        Agenda(M') = ∅;
(2)     for every pair (t',r') enabled in M'
                if r' > 0 then add (t',r') to Agenda(M')
                else if r' = 0 then include (t',r') in Agenda(M') only if the
                        pair (t',r') is after the pair (t,r) in Agenda(M).
                goto (2)
        goto (1)
```

Fig. 3. Branching algorithm.

```
   SF = { M0 }
   Explored = ∅
(1) while SF is not empty do
       Apply Rule 1    ; If the bottom level is exhausted then advance SF
       Remove marking M from SF with the smallest f(M) and depth(M) < top.
       Put M in the Explored List
  (2)  for every new marking M' generated using CGS and HST do
            if M' is the goal marking exit with success.
            else-if M' pass the test for similar markings of cb-NET A* then
               Call Rule 3 for marking M'
               Apply Rule 2 ; If move_at markings have been found then
                                                        advance SF
            goto (2)
       goto (1)
```

Fig. 4. DLSS* algorithm.

altering processing sequences on machines and not delaying any other operation. Obviously, the optimum solution is an active schedule. This branching scheme is called the controlled generator of successors (CGS) and is explained as follows.

Associated with any state of the search graph represented by a marking M, we define a list called $Agenda(M)$ that contains the set of transitions that are enabled at M and still can be fired to generate a new branch M'. Notice that $Agenda(M) \subseteq E$ for the general case with, being E the set of transitions enabled at M.

Agenda contains pairs of the form (t, r) with t an enabled transition and $r = c(M,t)$ the cost of firing t under M, i.e., the elapsed time for t to become enabled in M. The elements in *Agenda* are ordered by the increasing magnitude of r. Initially $Agenda(M) = \{(t,0); \forall t \text{ enabled in } M_0\}$. The standard branching procedure consisting of firing all the enabled transitions at the current marking under exploration is substituted by the procedures shown in Fig. 3.

That is, a transition t enabled at M can be decided not to be fired in order to introduce a tactical delay so other transitions can be fired instead. To avoid the generation of futile paths (nonactive schedules and schedule permutation of concurrent transitions), the transition will not be reconsidered until it becomes enabled again, which means that a change is observed in the input transitions representing resources.

With this strategy, it is possible to achieve a useful search reduction for admissible algorithms such as A^* and B&B. In [16], an experiment showed a reduction of nearly 75% of the reachability tree generated.

5) Summary of the Algorithm: DLSS* results from the integration of CGS, the PN-based A^* algorithm, and the concept of SF. The pseudo-code for the algorithm is shown in Fig. 4.

SF is initialized to the initial marking. The generation of the PN reachability tree follows A^*. The next marking is selected from SF using the heuristic function $f(m)$. The PN branching scheme CGS is integrated within the algorithm to determine the set of transitions to apply. For every new marking, Rule 3 is applied to see if inclusion in SF is possible. Finally, Rule 2 is checked to see if SF must advance.

IV. Experimental Results

The performance in terms of quality of the solution and execution time of DLSS* is controlled in our experiments by the three parameters *Top_level*, *Max_nodes*, *and Move_at*. Tuning guidelines can easily be deduced from the role of each parameter in rules 1–3 of DLSS*.

The initial value of *top_level* (which we refer as *Top_level*) controls the scope of recovery of the algorithm. As we increase *Top_level*, we increase the ability of DLLS* to recover from unpromising paths. This means that DLLS* is more robust against deadlocks or catastrophic scheduling decisions caused by bad estimations of the heuristic function.

$Max\_nodes(l)$ controls the search effort by limiting the scope of selection of the algorithm. Increasing *max_nodes* values, we allow DLLS* to explore a larger number of promising schedules in parallel. In our experiments, we consider the same value for each level which is referred as *Max_nodes*.

Move_at serves as an indicative of how much one can trust the heuristic function. Notice that irrevocable decisions are made when at least *Move_at* markings are explored at *Top_level*. Consequently, a lower setting of *Move_at* with respect to *Max_nodes* is appropriate only if we consider our heuristic function to be a very good one. Higher settings of *Move_at* are needed if we consider that our heuristic function is not very well informed, thus making DLSS* explore a larger number of the promising markings contained in SF.

A. Empirical Study of Computational Costs of DLSS

Since the number of nodes in the search window is limited, we can expect that the computational cost of the algorithm to be polynomial on the number of operations to schedule.

Ten FMS descriptions consisting of three machines, five jobs and four tasks per job were generated. The degree of flexibility (alternate routing) and the operation cost were randomly generated for each problem. Thirty problem sets were obtained by increasing the number of parts per job from 1 to 20, resulting in problems with between 20 to 600 operations.

Each of the 30 problem sets contained 10 problems which were solved by DLSS* ($Top\_Level = 10, Max\_Nodes = 5, Move\_At = 15$). Three parameters obtained from each set are:

1) the average relative difference (Rd) of the makespan obtained for each problem with respect to the Lb $h^*_{\mathrm{RCR}}(M_0)$ (Fig. 5 left);
2) the average number of iterations of the algorithm (number of markings explored) (Fig. 5 right),
3) the average execution time (Fig. 6).

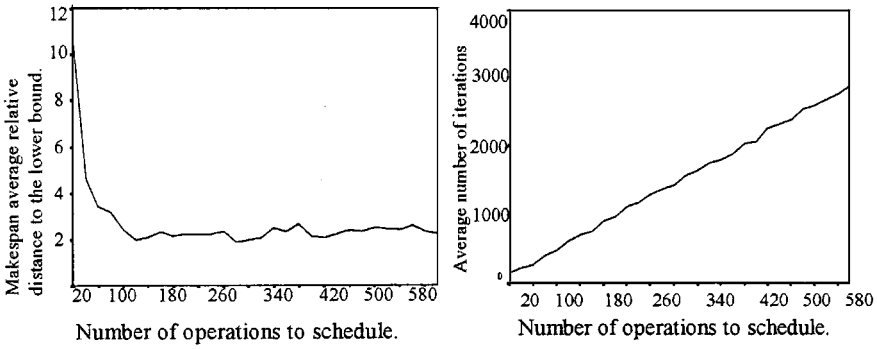

Fig. 5. Evolution of the solution obtained and the number of iterations.

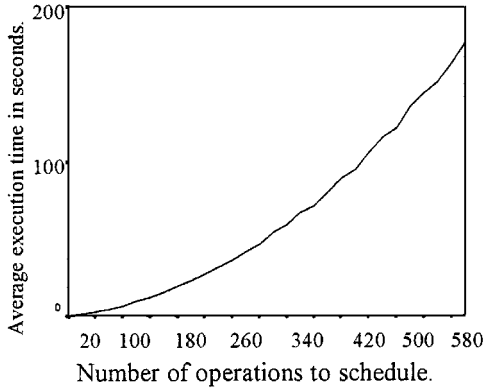

Fig. 6. Evolution of the execution time (in seconds).

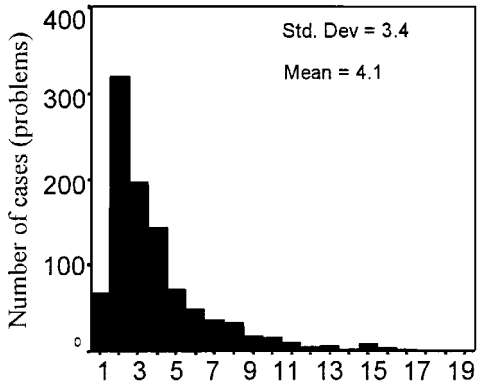

Fig. 7. Relative difference with the lower bound.

From the above results, the algorithm appears to perform a polynomial with the size of the problem. The optimality, expressed as the Rd to the Lb, seems to be constant, i.e., no degradation of the solution quality is observed as we increase the size of the problem. Since Lb is obtained assuming no machine idle time, the results when considering 1–2 parts per job show a greater distance due to a lower production demand. The quadratic behavior of the computational time was also an expected value, as algorithms employed in the simulation module of the system have a computational cost of $O(n^2)$, where n is the number of tokens in the marking.

B. Optimality of the Algorithm

The second experiment was conducted in order to determine the degree of effectiveness of the algorithm for problems which is likely to obtain optimum solutions that are closer to Lb defined by $h^*(M_0)$. A set of 1000 random problems where generated with the following characteristics. The system has three machines, five jobs, and each with a number of tasks between three and six. Fifty percent of jobs have two alternate plans. Eighty-five percent of operations can be performed by more than one machine. Each operation is assigned a random *ground* cost from an uniform distribution $[1 \ldots 100]$. Then the actual cost of each alternative is randomly obtained from a normal distribution of mean *ground* and variance of 15%. A total of ten parts of each job are to be produced in the system.

Each problem was solved by DLSS* with the following settings $(Top\_Level = 10, Max\_Nodes = 5, Move\_At = 5)$. The histogram of the Rd of the makespan obtained with respect to the heuristic Lb $h^*(M_0)$ is shown in Fig. 7. Notice that DLSS* may be reaching the optimal as the hypothetical Lb may be lower than the actual optimum in many cases. We believe this to be responsible for the normal curve being disturbed by a cliff on the right.

Each of the problems was also solved with a heuristic dispatching algorithm that selects the next transition to fire based in the least working remaining time (LWRT) rule, and ties are broken by applying the shortest processing time (SPT) rule. This approach is conceptually equivalent to [1]. The makespan obtained is an average of 18.8% times greater than the heuristic Lb.

Results indicate that the heuristic function quickly directs the search for those FMS with extreme flexibility, where the balanced machine workload can be achieved (which is a desirable designing objective), and there is low variation of operation cost between different alternatives for a *task*.

For problems where a resource is clearly a bottleneck and a balanced workload is not possible, the estimation of $h_{\mathrm{RCR}}(M)$ might be too optimistic. However, in our approach, the search effort is mainly controlled by DLSS*. Again, this is an interesting property, since even for those *ideal* FMS a machine breakdown can create a temporarily imbalance of the system. In this situation, $h_{\mathrm{RCR}}(M)$ also becomes too optimistic and forces DLSS* to increase the search within SF whilst ensuring that the search will advance toward a solution.

C. Comparison With Previous Work

The following experiment was conducted in order to compare DLSS* with different PN search algorithms.

1) DLSS$^*(10, 15, |T|/2)$, where $|T|$ is the number of transitions in the system.
2) Nonadmissible A^* with a DF heuristic function [21]: $h(m) = w \cdot A \cdot E - w \cdot A \cdot depth(M)$, where A is the mean operating cost of all operations and E is the nominal number of transition firings from the initial marking to a goal marking. Because there is no concept of concurrency in the total operating cost, w explains the extent of reduction of this expression.

TABLE I
DESCRIPTIVE STATISTICS

	%Problems improved	Relative difference of makespans			Std. Dev.
		Minimum	Maximum	Mean	
*A**	6.3%	-15.0	94.8	22.8	17.1
B&B	5.4%	-15.9	125	26.0	19.6
Beam	15.6%	-23.0	57.6	8.7	10.2

3) Incomplete B&B procedure: the search stops when the number of nodes expanded at least doubles the average nodes explored by DLSS*. We employ h_{RCR} as Lb for pruning purposes. For selecting the next branch, a combination of the LWRT and SPT rules is adopted. Such an algorithm can be considered a sophistication of the B&B approach of [1].
4) BS with a *beam-width* of 80; such a setting explores the same number of nodes as DLSS* by average. The algorithm uses the same heuristic function of DLSS* to select the beams, i.e., h_{RCR}. We believe such an approach is similar to the *limited-expansion* A^* proposed in [19] and the BS approach of [17].

The problem data of 1000 FMS descriptions were randomly obtained as follows. The number of jobs is uniformly obtained within the range $[5 \ldots 10]$, and the number of resources in the system is calculated as the number of jobs divided by two. Twenty-five percent, 50%, and 25% of jobs have three, two, and one process plans, respectively, and 65% of tasks have alternate routing and multiple resources may be required to perform a task. Each operation is assigned a random *ground* cost from an uniform distribution $[1 \ldots 100]$. Then the actual cost of each alternative is randomly obtained from a normal distribution of mean *ground* and variance of 33%.

The four algorithms solved each problem. In the case of A^*, we tried different settings for w in order to obtain a proper comparison. However, we found difficulties in choosing an adequate value for w. Setting w to $1/m$ resulted in the majority of the problems to be intractable. On the other hand, the $3/m$ results were too optimistic and the quality of the solutions obtained were too poor to be compared with DLLS*. We finally opted for $w = 2/m$, although this setting resulted in 24% of problems not being solved in a reasonable amount of time.

The makespan obtained by DLSS* was taken as the reference and compared with the rest of the solutions obtained. The comparison results are shown in Table I.

The first column stands for the percentage of problems for which the algorithm obtains a better makespan than DLSS*. The other columns show the descriptive statistics for the % Rd of the makespan obtained by each problem with respect to the solution makespan provided by DLSS*.

We experienced tuning problems with the A^* algorithm. The variance of the search effort was high and made a proper comparison difficult. Actually, 24% of executions were halted due to memory problems. Results with the B&B are the worst due to the fact that the B&B bases its strategies in chronological optimization of a first solution, rather than in accurate local improvement. When DLSS* is compared with BS, a less dramatic difference is obtained although it is still noticeable. Since both approaches follow a similar optimization philosophy, such difference is explained in terms of the backtracking recovery capacity of DLSS*.

D. Comparison With Some Benchmark Problems

Finally, we solved several concrete benchmark problems proposed in two papers. Table II shows the comparison results for three FMS problems in [5]. This paper implements A^* with a nonadmissible heuristic function. The results show considerable improvement. Besides the fact that h_{RCR} results in a better heuristic function, DLSS* allows us to increase the search effort without falling in a *breadth-first* (BF) search. In [5], authors reported that no further improvement of the solution could be made since relaxation of the DF component of the heuristic function resulted in the algorithm not being able to find the solution in a reasonable amount of time.

TABLE II
COMPARISON WITH BENCHMARKS PROPOSED IN [5]

Problem	Reported	DLSS*
Lee 94 (a)	426	329
Lee 94 (b)	298	254
Lee 94 (c)	273	237

TABLE III
COMPARISON RESULTS WITH [20]

Problem	Makespan			Number of markings explored		
	BF	*BT-BF*	*DLSS**	*BF*	*BT-BF*	*DLSS**
1	58	62	58	3437	1687	431
2	100	104	100	9438	8045	856
3	134	148	134	23092	18875	1204

The results for three problems presented in [20] are shown in Table III. The optimal solution is given by a pure Best-First technique, while [20] proposes a hybrid search technique between a B&B and BF. Results show that DLSS* finds the optimum solution with considerably less search effort (measured in terms of the number of markings explored).

V. CONCLUSION

The combination of PN modeling as a representation formalism and AI-based heuristic search methodologies has been studied in this paper.

A hybrid PN-based scheduling algorithm called DLSS* has been presented, whose preliminary study shows it to be a promising alternative to overcome the difficulties encountered with previous approaches. DLSS* is the further development of the works studied in [13] and [15]. Each of the methodologies within DLSS* concentrates on different aspects of the search space and aims to reduce the search effort while maximizing admissibility: the PN-based A^* together with the PN admissible heuristic function implement a powerful optimization strategy. CGS helps to identify successful alternatives and discard futile ones, and finally the SF behavior rules the strategy by avoiding exponential generation of markings and forcing the search to progress to a solution.

This results in an algorithm that is able to achieve a useful degree of optimality without experimenting exponential cost. When compared with other approaches, experimental tests suggest the superiority of our approach.

REFERENCES

[1] B. Abdallah, H. A. ElMaraghy, and T. ElMekkawy, "An efficient search algorithm for deadlock-free scheduling in FMS using Petri nets," *Proc. IEEE Int. Conf. on Robotics and Automation*, pp. 1793–1798, 1998.
[2] A. Inaba, F. Fujiwara, T. Suzuki, and S. Okuma, "Timed Petri net based scheduling for mechanical assembly—Integration of planning and scheduling," *IEICE Trans. Fundamentals Electron. Commun. Comput. Sci.*, vol. E-81A, pp. 615–625, 1998.
[3] M. D. Jeng, W. D. Chiou, and Y.-L. Wen, "Deadlock-free scheduling of flexible manufacturing systems based on heuristic search and Petri net structures," in *Proc. 28th Int. Conf. Systems, Man and Cybernetics*, 1998, pp. 26–31.
[4] R. E. Korf, "Real—Time heuristic search.," *Artif. Intell.*, vol. 42, pp. 189–211, 1990.
[5] D. Y. Lee and F. Dicesare, "Scheduling flexible manufacturing systems using Petri nets and heuristic search," *IEEE Trans. Robot. Automat.*, vol. 10, pp. 123–132, 1994.

[6] S. Lloyd, H. Yu, and N. Konstants, "FMS scheduling using Petri net modeling and Brach & Bound search," *Proc. IEEE Int. Symp. Assembly and Task Planning*, pp. 141–146, Aug. 1995.
[7] T. Murata, "Petri nets: Properties, analysis and applications," *Proc. IEEE*, vol. 77, pp. 541–580, 1989.
[8] P. S. Ow and T. E. Morton, "Filtered beam search in scheduling," *Int. J. Prod. Res.*, vol. 26, pp. 35–62, 1988.
[9] J. P. Heuristics, *Intelligent Search Strategies for Computer Problem Solving*. Boston, MA: Addison-Wesley, 1984.
[10] M. Pinedo, *Scheduling: Theory, Algorithms and Systems*. Englewood Cliffs, NJ: Prentice-Hall, 1995.
[11] J. M. Proth and X. L. Xie, *Petri Nets: A Tool for Design and Management of Manufacturing Systems*. New York: Wiley, 1996.
[12] A. Reyes, H. Yu, G. Kelleher, and S. Lloyd, "Integrating Petri nets and hybrid heuristic search for scheduling FMS," *Computers in Industry*, vol. 47, pp. 123–138, 2002.
[13] A. Reyes, H. Yu, and G. Kelleher, "Applying new search methodologies for scheduling FMS using PN," in *Proc. 16th Workshop of the UK Planning and Scheduling*, Huddersfield, U.K., Sept. 1998.
[14] ——, "Petri nets, heuristic search and natural evolution: A promising scheduling algorithm for job shop systems," in *Proc. 3rd Int. Symp. Intelligent Industrial Automation*, Genoa, Italy, June 1999.
[15] ——, "Advanced scheduling methodologies for flexible manufacturing systems using Petri nets and heuristic search," *Proc. IEEE Int. Conf. Robotics and Automation. ICRA200*, pp. 24–28, Apr. 2000.
[16] ——, "A PN reachability graph branching scheme with application to the scheduling of flexible manufacturing systems," in *Proc. IFAC Conf. Control Systems Design*, Bratislava, June 2000, pp. 19–22.
[17] H. M. Shih and T. Sekiguchi, "A timed Petri net and beam search based on-line FMS scheduling system with routing flexibility," *Proc. IEEE Int. Conf. Robotics Automat.*, pp. 2548–2553, 1991.
[18] S. Shukla and F. F. Chen, "The state of the art in intelligent real-time FMS control: A comprehensive survey," *J. Intell. Manufact.*, vol. 7, pp. 441–455, 1996.
[19] T. Sun, C. Cheng, and L. Fu, "A Petri net based approach to modeling and scheduling for an FMS and a case study," *IEEE Trans. Ind. Electron.*, vol. 41, pp. 593–601, 1994.
[20] H. H. Xiong and M. Zhou, "Scheduling of semiconductor test facility via Petri nets and hybrid heuristic search," *IEEE Trans. Semiconduct. Manufact.*, vol. 11, pp. 384–393, 1998.
[21] S. J. Yim and D. Y. Lee, "Multiple objective scheduling for flexible manufacturing systems using Petri nets and heuristic search," in *IEEE Int. Conf. Systems, Man, & Cybernetics*, 1996, pp. 2984–2989.
[22] C. Zhou and F. DiCesare, *Petri Net Synthesis for Discrete Event Control of Manufacturing Systems*. Boston, MA: Kluwer, 1993.

An Adaptive Iterative Learning Control Algorithm With Experiments on an Industrial Robot

Mikael Norrlöf

***Abstract*—An adaptive iterative learning control (ILC) algorithm based on an estimation procedure using a Kalman filter and an optimization of a quadratic criterion is presented. It is shown that by taking the measurement disturbance into consideration the resulting ILC filters become iteration-varying. Results from experiments on an industrial robot show that the algorithm is successful also in an application.**

***Index Terms*—Disturbance rejection, iterative learning control, robot application, synthesis.**

I. Introduction

Iterative learning control (ILC) is a well-established method for control of repetitive processes. It is in general considered to be an approach for trajectory tracking, and this is how it is usually described in the literature (see, for example, the surveys [1]–[3]). In this paper, we will use ILC in a different setting by applying ILC for disturbance rejection (see also [4]). In Section V, we will show how we can apply the results to a standard tracking application for ILC. Disturbance rejection aspects of ILC have also been covered earlier in, e.g., [5]–[7], where disturbances such as initial state disturbances and measurement disturbances are addressed.

In Fig. 1, the structure used in the disturbance rejection formulation to ILC is shown as a block diagram.

The goal of ILC is to iteratively find the input to a system such that some error is minimized. In the disturbance rejection formulation, the goal becomes to find an input $u_k(t)$ such that the output $z_k(t)$ is minimized. If the system is known and invertible, and the disturbance $d_k(t)$ is known, then the obvious approach would be to filter $d_k(t)$ through the inverse of the system and use the resulting $u_k(t)$ as a control input. This means that the optimal input looks like

$$u_k(t) = -(G^0)^{-1} d_k(t).$$

Different aspects of this approach to ILC is considered in the paper. Results from using the methods on an industrial robot are also presented.

II. A State Space-Based Approach to ILC

A. Matrix Description of the System

An ILC system is characterized by the fact that it is only defined over a finite interval of time. If the sampling time is equal to one, this means that $0 \leq t \leq n-1$. This is also the reason why it is possible to write the system description in matrix form as

$$\begin{aligned} \boldsymbol{z}_k &= \boldsymbol{G}^0 \boldsymbol{u}_k + \boldsymbol{d}_k \\ \boldsymbol{y}_k &= \boldsymbol{z}_k + \boldsymbol{n}_k \end{aligned} \tag{1}$$

with

$$\boldsymbol{d}_{k+1} = \boldsymbol{d}_k + \boldsymbol{\Delta}_{d_k} \tag{2}$$

Manuscript received May 31, 2001. This paper was recommended for publication by Associate Editor Y. Xu and Editor A. De Luca upon evaluation of the reviewers' comments. This work was supported by VINNOVA's Center of Excellence ISIS at Linköpings universitet, Linköping, Sweden.

The author is with the Department of Electrical Engineering, Linköpings Universitet, SE-581 83 Linköping, Sweden (e-mail: mino@isy.liu.se).

Publisher Item Identifier S 1042-296X(02)02653-8.