ACKNOWLEDGMENT

The authors would like to thank Dr. J. Boyd at the Visual Computing Laboratory, University of California, San Diego, CA, for providing gait data and giving invaluable advice.

REFERENCES

[1] M. S. Nixon *et al.*, "Automatic gait recognition," in *BIOMETRICS—Personal Identification in Networked Society*, A. Jain, R. Bolle, and S. Pankanti, Eds. New York: Kluwer, Jan. 1999, ch. 11, pp. 231–249.

[2] R. B. Davis and P. A. DeLuca, "Clinical gait analysis—Current methods and future directions," in *Human Motion Analysis*, G. F. Harris and P. A. Smith, Eds. Piscataway, NJ: IEEE Press, 1997, ch. 2, pp. 17–42.

[3] G. Johansson, "Visual perception of biological motion and a model for its analysis," *Perception Psychophys.*, vol. 14, no. 2, pp. 201–211, 1973.

[4] C. Cedras and M. Shah, "A survey of motion analysis from moving light displays," in *Proc. IEEE Conf. Comput. Vision Pattern Recognit.*, Seattle, WA, June 1994, pp. 214–221.

[5] J. E. Cutting and L. T. Kozlowski, "Recognizing friends by their walk: Gait perception without familiarity cues," *Bull. Psychonom. Society*, vol. 9, no. 5, pp. 353–356, 1977.

[6] S. A. Niyogi and E. H. Adelson, "Analysis and recognizing walking figures in xyt," in *Proc. IEEE Conf. Comput. Vision Pattern Recognit.*, Seattle, WA, June 1994, pp. 469–474.

[7] J. Little and J. Boyd, "Recognizing people by their gait: The shape of motion," *Videre*, vol. 1, no. 2, pp. 1–32, 1998.

[8] H. Murase and R. Sakai, "Moving object recognition in eigenspace representation: Gait analysis and lip reading," *Pattern Recognit. Lett.*, vol. 17, pp. 155–162, 1996.

[9] P. S. Huang, C. J. Harris, and M. S. Nixon, "Recognizing humans by gait via parametric canonical space," *J. Artif. Intell. Eng.*, vol. 13, no. 4, pp. 359–366, Oct. 1999.

[10] ——, "Human gait recognition in canonical space using temporal templates," *Proc. Inst. Elect. Eng.,—Vision, Image, Signal Process.*, vol. 146, no. 2, pp. 93–100, Apr. 1999.

[11] ——, "A statistical approach for recognizing humans by gait using spatial–temporal templates," in *Proc. Int. Conf. Image Process.*, vol. 3. Chicago, IL, Oct. 1998, pp. 178–182.

[12] M.-P. Dubuisson and A. K. Jain, "Contour extraction of moving objects in complex outdoor scenes," *Int. J. Comput. Vision*, vol. 14, no. 6, pp. 83–105, 1995.

[13] H. Bulthoff, J. Little, and T. Poggio, "A parallel algorithm for real-time computation of optical flow," *Nature*, vol. 337, pp. 549–553, Feb. 1989.

[14] P. S. Huang, C. J. Harris, and M. S. Nixon, "Comparing different template features for recognizing people by their gait," in *Proc. Ninth British Machine Vision Conf.* Southampton, U.K.: BMVA, Sept. 1998, vol. 2, pp. 639–648.

[15] K. Fukunaga, *Introduction to Statistical Pattern Recognition*, 2nd ed. New York: Academic, 1990.

[16] C. M. Bishop, *Neural Networks for Pattern Recognition*. Oxford, U.K.: Oxford Univ. Press, 1996.

GA-Based Discrete Dynamic Programming Approach for Scheduling in FMS Environments

Jian-Bo Yang

***Abstract*—This paper presents a new genetic algorithm (GA)-based discrete dynamic programming (DDP) approach for generating static schedules in a flexible manufacturing system (FMS) environment. This GA-DDP approach adopts a sequence-dependent schedule generation strategy, where a GA is employed to generate feasible job sequences and a series of discrete dynamic programs are constructed to generate legal schedules for a given sequence of jobs. In formulating the GA, different performance criteria could be easily included. The developed DDP algorithm is capable of identifying locally optimized partial schedules and shares the computation efficiency of dynamic programming. The algorithm is designed in such a way that it does not suffer from the state explosion problem inherent in pure dynamic programming approaches in FMS scheduling. Numerical examples are reported to illustrate the approach.**

***Index Terms*—Dynamic programming, flexible manufacturing system (FMS), genetic algorithms (GAs), heuristics, scheduling.**

I. INTRODUCTION

Development of job processing schedules in a flexible manufacturing system (FMS) environment is a complex task. The number of legal (feasible) schedules increases exponentially with the increase of the number of jobs that must be processed, the number of operations required by each job, and the number of flexible workstations that can perform more than one operation. This exponential growth makes it very difficult or even impossible to use mathematical programming or exhaustive search approaches for finding global optimal schedules in terms of any performance measure for problems of practical complexity [2]–[4], [6].

The applications of dynamic programming (DP) to FMS scheduling have been reported in literature in recent years, for example automated guided vehicle scheduling [3], [10] and production sequencing [11]. For small-sized problems, optimal schedules could be generated using a pure DP approach. However, the computational requirements for a pure DP approach are impracticably demanding for a large-sized problem [11]. In such circumstances, one tends to rely on heuristic or adaptive search techniques to find good quality schedules instead of global optimal schedules.

Recent years have seen a growing research interest in applying GA-based approaches to deal with FMS scheduling problems [5], [9], [12], [14], [16]. GA-based approaches can effectively combine the prespecified problem processing knowledge with rote-learned knowledge to generate good quality schedules [9]. However, a pure GA-based approach requires specially designed GA operators and is liable to generate illegal schedules. As such it may not be directly used as a general schedule generation tool [9].

Holsapple *et al.* [9] investigated a hybrid scheduler combining GA with traditional heuristics. In the hybrid scheduler, GA was used to generate job sequences, based on which heuristics were then applied to produce legal schedules. As discussed below, there are several benefits of employing such a sequence-dependent schedule generation strategy,

Manuscript received October 15, 1996; revised March 28, 2001. This paper was recommended by Associate Editor C. Hsu.

The author is with the Manchester School of Management, University of Manchester Institute of Science and Technology (UMIST), Manchester, U.K. (e-mail: jian-bo.yang@umist.ac.uk).

Publisher Item Identifier S 1083-4419(01)05974-X.

though it suffers from not guaranteeing to generate a global optimal schedule even for a small-sized problem.

First of all, with any tree search methods for generating job sequences, sequence generation and schedule generation must proceed simultaneously and it is generally not feasible to generate all possible sequences of n jobs ($n!$ possibilities) [6]. In this case, a schedule generation algorithm may be invoked with an incomplete sequence. With the GA approach, however, a schedule generation algorithm is always invoked with only complete job sequences and hence no additional effort is required for unnecessary algorithm invocations. Secondly, as the evaluation is based entirely on actual job sequences with no dependence on estimates, it could be expected that the quality of subsequent sequences generated based on the evaluation of existing sequences would be better as well. Third, with the GA approach, it is possible to carry out schedule generation in a parallel processing environment as a generation of job sequences can be generated simultaneously [9]. Finally, the GA approach can be readily extended to deal with multiple objective scheduling problems [2], [5], [7].

This paper presents a new GA-based discrete dynamic programming (GA-DDP) approach for generating static schedules in a FMS environment. This new GA-DDP approach adopts the sequence-dependent schedule generation strategy and thus shares its benefits as well as its drawbacks. It also uses GA to generate feasible job sequences but adopts a series of discrete dynamic programs (DDPs) to generate legal schedules for a given sequence of jobs. The dynamic programs are specially designed, so that they can identify locally optimized partial schedules and benefit from the computation efficiency of DP while the state explosion problem inherent in pure DP approaches in FMS scheduling is avoided.

The recent literature survey shows that little research has been conducted to combine DDP with GA for FMS scheduling. This research is intended to draw more attention to the development of hybrid scheduling approaches combining traditional algorithmic procedures with heuristic or adaptive search techniques. It is believed that this could provide an opportunity to generate a new family of promising approaches for large scale and multiple objective scheduling in FMS environments.

After a brief background discussion, a new FMS scheduling heuristic and its DP realization are investigated in detail. In Section IV, the GA-DDP approach is fully explored and a methodology for implementing the approach is also discussed. Section V is devoted to a numerical study. The paper concludes in Section VI.

II. Background

A. Scheduling in a FMS Environment

A typical flexible manufacturing system is composed of multiple workstations (or machine centers), a material handling system, and a loading-unloading station. Such systems are aimed at facilitating the efficient processing of parts with the low- or medium-volume range. In such a system, a workstation is capable of performing one or more manufacturing operations and a given operation may be performed on more than one workstation. It is often the case that operations required by a job need to be performed in a particular order called "operation precedence requirement." The problem of interest is how to schedule the operations of jobs to workstations to best meet the organizational objectives [1].

The scheduling problem to be discussed in this paper is defined as follows. Suppose there are n jobs, each of which may require several operations that can be processed on m workstations. The operation precedence requirement for the operations of each job and the processing time of each operation on a workstation are prescribed. The transfer time from one workstation to another and the due date for each job are also given. It is possible that the transfer of jobs from some workstations to others is prohibited. It is assumed that all jobs are available at time zero and only one operation can be performed on a workstation at any time. It is also assumed that the processing of an operation must never be interrupted once it starts.

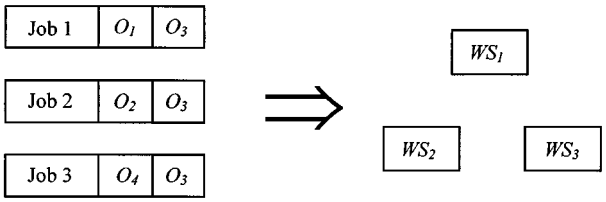

Fig. 1. Example FMS scheduling problem.

TABLE I
PROCESSING TIMES FOR EXAMPLE FMS SCHEDULING PROBLEM

	Operations			
Workstation	O_1	O_2	O_3	O_4
WS_1	20	20	∞	39
WS_2	26	∞	52	41
WS_3	∞	30	38	∞
Average	23	25	45	40

TABLE II
JOB RELATED DATA FOR EXAMPLE FMS SCHEDULING PROBLEM

Job No.	Required operations	ATPT	DD	Tardiness cost
J_1	O_1, O_3	68	94	1
J_2	O_2, O_3	70	100	1
J_3	O_4, O_3	85	101	1

A schedule consists of the assignment of the operations of all jobs to workstations. A schedule is said to be legal if it satisfies the following three conditions:

1) the precedence requirement for the operations of any job is followed;
2) each workstation processes only one operation at a time and the processing is not interrupted;
3) the operations of the same job are only assigned to different workstations where the transfer as required is not prohibited.

An optimal schedule is the one that can optimize certain performance objectives. Typical objectives include the optimization of total makespan (TMS), total flow time (TFT), and total tardiness cost (TTC). TMS is the actual time required to complete all jobs in question. TFT is the sum of the completion times for all jobs. TTC is the sum of tardiness costs for all jobs, where the tardiness cost for a job is the multiplication of its unit tardiness cost and the absolute difference between its completion time and its due date, given that the former is larger than the latter.

Fig. 1 shows a simple FMS scheduling problem with three workstations, four different operations in total, and three jobs, each of which requires two operations. Note that in this example a job may be transferred from one workstation to any of the others.

Table I shows the processing time of each operation on a workstation where ∞ means that an operation cannot be performed on that workstation. Table II shows the operations required by each job with the precedence requirements, the average total processing time (ATPT) of a job, the due date (DD) of a job, and the tardiness cost per unit time. This problem was examined in [9]. The scheduling problem is how to assign the six operations of the three jobs onto the three workstations so that certain objectives can be optimized.

B. Algorithms and Heuristics

To generate legal schedules, algorithmic methods may be used, most of which are based on tree search. In such methods, all legal schedules are explicitly or implicitly enumerated and evaluated. Such methods are useful to deal with small-sized problems. In the above example problem, for instance, there are only 5760 legal schedules in total [9].

A legal schedule must satisfy the three conditions as discussed in Section II-A. For example, a schedule with the sequence of operations $\langle(J_1, O_1,\ (J_1, O_3),\ (J_2, O_3),\ (J_2, O_2),\ (J_3, O_4),\ (J_3, O_3)\rangle$ is illegal as it violates the precedence requirement for J_2. Let $(J_i, O_j)/(WS_k\colon t_l,\ t_h)$ denote that operation j of job i is scheduled on workstation k from time t_l to time t_h. Then the following schedule is legal:

$$\begin{aligned} S_1\colon\ & \langle(J_1, O_1)/(WS_2\colon 0, 26),\ (J_1, O_3)/(WS_2\colon 26, 78),\\ & (J_2, O_2)/(WS_3\colon 0, 30),\ (J_2, O_3)/(WS_3\colon 30, 68),\\ & (J_3, O_4)/(WS_1\colon 0, 39),\ (J_3, O_3)/(WS_3\colon 68, 106)\rangle. \quad (1)\end{aligned}$$

The above schedule reads that jobs 1 and 2 are processed on workstations 2 and 3, respectively. The first operation O_4 of job 3 is processed on workstation 1 and finally the last operation of job 3 is processed on workstation 3 once job 2 is finished. The TMS of the above schedule is 106 time units (TU), the TFT is $78 + 68 + 106 = 252$, and the TTC is $0 + 0 + 1 \times (106 - 101) = 5$. A tree search algorithm may attempt to generate and evaluate all the 5760 legal schedules either explicitly or implicitly and then choose the best schedule that minimizes either TMS, TFT, or TTC.

It has been realized, however, that in an FMS environment the time required to find a globally optimized schedule using an algorithm increases exponentially as the problem size increases. This means that an algorithmic solution procedure searching for optimal solutions is impractical for problems of any realistic size [2], [4]. Many production scheduling problems are therefore approached using heuristic methods. Instead of producing optimal schedules, heuristic methods attempt to search for legal schedules with good quality in general.

Many heuristics have been put forward and used for real-world scheduling problems. When scheduling jobs on a single machine, for example, the mean flow time is minimized by sequencing the shortest processing time (SPT) job first, where the mean flow time is the total flow time divided by the number of jobs [1]. In a FMS environment, this SPT first rule could be interpreted as the shortest average processing time (SAPT) first rule in order to sequence jobs [9]. In the above example, this SAPT rule results in the following job sequence $\langle J_1, J_2, J_3\rangle$. For this sequence, different heuristics may be used to schedule the operations of each of the three jobs to the three workstations.

One intuitively simple heuristic is to allocate the best available resource to the first operation of the first job in the sequence first. Then the best of the remaining available resources is allocated to the second operation of the first job next, and so on until all operations of the first job are scheduled. The process is repeated for each of the jobs in the sequence.

The above heuristic method is easy to implement as no complex search is required to generate a schedule. Suppose the minimization of total makespan is selected as the performance measure. Given the job sequence $\langle J_1, J_2, J_3\rangle$, the above heuristic requires that the first operation (O_1) of job 1 (J_1) should be scheduled first. From Table I, O_1 can be performed on workstation 1 or 2. Thus O_1 of J_1 should be scheduled to workstation 1 from time 0 to 20 so that it can be completed as soon as possible. O_3 of J_1 is scheduled next which can be processed on workstation 2 or 3. Thus O_3 of J_1 should be scheduled to workstation 3 from time 20 to 58 for the quickest completion. The operations of the other two jobs can be scheduled in the same way. The schedule obtained by using the SAPT first rule and the simple heuristic is given by

$$\begin{aligned} S_2\colon\ & \langle(J_1, O_1)/(WS_1\colon 0, 20),\ (J_1, O_3)/(WS_3\colon 20, 58)\\ & (J_2, O_2)/(WS_1\colon 20, 40),\ (J_2, O_3)/(WS_2\colon 40, 92)\\ & (J_3, O_4)/(WS_1\colon 40, 79),\ (J_3, O_3)/(WS_3\colon 79, 117)\rangle. \quad (2)\end{aligned}$$

In the above schedule, TMS, TFT, and TTC are 117, 267, and 16, respectively.

In the next section, a new heuristic is first discussed for reducing the search space of legal schedules. This heuristic is realized by constructing a DDP for scheduling the operations of one job. A series of similar DDPs are constructed for scheduling a given sequence of jobs. The above example will be used to demonstrate the realization.

III. New Heuristic and Its Dynamic Programming Realization

A. New Heuristic for Job Sequence Dependent Scheduling

In FMS, an operation may be performed on more than one workstation and a workstation may be capable of processing more than one operation. In light of this feature, a new heuristic is put forward as follows. Given a sequence of jobs, the operations of the first job are schedules. In doing so, all the other jobs are temporarily disregarded and all the possible ways of scheduling the operations of the first job are exhaustively examined for initially available resources. The aim is to find a best partial schedule that has the minimum up-to-the-minute makespan. If there are m_1 $(m_1 \geq 1)$ workstations capable of performing the last operation of the first job, then for each such workstation a best partial schedule is found assuming that the last operation is performed on this workstation. Thus m_1 locally optimized partial schedules are obtained after the first job has been scheduled.

Based on each of the m_1 partial schedules obtained, the second job is scheduled by using the remaining resources. In doing so, the third and onward jobs in the sequence are not considered and the same principle for scheduling the first job is used. For every workstation capable of processing the last operation of the second job, m_1 partial schedules are thus generated with both job 1 and job 2 scheduled. From the m_1 partial schedules obtained, the one with the minimum up-to-the-minute makespan is chosen as the best partial schedule for the workstation. If there are m_2 $(m_2 \geq 1)$ workstations capable of processing the last operation of the second job, m_2 new partial schedules will be generated after both job 1 and job 2 have been scheduled.

From each of the m_2 new partial schedules obtained, the third job is in turn scheduled in the same way as for scheduling the second one. The procedure is repeated for the rest of the jobs in the sequence until the last job is scheduled. If there are m_l workstations capable of processing the last operation of the last job, m_l complete schedules will be generated, from which the schedule with the smallest makespan is picked up.

Note that in the above heuristic it is assumed that a job may be scheduled independently of others. It will be shown in Sections IV-A and B and V-D that the number of schedules directly examined using the new heuristic only increases linearly with the increase of jobs. For a given sequence, the implementation of the above heuristic can dramatically reduce search effort while many good schedules are selected and assessed. This is because all the possible ways of independently scheduling the operations of each job are examined with the attempt to find a best partial schedule.

However, the heuristic does not result in enumerating all the possible ways of scheduling all jobs in a given sequence. Rather, it only leads to the examination of a small portion of good schedules. Suppose there are k jobs in a sequence and each job has n operations that can be performed on any of m workstations. Then there are altogether $(m^{n+1})^k$

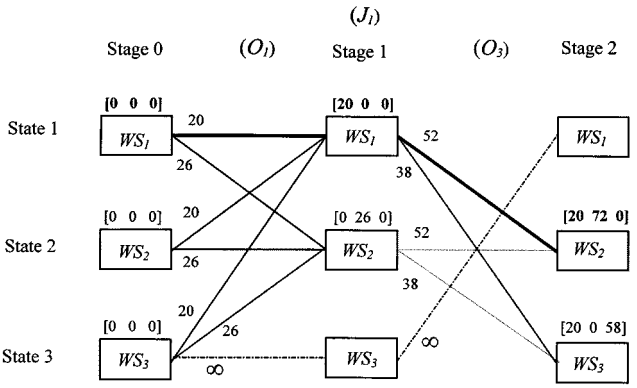

Fig. 2. Scheduling the operations of job 1.

possible schedules for this sequence. Using the above heuristic, only $k \times m^{n+2}$ of these schedules are directly selected for examination although many other schedules are indirectly assessed as well. These examined and evaluated schedules are expected to represent a typical set of good schedules. Without full enumeration, however, the heuristic does not guarantee to find the best schedule for a given job sequence.

B. Discrete Dynamic Programming for Scheduling the Operations of One Job

In this section, a DDP is formulated to realize the above new heuristic. Readers not familiar with DDP may refer to [8] or any other textbooks introducing DDP. The example problem as shown in Tables I and II is used to demonstrate the realization. The job sequence $\langle J_1, J_2, J_3 \rangle$ is taken for example in this section in order to compare the new heuristic with the simple heuristic as discussed in Section II-B which resulted in the schedule S_2.

The terms in DDP are defined first with regard to a scheduling problem in FMS. A state is defined as a workstation and a stage as the process of transferring a job from a workstation to a next (successor) workstation if necessary and processing an operation on the latter workstation. The precedence order for the operations of a job determines the order of stages. An action is defined as which workstation performs the next operation. A stage return is defined as the sum of the time required to transfer a job from a workstation to the next and the time used to process an operation on the latter workstation. Finally, the value of a state is defined as the minimum up-to-the-minute makespan. A state will be represented by a vector showing the latest finishing times of all workstations given that the last operation is processed on this workstation (state). This vector characterizes the remaining resources at a state and may be referred to as a characteristic vector of a state, or simply a state vector.

Suppose all workstations are available initially. Using the new heuristic, the first job of the sequence, that is job 1, is scheduled first. The first operation (O_1) of job 1 (or J_1) is scheduled first, followed by the second operation (O_3). Based on the data given in Tables I and II, the above definitions can be used to construct a DDP for scheduling job 1, as shown in Fig. 2.

In Fig. 2, three stages are formulated. Stage 0 stands for the initial conditions of the FMS, stage 1 for the processing of the operation O_1 and stage 2 for the processing of the operation O_3. A rectangle represents a state (workstation or WS) and there are three states in each column or at each stage. A (dark solid, solid, or dotted) line connecting two rectangles of two adjacent stages denotes an action, which indicates that a job (job 1) is transferred from the left workstation to the right one and the current operation is processed on the right workstation. A number associated with a line is a stage return incurred when an action represented by the line is taken. A row vector above a rectangle is a state vector. If an operation cannot be processed on a workstation, then there is no line connected to the state at the stage. If a job cannot be transferred from a workstation to another, then they are not connected at all. For the purpose of implementing the DP on a computer, however, no connection could be illustrated by a dot-and-dash line associated with an infinite stage return.

In Fig. 2, for instance, at state 1 of stage 0 stands a rectangle for workstation 1 (WS_1). The state vector is [0, 0, 0] showing that all the three workstations are available at time 0. There are two lines connecting this state to states 1 and 2 of the next stage (stage 1). The first line means that the operation O_1 could be processed on workstation 1. The second line means that job 1 could be transferred from workstation 1 to workstation 2 and then O_1 is processed on workstation 2. The number 20 above the first line is the time required to process O_1 on workstation 1, and 26 below the second line is the sum of the time needed for transferring job 1 from workstation 1 to workstation 2 and the time required to process O_1 on workstation 2. Note that the transfer time between any workstations is assumed to be zero in this example.

At state 2 of stage 2 stands a rectangle for workstation 2 (WS_2). The state vector [20, 72, 0] means that workstations 1, 2, and 3 are available at times 20, 72, and 0, respectively. The vector also shows that the minimum up-to-the-minute (stage 2) makespan is 72 if O_3 of J_1 is completed on WS_2. It will be explained later how this state vector is generated.

In Fig. 2, a path from a state of stage 0 to a state of stage 2 represents a legal partial schedule for job 1. For example, a path ($WS_1 \rightarrow WS_1 \rightarrow WS_2$) represents a partial schedule indicating that based on the initial condition [0, 0, 0] at WS_1 the operation O_1 of job 1 (J_1) is processed on WS_1 and the operation O_3 on WS_2. This partial schedule for job 1 can be precisely recorded by S_{31} as follows:

$$S_{31}\colon \langle (J_1, O_1)/(WS_1\colon 0, 20), (J_1, O_3)/(WS_2\colon 20, 72), \ldots \rangle. \quad (3)$$

Let us examine how to generate a partial schedule using the dynamic programming network in order to minimize the makespan for processing job 1. First of all, the operation precedence requirement for the operations of job 1 is catered for by defining the three stages. The three machines are all assumed to be available at time zero. This has been accounted for by defining the state vectors of the three states of stage 0 to be [0, 0, 0].

The operation O_1 of job 1 is scheduled first. From Table I and Fig. 2, O_1 could be processed on WS_1 or WS_2. In the first case where O_1 is processed on WS_1, if job 1 is located at WS_2 or WS_3, it can be transferred to WS_1. In the second case where O_1 is processed on WS_2, if job 1 is located at WS_1 or WS_3, it can be transferred to WS_2. If O_1 is processed on WS_1, it can be completed as early as time 20, resulting in a state vector [20, 0, 0] at state 1 of stage 1. If O_1 is processed on WS_2, it can be completed as early as time 26, resulting in a state vector [0, 26, 0] at state 2 of stage 1.

Then, the operation O_3 of job 1 is scheduled, which can be processed on WS_2 or WS_3. In either case, the aim is to find a best plan with the minimum up-to-stage 2 makespan. In case 1 where O_3 is processed on WS_2, there exist two possible ways of scheduling O_3. First, if O_1 was completed on WS_1, job 1 can be transferred from WS_1 to WS_2 and then we can start scheduling O_3 on WS_2 at time 20 although workstation 2 has been idle from time 0. This is because O_3 cannot be processed until the processing of O_1 is finished at time 20 (the precedence requirement). O_3 can be completed as early as time 72 as the processing time for O_3 on WS_2 is 52. The above scheduling results in a state vector [20, 72, 0] for state 2 of stage 2. Secondly, if O_1 was completed on WS_2, O_3 can be processed at time 26 on WS_2, as the workstation is not available until time 26, and completed at time 78. This results in a state vector [0, 78, 0] for state 2 of stage 2.

From the two state vectors generated above for state 2 of stage 2, it can be seen that the state vector [20, 72, 0] provides a smaller up-to-stage 2 makespan of 72, compared with 78 given by the state vector [0, 78, 0]. Therefore [20, 72, 0] is chosen as the state vector for state 2 of stage 2. The dark solid line connecting WS_1 at stage 1 to WS_2 at stage 2 denotes the chosen action. The dotted line connecting WS_2 at stage 1 to WS_2 at stage 2 denotes the abandoned action.

In case 2 where O_3 is processed on WS_3, there also exist two possible ways of scheduling O_3, either transferring job 1 from WS_1 to WS_3 and then processing O_3 on WS_3 or transferring job 1 from WS_2 to WS_3 and then processing O_3 on WS_3. Following the same procedure as in case 1, it can be shown that the state vector for state 3 of stage 2 should be chosen as [20, 0, 58] and the solid line connecting WS_1 at stage 1 to WS_3 at stage 2 denotes the chosen action.

The above process results in two good state vectors, [20, 72, 0] and [20, 0, 58]. The first one results from processing O_1 on WS_1 and then processing O_3 on WS_2. Three paths (or legal partial schedules) leads to this state vector, that is ($WS_1 \rightarrow WS_1 \rightarrow WS_2$), ($WS_2 \rightarrow WS_1 \rightarrow WS_2$), and ($WS_3 \rightarrow WS_1 \rightarrow WS_2$). These partial schedules for job 1 can be precisely recorded. For example, the partial schedule ($WS_1 \rightarrow WS_1 \rightarrow WS_2$) can be recorded by S_{31} as given by (3). The second state vector results from processing O_1 on WS_1 and then processing O_3 on WS_3. There are also three paths leading to this second state vector, which are ($WS_1 \rightarrow WS_1 \rightarrow WS_3$), ($WS_2 \rightarrow WS_1 \rightarrow WS_3$), and ($WS_3 \rightarrow WS_1 \rightarrow WS_3$).

The state vector [20, 0, 58] provides the minimum up-to-stage 2 makespan. The new heuristic, however, keeps both state vectors [20, 72, 0] and [20, 0, 58], which represent the conditions that resulted from six good legal partial schedules. Note that there are altogether 12 possible legal partial schedules for scheduling job 1, all of which were assessed in the above procedure. Since in the above scheduling procedure Bellman's optimality principle was adopted in a forward recurrence fashion [8], the calculations required are dramatically reduced in comparison with simply enumerating all the possible partial schedules, as discussed in Section IV-A in more detail. Another feature of the procedure is that it can guarantee to generate a legal partial schedule for a job if there exists a legal schedule for the job in the FMS. Here it is assumed that a job can be scheduled independently of other jobs.

C. Sequence Dependent Job Scheduling Using DDP

Having scheduled job 1 using discrete dynamic programming (DDP), the second job (job 2 or J_2) of the given sequence is scheduled next. Ideally, job 2 should be scheduled following every possible partial schedule resulting from scheduling job 1. However, this would lead to an exponential increase of calculations with the increase of problem sizes. In the new heuristic, only good partial schedules with relatively small up-to-the-stage makespan are considered, as represented by the two state vectors [20, 72, 0] and [20, 0, 58] in the example. In the rest of the section, we first explain how to schedule job 2 based on these two state vectors using DDP.

Given the initial condition represented by the state vector [20, 72, 0], a discrete dynamic program for scheduling job 2 can be constructed, as shown in Fig. 3(a), where stage 2 stands for the initial condition, stage 3 for the processing of the operation O_2 of job 2 and stage 4 for the processing of O_3. Note that there are two workstations capable of processing O_3, WS_2, and WS_3. Job 2 can be scheduled in the same way as for scheduling job 1. If O_3 is processed on WS_2, two state vectors [20, 124, 30] and [40, 124, 0] can be generated for state 2 of stage 4, both of which have the same makespan of 124. The state vector [20, 124, 30] resulted from one path ($WS_3 \rightarrow WS_3 \rightarrow WS_2$) and [40, 124, 0] from two paths ($WS_1 \rightarrow WS_1 \rightarrow WS_2$) and ($WS_3 \rightarrow WS_1 \rightarrow WS_2$). If O_3 is processed on WS_3, a state vector [20, 72, 68] can be generated for state 3 of stage 4, which resulted from the path

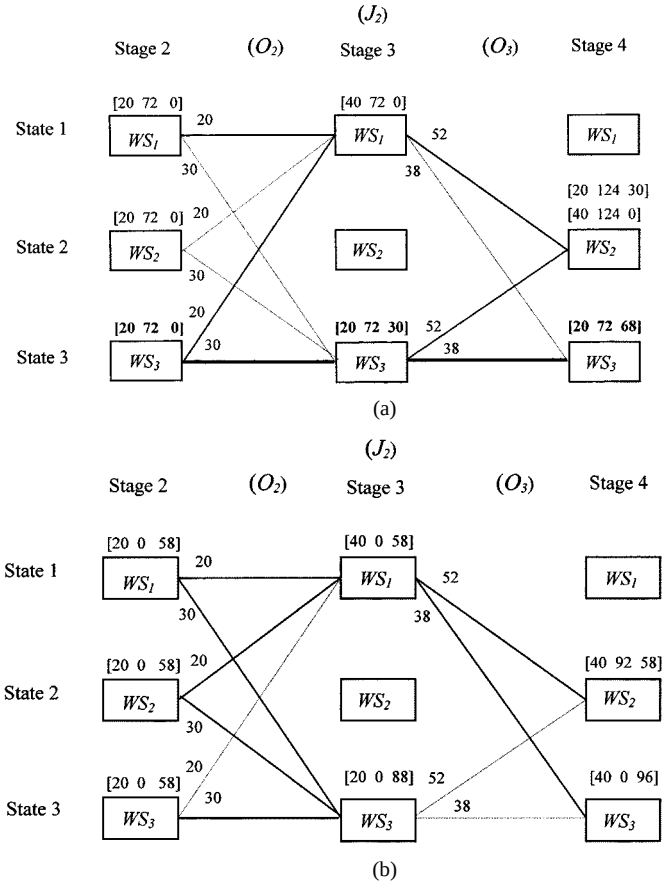

Fig. 3. (a) Given a starting state [20 72 0], schedule the operations of job 2. (b) Given a starting state [20 0 58], schedule the operations of job 2.

($WS_3 \rightarrow WS_3 \rightarrow WS_3$). These paths (partial schedules for job 3) can be precisely recorded. For example, the partial schedule ($WS_3 \rightarrow WS_3 \rightarrow WS_3$) can be recorded by S_{32} as follows:

$$S_{32}\colon \langle \cdots (J_2, O_2)/(WS_3\colon 0, 30), (J_2, O_3)/(WS_3\colon 30, 68) \cdots \rangle. \tag{4}$$

Given the other initial condition represented by the state vector [20, 0, 58], another DDP can be constructed for scheduling job 2, as shown in Fig. 3(b). Similarly, if O_3 is processed on WS_2, another state vector [40, 92, 58] can be generated for state 2 of stage 4, which resulted from two paths ($WS_1 \rightarrow WS_1 \rightarrow WS_2$) and ($WS_2 \rightarrow WS_1 \rightarrow WS_2$). If O_3 is processed on WS_3, a state vector [40, 0, 96] can be generated for state 3 of stage 4, which resulted from two paths ($WS_1 \rightarrow WS_1 \rightarrow WS_3$) and ($WS_2 \rightarrow WS_1 \rightarrow WS_3$).

Now there is a need to decide the state vector for states 2 and 3 of stage 4. The above procedure generated three state vectors for state 2 of stage 4, i.e., [20, 124, 30], [40, 124, 0] and [40, 92, 58]. [40, 92, 58] is chosen as the state vector for state 2 of stage 4 because it provides the minimum up-to-the-stage makespan for the state. Two state vectors were generated for state 3 of stage 4, i.e., [20, 72, 68] and [40, 0, 96]. Similarly, [20, 72, 68] is chosen as the state vector for state 3 of stage 4.

Having scheduled job 1 and job 2, the last job (job 3) of the sequence can be scheduled on the basis of the initial conditions represented by the two state vectors [20, 72, 68] and [40, 92, 58]. Based on the two state vectors, two dynamic programs can be constructed as shown by Fig. 4(a) and (b), where stage 6 is the last stage for the given job sequence. From Fig. 4(a), in the same way as for scheduling job 2 the state vector [59, 124, 68] can be generated for state 2 of stage 6 and [59, 72, 106] as a state vector for state 3 of stage 6. The path

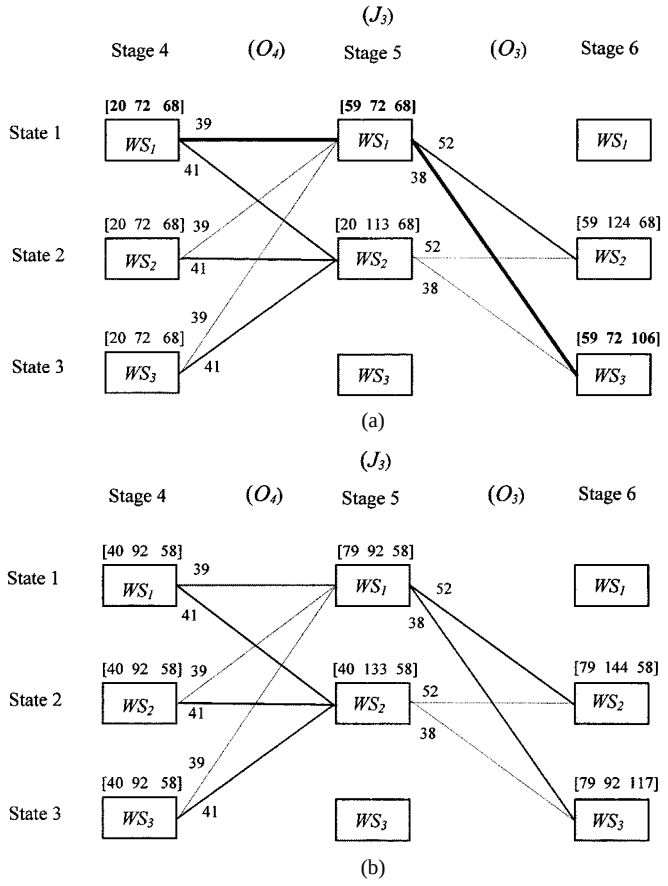

Fig. 4. (a) Given a starting state [20 72 68], schedule the operations of job 3. (b) Given a starting state [40 92 58], schedule the operations of job 3.

$(WS_1 \rightarrow WS_1 \rightarrow WS_3)$ leads to the state vector [59, 72, 106], which can be recorded by S_{33} as follows:

$$S_{33}\colon\ \langle \cdots (J_3,\ O_4)/(WS_1\colon 20,\ 59),\ (J_3,\ O_3)/(WS_3\colon 68,\ 106)\rangle. \tag{5}$$

From Fig. 4(b), [79, 144, 58] can be generated as a state vector for state 2 of stage 6 and [79, 92, 117] as a state vector for state 3 of stage 6.

As a result of scheduling job 3 based on the two chosen initial conditions, we generated four state vectors, [59, 124, 68], [59, 72, 106], [79, 144, 58], and [79, 92, 117], among which the state vector [59, 72, 106] provides the smallest up-to-stage 6 makespan of 106. Since stage 6 is the last stage for the job sequence, [59, 72, 106] is then chosen to be the best state vector generated by the above scheduling procedure. Note that the above four partial schedules for job 3 were generated based on the two selected initial conditions only. Therefore, it cannot be guaranteed that 106 would be the overall minimum makespan for the job sequence.

D. Generation of a Complete Schedule for a Given Job Sequence

Having scheduled each of the three jobs in the sequence separately using DDP, good partial schedules are generated for each job. They can be logically linked together to construct complete schedules. In this section, we discuss how to trace back the above procedure for finding a complete schedule with the minimum identified makespan.

As discussed in the last paragraph of the last section, the state vector [59, 72, 106] provides the makespan of 106 that is the best found so far. It is in Fig. 4(a) that [59, 72, 106] was generated through the path $(WS_1 \rightarrow WS_1 \rightarrow WS_3)$ as recorded by S_{33} in (5). Therefore, S_{33} is chosen as the final partial schedule for job 3.

To identify the final partial schedule for job 2, the initial condition is examined from which [59, 72, 106] was generated. In Fig. 4(a), [59, 72, 106] was generated on the basis of the state vector [20, 72, 68] that was generated in Fig. 3(a). Therefore, the partial schedule leading to [20, 72, 68] can be identified in Fig. 3(a). Obviously, the partial schedule is $(WS_3 \rightarrow WS_3 \rightarrow WS_3)$ as recorded by S_{32} in (4). S_{32} is thus the final partial schedule for job 2.

Similarly, [20, 72, 68] was generated on the basis of [20, 72, 0] which resulted from Fig. 2. Then the final partial schedule for job 1 is identified in Fig. 2. The partial schedule leading to [20, 72, 0] is $(WS_1 \rightarrow WS_1 \rightarrow WS_2)$ or $(WS_2 \rightarrow WS_1 \rightarrow WS_2)$ or $(WS_3 \rightarrow WS_1 \rightarrow WS_2)$. Any of the three paths could be chosen as the final partial schedule for job 1. $(WS_1 \rightarrow WS_1 \rightarrow WS_2)$ is recorded by S_{31} in (3). S_{31} can thus be chosen as the final partial schedule for job 1.

The complete legal schedule resulting in the makespan of 106 for the sequence is finally obtained by combining S_{31}, S_{32}, and S_{33} in sequence as shown by the dark solid lines in Figs. 2, 3(a), and 4(a), that is

$$\begin{aligned} S_3 &= S_{31} \oplus S_{32} \oplus S_{33} \\ &= \langle (J_1,\ O_1)/(WS_1\colon 0,\ 20),\ (J_1,\ O_3)/(WS_2\colon 20,\ 72) \\ &\quad (J_2,\ O_2)/(WS_3\colon 0,\ 30),\ (J_2,\ O_3)/(WS_3\colon 30,\ 68) \\ &\quad (J_3,\ O_4)/(WS_1\colon 20,\ 59),\ (J_3,\ O_3)/(WS_3\colon 68,\ 106)\rangle. \end{aligned} \tag{6}$$

TMS, TFT, and TTC of the schedule S_3 are 106, 246, and 5, respectively. Generated on the basis of the same job sequence $\langle J_1,\ J_2,\ J_3\rangle$, S_3 is obviously better than S_2 in terms of all the three performance measures.

To compare the new heuristic with the simple one, let us demonstrate how S_2 was generated using the simple heuristic and why it is inferior to S_3. In Fig. 2, following the simple heuristic the operation O_1 of job 1 should be scheduled to workstation 1 so that O_1 could be completed as early as possible. In this example, the new heuristic also made the same choice although it may not always do so. At the next stage for scheduling the second operation (O_3) of job 1, the simple heuristic immediately selects workstation 3 so that O_3 could be completed as soon as possible. The new heuristic, however, did not make a choice until the final stage (stage 6). The final choice it made was to select workstation 2 to process O_3 of job 1 instead of workstation 3. It is at this stage that the difference between S_3 and S_2 becomes obvious. This difference results from the fact that the simple heuristic selects an action to achieve immediate benefits when scheduling a current operation while the new heuristic chooses an action based on the overall outcome after scheduling the whole sequence of jobs. The dominance of the new heuristic over the simple one is therefore out of question.

To complete the discussion, let us use the simple heuristic to find S_2 from the DDP diagrams. In Fig. 2, using the heuristic the partial schedule for job 1 is given by the path $(WS_1 \rightarrow WS_1 \rightarrow WS_3)$, as recorded by S_{21}

$$S_{21} = \langle (J_1,\ O_1)/(WS_1\colon 0,\ 20),\ (J_1,\ O_3)/(WS_3\colon 20,\ 58),\ \ldots\rangle.$$

From Fig. 3(b), the partial schedule for job 2 is the path $(WS_1 \rightarrow WS_1 \rightarrow WS_2)$, as recorded by S_{22}

$$S_{22} = \langle \ldots (J_2,\ O_2)/(WS_1\colon 20,\ 40),\ (J_2,\ O_3)/(WS_2\colon 40,\ 92),\ldots\rangle.$$

From Fig. 4(b), the partial schedule for job 3 is the path $(WS_1 \rightarrow WS_1 \rightarrow WS_3)$, as recorded by S_{23}

$$S_{23} = \langle \cdots (J_3,\ O_4)/(WS_1\colon 40,\ 79),\ (J_3,\ O_3)/(WS_3\colon 79,\ 117)\rangle.$$

TABLE III
SCHEDULE PERFORMANCE FOR SIX JOB SEQUENCES

Job Sequence	Performance Measure		
	TMS	TFT	TTC
$<J_2, J_1, J_3>$	97	233	0
$<J_2, J_3, J_1>$	96	247	0
$<J_3, J_2, J_1>$	106	265	5
$<J_3, J_1, J_2>$	115	270	15
$<J_1, J_3, J_2>$	96	247	0
$<J_1, J_2, J_3>$	106	246	5

Thus, S_2 was also identified in the above DDP scheduling procedure and can be obtained by combining S_{21}, S_{22}, and S_{23} as follows:

$$S_2 = S_{21} \oplus S_{22} \oplus S_{23}.$$

Obviously, the new heuristic does not recommend it as the final schedule for the sequence. Note that the SAPT rule plus the simple heuristic may fail to generate a legal schedule if transferring a job between some workstations is not allowed.

E. Job Sequencing and DDP Scheduling

The above DDP scheduling procedure is based on a given sequence of jobs. For different job sequences, the procedure can be repeated to search for good legal schedules. In the example problem, for example, there are only six different job sequences in total. A good schedule can be generated for each of the six sequences using the above DDP procedure. The total makespan (TMS), the total flow time (TFT), and the total tardiness cost (TTC) of the schedules for the six sequences are listed in Table III.

The results shown in Table III are either better than or the same as those reported in [9, Table III] which were generated using the simple heuristic.

IV. GA-BASED DISCRETE DYNAMIC PROGRAMMING APPROACH

A. Discrete Dynamic Programming (DDP) Algorithm

In the above sequence dependent scheduling procedure, all the possible ways of independently scheduling the operations of each job were exhaustively examined for a given initial condition. As discussed in the last section, this procedure can be realized using DDP to reduce the calculations required to schedule the operations of a job. In this section, a DDP algorithm is developed in order to implement the procedure on computers.

In the DDP procedure illustrated in Figs. 2–4, a state vector was defined, which will be generalized in this section. Let WS_i^n denote the state vector of workstation i at stage n as follows:

$$WS_i^n = \begin{bmatrix} w_{i1}^n & w_{i2}^n & \cdots & w_{ij}^n & \cdots & w_{im}^n \end{bmatrix} \tag{7}$$

where w_{ij}^n is the earliest available time of workstation j at stage n after operations 1 to n have been scheduled and if the latest operation (operation n) is scheduled to workstation i.

To account for machine availability and the precedence requirements of operations, a traditional forward recurrence algorithm in dynamic programming [8] may not be used without modification. In the DDP algorithm developed below, however, a new approach is proposed for constructing stage return matrices without changing the forward recurrence nature of the DDP algorithm.

To show data requirements and describe the DDP algorithm, a scheduling problem is represented using Tables IV–VI. In Table IV, pt_{jn} is a nonnegative real number, representing the processing time of operation n on workstation j. In Table V, tt_{ij} is a nonnegative real number, representing the transfer time from workstation i to workstation j. In Table VI, ro_{ij} is a nonnegative integer, indicating that O_j is the ro_{ij}th operation required by job i. $ro_{ij} = 0$ means that O_j is not required by job i.

TABLE IV
PROCESSING TIMES OF OPERATIONS ON WORKSTATIONS

pt_{ij}	O_1	O_2	$\cdots$	O_l
WS_1	pt_{11}	pt_{12}	$\cdots$	pt_{1l}
WS_2	pt_{21}	pt_{22}	$\cdots$	pt_{2l}
$\vdots$	$\vdots$	$\vdots$		$\vdots$
WS_m	pt_{m1}	pt_{m2}	$\cdots$	pt_{ml}

TABLE V
TRANSFER TIMES BETWEEN WORKSTATIONS

tt_{ij}	WS_1	WS_2	$\cdots$	WS_m
WS_1	tt_{11}	tt_{12}	$\cdots$	tt_{1m}
WS_2	tt_{21}	tt_{22}	$\cdots$	tt_{2m}
$\vdots$	$\vdots$	$\vdots$		$\vdots$
WS_m	tt_{m1}	tt_{m2}	$\cdots$	tt_{mm}

TABLE VI
REQUIRED OPERATIONS BY EACH JOB

ro_{ij}	O_1	O_2	$\cdots$	O_l
J_1	ro_{11}	ro_{12}	$\cdots$	ro_{1l}
J_2	ro_{21}	ro_{22}	$\cdots$	ro_{2l}
$\vdots$	$\vdots$	$\vdots$		$\vdots$
J_k	ro_{k1}	ro_{k2}	$\cdots$	ro_{kl}

Define stage return matrix R and state value vector F as follows:

$$R_n = \begin{bmatrix} r_{11}^n & r_{12}^n & \cdots & r_{1m}^n \\ r_{21}^n & r_{22}^n & \cdots & r_{2m}^n \\ \vdots & \vdots & & \vdots \\ r_{m1}^n & r_{m2}^n & \cdots & r_{mm}^n \end{bmatrix} \tag{8}$$

$$F_n = \begin{bmatrix} f_1^n & f_2^n & \cdots & f_i^n & \cdots & f_m^n \end{bmatrix}^T \tag{9}$$

where r_{ij}^n is the stage return from state i of stage n to state j of stage $n+1$; f_i^n is the best value of state i at stage n.

Let f_i^{n-1} be the earliest available time of workstation i after operations 1 to $n-1$ have been scheduled, that is

$$f_i^{n-1} = w_{ii}^{n-1}. \tag{10}$$

Let $\overline{r}_{ij}^{n-1}$ be the sum of the processing time pt_{jn} required to process operation n on workstation j and the transfer time tt_{ij} from workstation i to workstation j, that is

$$\overline{r}_{ij}^{n-1} = pt_{jn} + tt_{ij}. \tag{11}$$

Then a new forward recurrence algorithm can be formulated as follows:

$$f_i^n = \min_j \left\{ \overline{r}_{ji}^{n-1} + \max\left\{ f_j^{n-1},\, w_{ji}^{n-1} \right\} \right\}. \tag{12}$$

Given f_i^{n-1} as in (10), f_i^n is then taken as the earliest available time of workstation i after operation n has been processed. In (12), the machine availability has been accounted for by the second term of the right-hand side.

Algorithm (12) can be converted to the traditional form of a forward recurrence algorithm in dynamic programming [8]. Note that the following equation always holds:

$$\max\left\{ f_j^{n-1},\, w_{ji}^{n-1} \right\} = f_j^{n-1} + \max\left\{ 0,\, w_{ji}^{n-1} - f_j^{n-1} \right\}. \tag{13}$$

Define a stage return as follows:

$$r_{ji}^{n-1} = \overline{r}_{ji}^{n-1} + \max\left\{ 0,\, w_{ji}^{n-1} - w_{jj}^{n-1} \right\}. \tag{14}$$

Referring to (10) and (13), we then have the following forward recurrence algorithm:

$$f_i^n = \min_j \left\{ r_{ji}^{n-1} + f_j^{n-1} \right\}. \tag{15}$$

The best policy generated using (15) can be recorded by BP

$$BP = [\, bp_1 \quad bp_2 \quad \cdots \quad bp_m \,]^T \tag{16}$$

where

$$bp_i = j_{\min} \tag{17}$$

and $j_{\min}$ is given so that

$$f_i^n = r_{j_{\min} i}^{n-1} + f_{j_{\min}}^{n-1} = \min_j \left\{ r_{ji}^{n-1} + f_j^{n-1} \right\}. \tag{18}$$

The state vector as defined by (7) can then be updated as follows:

$$w_{i\tau}^n = \begin{cases} f_i^n, & \text{if } \tau = i \\ w_{j_{\min}\tau}^{n-1}, & \text{if } \tau \neq i. \end{cases} \tag{19}$$

Equations (14)–(19) consist of a DDP algorithm for scheduling the operations of a job in a FMS environment. Given Tables IV–VI and initial state vectors WS_i^0 for workstation i, for example $WS_i^0 = \mathbf{0}$, the equations can be used recursively for scheduling all the operations of a job to achieve minimum up-to-the-minute makespan for the job.

Suppose a job has n operations which can be performed on any of m workstations. Then there are m^{n+1} possible partial schedules for this job. A full enumeration would require $(n+1)m^{n+1}-1$ calculations but the above dynamic programming algorithm only needs $nm(2m-1)$ calculations. $(n+1)m^{n+1} - 1 = nm(2m-1)$ if $n = m = 1$. If $n = m = 10$, however, then $(n+1)m^{n+1} - 1 = 1.1 \times 10^{12} - 1$ and $nm(2m-1) = 1900$. This dramatic reduction of calculation effort is significant and desirable because the DDP algorithm needs to be invoked for each job in each job sequence. Note that with a full enumeration the calculations increase exponentially with the operation number n and the workstation number m. With the DDP algorithm, however, the calculations only increase linearly with n and quadratically with m.

B. Legal Schedule Generation Using the DDP Algorithm

A complete legal schedule consists of assigning every operation of each job in a job sequence to a workstation for processing within a certain time period with the three conditions as discussed in Section II satisfied. In this section, we investigate how to generate and record partial and complete schedules using the above DDP algorithm.

For each job, three matrices are defined to record where an operation of the job is processed, when the processing starts and when it is completed for the selected schedules, referred to as processing machine (PM) matrix, starting time (ST) matrix, and finishing time (FT) matrix, respectively

$$PM^n = \begin{array}{c} \\ WS_1 \\ WS_2 \\ \vdots \\ WS_m \\ \\ \end{array} \begin{array}{c} \begin{array}{cccc} O_1 & O_2 & \cdots & O_l \end{array} \\ \begin{bmatrix} p_{11}^n & p_{12}^n & \cdots & p_{1l}^n \\ p_{21}^n & p_{22}^n & \cdots & p_{2l}^n \\ \vdots & \vdots & \ddots & \vdots \\ p_{m1}^n & p_{m2}^n & \cdots & p_{ml}^n \end{bmatrix} \\ \begin{array}{cccc} O_1 & O_2 & \cdots & O_l \end{array} \end{array} \tag{20}$$

$$ST^n = \begin{array}{c} \\ WS_1 \\ WS_2 \\ \vdots \\ WS_m \end{array} \begin{array}{c} \begin{array}{cccc} & & & \end{array} \\ \begin{bmatrix} st_{11}^n & st_{12}^n & \cdots & st_{1l}^n \\ st_{21}^n & st_{22}^n & \cdots & st_{2l}^n \\ \vdots & \vdots & \ddots & \vdots \\ st_{m1}^n & st_{m2}^n & \cdots & st_{ml}^n \end{bmatrix} \\ \begin{array}{cccc} & & & \end{array} \end{array} \tag{21}$$

$$FT^n = \begin{array}{c} \\ WS_1 \\ WS_2 \\ \vdots \\ WS_m \end{array} \begin{array}{c} \begin{array}{cccc} O_1 & O_2 & \cdots & O_l \end{array} \\ \begin{bmatrix} ft_{11}^n & ft_{12}^n & \cdots & ft_{1l}^n \\ ft_{21}^n & ft_{22}^n & \cdots & ft_{2l}^n \\ \vdots & \vdots & \ddots & \vdots \\ ft_{m1}^n & ft_{m2}^n & \cdots & ft_{ml}^n \end{bmatrix} \\ \begin{array}{cccc} & & & \end{array} \end{array}. \tag{22}$$

In equations (20)–(22), p_{ij}^n is a positive integer and st_{ij}^n and ft_{ij}^n are nonnegative real numbers. They mean that operation j ($j \leq n$) of a job is processed on workstation p_{ij}^n from time st_{ij}^n to time ft_{ij}^n in a schedule where operation n of the job is performed on workstation i.

A step-by-step algorithm can be formulated for scheduling all operations of a job in order to achieve a local minimum makespan.

Step 1) Initialize the state vector WS_i^0 for workstation i and let $n = 1$. For the first job in a sequence, for example, the initial state may be defined as the zero state, or

$$WS_i^0 = [\, 0 \quad 0 \quad \cdots \quad 0 \,] \qquad \text{for } i = 1, 2, \ldots, m. \tag{23}$$

Step 2) Calculate the best value f_i^{n-1} of each workstation at stage $n-1$ using (10) and the stage return matrix R_{n-1} using (8) and (14).

Step 3) Calculate the best value f_i^n for each workstation at stage n using (15) and record the best policy using (16)–(18).

Step 4) Update the state vector for each workstation using (19).

Step 5) Update the processing machine matrix as follows, where $bp_i = j_{\min}$:

$$p_{i\tau}^n = \begin{cases} i, & \text{if } \tau = n \\ p_{j_{\min}\tau}^{n-1}, & \text{if } \tau < n. \end{cases} \tag{24}$$

Step 6) Update the starting time matrix as follows, where $bp_i = j_{\min}$:

$$st_{it}^n = \begin{cases} \max\left\{ w_{j_{\min} i}^{n-1},\, w_{j_{\min} j_{\min}}^{n-1} \right\}, & \text{if } t = n \\ st_{j_{\min} t}^{n-1}, & \text{if } t < n. \end{cases} \tag{25}$$

Step 7) Update the finishing time matrix as follows, where $bp_i = j_{\min}$:

$$ft_{i\tau}^n = \begin{cases} w_{ii}^n, & \text{if } \tau = n \\ ft_{j_{\min}\tau}^{n-1}, & \text{if } \tau < n. \end{cases} \tag{26}$$

Step 8) If $n = l$, stop; otherwise, let $n = n + 1$, go to step 2.

The above algorithm can achieve a local minimum makespan for scheduling a job whichever workstation performs the last operation of the job for a given initial condition. For a sequence of jobs, the first job is scheduled using the above algorithm from a given initial state, say the zero state, resulting in a state vector for each workstation which can perform the last operation of the first job. The second job can then be scheduled on the basis of the state vectors generated for scheduling the first job. This process is repeated for the other jobs in the sequence until the last job is scheduled. The schedule that achieves the smallest makespan for the last job is then chosen to be the best schedule for the sequence.

C. GA-Based Sequence Generation

To implement the above DDP algorithm, a job sequence has to be generated. Different techniques or heuristics may be used for generating legal sequence. The shortest average processing time (SAPT) rule is one of such heuristics. This heuristic as discussed in Section II, however, did not generate a good job sequence for the example problem as discussed in Section III-A and shown in Table III.

The complete enumeration of all possible job sequences is normally impractical for scheduling purpose unless the number of jobs is very small. This is because the number of sequences for k jobs is $k!$. Given $k = 10$, for example, we have $10! = 3\,628\,800$, which means that a schedule generation algorithm needs to be invoked over three and a half million times if all possible job sequences are examined. For a larger number of jobs, say $k = 40$, much greater computation effort is required ($40! = 8.16 \times 10^{47}$ possible sequences). Thus, other techniques for job sequence generation are required such as tree search methods and GA-based approaches.

GA-based approaches are favored in this paper due to the following main reasons. First, with GA approaches we could always invoke the schedule generation algorithm with only complete sequences. A large number of unnecessary invocations using incomplete sequences as with tree search methods can thus be avoided. The second reason is that the evaluation of a sequence with GA approaches is based on the actual performance of the sequence instead of any estimate as with tree search methods. Consequently, the quality of subsequent sequences generated using GA approaches could be expected to be better. In a GA-approach, sequence generation and schedule generation could be treated as two separate procedures and a GA approach can generate and evaluate several sequences simultaneously. It is therefore possible to implement the two procedures in a parallel environment. Finally, a GA-based approach can readily be extended to deal with multiobjective scheduling problems.

The GA approach as used in this paper employs three main genetic operators:

1) reproduction;
2) crossover;
3) mutation [13], [15].

In reproduction, an initial parent population of feasible sequences is generated randomly. The size of the population, or the number of sequences in the population, is denoted by N and may be fixed. Each sequence has a performance measure or a fitness function f associated with it. The total makespan (or TFT, TTC, etc.) for a sequence may be used as such a fitness function. However, other performance measures may also be used as a fitness function. We assume that the fitness f_i, $i = 1, \ldots, N$, have been assigned for the N members of the parent population. To generate the next generation of sequences, M members ($M \leq N$) are sampled from the parent population. In this selection, f_i/F may be used as the probability of the ith member of the population being chosen, where F is the total fitness of the parent population, i.e., $F = \sum_{i=1}^{N} f_i$. Thus a member with high fitness is more likely to be selected. Following reproduction, a mating pool of M sequences is obtained to which further genetic operators are applied.

The crossover transform allows the characteristics of the sequences in the mating pool to be altered, so that the best characteristics could be represented in the next generation. In this transform, however, an offspring sequence must be legal in the sense that each job in the sequence must appear and must only appear once. The subtour chunking crossover [9] may be used for this purpose. Given two parent sequences, S_1 and S_2, an offspring O_{12} is constructed from the two parents by alternately taking "chunks" from the two parent sequences and placing them in the offspring sequence at approximately the same positions that they occupied in the parent sequences. Conflicts are resolved by trimming the chunks to avoid duplication of jobs in the offspring and by sliding them to the nearest available free location in the offspring. An example of this crossover transform was given in [9].

Mutation prevents the genetic search process from a premature loss of genetic material due to reproduction and crossover and it compensates for sampling error. For a given sequence, the process of mutation consists simply of randomly picking two distinct locations within the sequence and exchanging the elements (jobs) at these locations to generate a single offspring sequence. The process may be applied to a few members from the population pool according to the probability of mutation. Obviously, the mutation process only involves swap transformation and can thus preserve the legality of a sequence.

The GA search process is governed by the size of population, the number of generations, the probabilities of crossover and mutation, and probably the generation gap or the proportion to be replaced with new sequences in the next generation. These parameters could be adjusted to improve the quality of the GA search. This GA approach for sequence generation is task independent and can be combined with the DDP algorithm to serve as a general scheduling approach in FMS environments.

D. Methodology for Implementing the GA-DDP Approach

Real world FMS scheduling problems are complex in that many factors need to be catered for and it may not be straightforward to construct scheduling models. As a structured procedure, the GA-DDP approach works on the basis of a set of data and a specially designed model in which to describe a scheduling problem. Tables IV–VI show the data requirements for the approach. A DDP model can be constructed using (8)–(19). To apply the approach, the first step is, therefore, to analyze a scheduling problem, collect data and build a model using these tables and equations. In this modeling process, assumptions and approximation may need to be made.

There are two main iterative steps in the approach: a feasible job sequence generation step using GA and a legal schedule generation step using DDP. If there are only a small number of jobs, e.g., less than ten, the GA step may be replaced by a complete enumeration of all feasible job sequences. Otherwise, the GA algorithm described in Section IV-C can be employed to sample feasible job sequences. Promising job sequences achieving good performance (e.g., makespan) will survive and evolve in the GA step.

For each generated job sequence, the DDP algorithm developed in Sections IV-A and B is then invoked to obtain locally optimized legal schedules. In formulating a DDP model, makespan is used as a performance measure. Note that the same measure can be used in formulating the GA as shown in Section V. In this case, the overall objective is to minimize total makespan only. However, a GA fitness function could also be defined as another performance measure or a combination of measures using a proper criteria aggregation strategy in multiple criteria decision analysis [17], [18]. For instance, the TTC of a sequence of jobs could be defined as a GA fitness function. If such a GA-DDP formulation is applied, then job sequences with low TTC will survive and evolve in the GA step while partial schedules of a job with low makespan will be generated in the DDP step. Consequently, the final generated schedules will achieve both relatively low tardiness cost and makespan.

Note that the GA can generate a family of job sequences simultaneously, each of which may be used to invoke the DDP algorithm separately. This means that the generation of partial schedules using the DDP algorithm may be implemented in a parallel computing environment. Finally, one needs to pay attention to the selection of GA parameters. Experience from preliminary numerical studies shows that good quality schedules can always be generated by using the conventional settings of these parameters [13], [15], such as high crossover rate (above 0.5)

TABLE VII
PROCESSING TIMES OF OPERATIONS ON WORKSTATIONS

Work-station	Operations				
	O_1	O_2	O_3	O_4	O_5
WS_1	8	∞	6	∞	12
WS_2	10	12	∞	9	4
WS_3	∞	∞	8	10	7
WS_4	∞	9	8	7	∞
WS_5	7	7	10	∞	8
Average	8.33	9.33	8	8.67	7.75

TABLE VIII
TRANSFER TIME BETWEEN WORKSTATIONS

From Workstation	To Workstation				
	WS_1	WS_2	WS_3	WS_4	WS_5
WS_1	0	1	∞	2	∞
WS_2	∞	0	1	∞	4
WS_3	4	∞	0	3	2
WS_4	∞	3	2	0	∞
WS_5	2	1	∞	1	0

TABLE IX
OPERATIONS REQUIRED BY JOBS

Job	Required Operations					ATPT
	O_1	O_2	O_3	O_4	O_5	
J_1	1	2	3	0	0	25.66
J_2	1	0	2	0	0	16.33
J_3	1	0	0	2	3	24.75
J_4	3	1	0	2	4	34.08
J_5	0	1	0	0	2	17.08
J_6	0	1	3	2	4	33.75
J_7	1	0	0	0	2	16.08
J_8	1	3	4	2	0	34.38
J_9	1	0	3	2	0	25
J_{10}	0	1	2	0	3	25.08

and low mutation rate (around 0.1). Population size times the number of generations defines the sample size of the GA search. The selection of these two parameters depends on a compromise between the number of jobs in question and the time that can be spent on the analysis. A large sample size provides a better chance of locating the best job sequences but requires more time to complete the search.

V. NUMERICAL STUDY

A. Example 2

This example problem is to schedule ten jobs with each job requiring two to four operations in a predetermined order. It is assumed that there are five workstations, each of which can perform more than one but not all operations. Table VII shows the processing time of each operation on each workstation, where ∞ means that a workstation cannot perform an operation. For instance, it takes eight time units (TUs) to process operation 1 on workstation 1 which cannot process operations 2 or 4. The average processing time of an operation is defined as the mean time required to perform the operation in the five workstation FMS environment. The transfer time between two workstations is also assumed as shown in Table VIII, where ∞ means that a job cannot be transferred from a workstation to another. For instance, transferring a job from workstation 1 to workstation 2 requires 1 TU but it is not possible to transfer a job from workstation 1 to workstation 3 or 5. The required operations of each job with precedence requirements are listed in Table IX , where the average total processing time (ATPT) of a job is defined as the sum of the average processing times of the operations required by the job. The first job J_1, for example, requires operations 1, 2, and 3 in sequence.

B. Problem Complexity

Following the way as suggested by Holsapple *et al.* [9], the complexity of the above scheduling problem is investigated using the data given in Tables VII–IX. For all the ten jobs, there are altogether 30 operations that must be completed. If either the precedence requirements or the machine flexibility is not considered at the time being, then there are $30! = 2.6525 \times 10^{32}$ legal and illegal schedules.

Suppose the precedence requirements are considered. Job 1, for example, involves three operations, O_1, O_2, and O_3. There are $3!$ possible ways to arrange (or sequence) the three operations, of which only one is legal due to the precedence requirement, that is $\langle O_1, O_2, O_3 \rangle$. Thus, the total legal schedules are reduced to $30!/3!$ if the precedence requirement for the three operations of job 1 is taken into account. If the precedence requirements for all the ten jobs are taken into account, the total legal schedules will be reduced to

$$30!/(3! \times 2! \times 3! \times 4! \times 2! \times 4! \times 2! \times 4! \times 3! \times 3!).$$

To account for machine flexibility, take operation 1 (O_1), for example. O_1 is required by jobs 1, 2, 3, 4, 7, 8, and 9. In other words, O_1 appears seven times in any legal schedule. Note from Table VII that O_1 can be performed by any of the three workstations WS_1, WS_2, and WS_5. Therefore, there are 3^7 alternative ways to arrange the processing of operation 1 in each of the legal schedules generated without taking into account machine flexibility. Applying the same argument to all the five operations, it can be concluded that the total number of legal schedules of the problem is actually

$$\begin{aligned} &30!/(3! \times 2! \times 3! \times 4! \times 2! \times 4! \times 2! \times 4! \times 3! \times 3!) \\ &\quad \times (3^7 \times 3^6 \times 4^6 \times 3^5 \times 4^6) \\ &\approx 1.2029 \times 10^{40}. \end{aligned}$$

The task is to find a feasible legal schedule with good quality in terms of certain performance measure such as minimal total makespan. Enumerating and evaluating such a large number of legal schedules is obviously impractical even on the most powerful computers. In the next section, we demonstrate how to generate good quality schedules using the GA-DDP approach.

C. Results and Analysis

First the SAPT rule is applied to sequence the ten jobs and then the simple heuristic discussed in Section II-B is used to generate a legal schedule. This can be achieved manually without relying on a software package. The schedule generated will be compared with other schedules obtained using the GA-DDP approach.

Based on the SAPT rule, a job with the shortest average processing time should be scheduled first. From Table IX, the following job sequence is generated

$$\langle J_7, J_2, J_5, J_3, J_9, J_{10}, J_1, J_6, J_4 J_8 \rangle.$$

Using the simple heuristic, a legal schedule was generated. The Gannt chart of the schedule is shown in Fig. 5, where the (TMS) and the (TFT) of the schedule are 95 and 453, respectively. For the same job sequence the DDP algorithm was used to generate a schedule with the TMS of 88.

The new GA-DDP approach has been coded using C and C++. In the rest of the paper, results generated using the software will be demonstrated and analyzed. The following parameters were used for the GA algorithm: population size = 100; number of generation = 20; probability of crossover = 0.5; probability of mutation = 0.1; Generation gap = 0.9. Fig. 6 shows one of the schedules generated using the software. The job sequence for the schedule is given by

$$\langle J_6, J_9, J_2, J_4, J_8, J_7, J_{10}, J_5, J_1 J_3 \rangle.$$

The TMS and the TFT of this schedule are 60 and 415, respectively. This schedule is obviously much better than that obtained using the

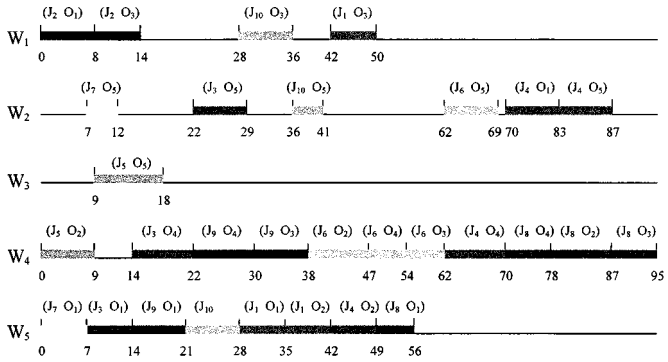

Fig. 5. Gantt chart for the sequence $\langle J_7\ J_2\ J_5\ J_3\ J_9\ J_{10}\ J_1\ J_6\ J_4\ J_8\rangle$.

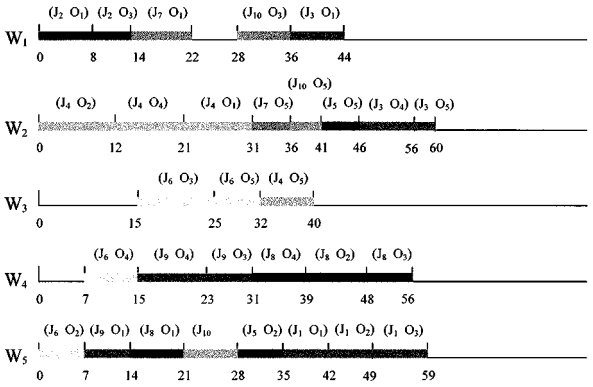

Fig. 6. Gantt chart for the sequence $\langle J_6\ J_9\ J_2\ J_4\ J_8\ J_7\ J_{10}\ J_5\ J_1\ J_3\rangle$.

SAPT rule plus the simple heuristic with regard to both TMS and TFT. In fact, the TMS of the schedule shown in Fig. 6, compared with that in Fig. 5, is reduced by over one third.

Although the best schedules of the problem are not known in terms of either TMS or TFT, the quality of the schedule as illustrated in Fig. 6 is clearly satisfactory. Using the GA parameters as mentioned above, one run of the software took only 63 s on a Pentium 133 PC with 16 MB RAM. Different runs of the software with the same or different sets of GA parameters were also conducted and similar good quality schedules were always generated within relatively short time periods (roughly proportional to the product of population size and generation number).

D. Results Obtained by Changing the Number of Jobs

Example 2 is still a small-sized problem with only 30 operations although it is not simple at all. In this section, the number of jobs is increased in order to investigate the performance of the new approach when it is applied to larger scheduling problems. In Section III-A, it was concluded that the calculations required by the new heuristic only increase linearly with the increase of jobs. The following numerical study will demonstrate this feature.

In Tables VII–IX, a set of ten jobs was defined. An identical set of the ten jobs is appended to Table IX with job 11 being the same as job 1, job 12 as job 2 and so on. For this twenty-job problem, it took 1.95 min to solve the problem using the new approach and many good schedules were generated with a makespan of 124. Then two more identical sets of the ten jobs are appended to Table IX to increase the total number of jobs to 40. Many schedules with a makespan of 248 were found in 4.08 min. In the same way, the total number of jobs is increased up to 80. The results are summarized in Table IX.

In the above study, we only increased the number of identical sets of jobs. To demonstrate the potential of the approach for dealing with large problems with different jobs a scheduling problem is created randomly that has forty different jobs with 144 operations in total, as defined in Table XIII in the Appendix where job 1 to job 10 are the same as in Table IX. The FMS is still defined by Tables VII and VIII. In this problem the total number of legal schedules can be calculated as follows:

$$\begin{aligned} &144!/\left[(2!)^3\times(3!)^{13}\times(4!)^{21}\times(5!)^3\right]\\ &\quad\times\left(3^{32}\times3^{31}\times4^{28}\times3^{23}\times4^{30}\right)\\ &\approx 4.786\times10^{260}. \end{aligned}$$

For this size problem with a huge number of legal schedules, any full-scale enumeration is impossible. However, it took only 4.48 min to solve the problem using our GA-DDP software on the small PC mentioned above.

To study the impact of increasing the number of different jobs on calculation time, the following experiment is conducted. First of all, the first ten jobs in Table XIII and then the first 20 jobs were scheduled, followed by the first 30 jobs. Finally an identical set of the 40 jobs as defined in Table XIII is appended to Table XIII to create an 80 job problem. The calculation results generated from this experiment are shown in Table XI.

It can be seen from Table XI that both the total makespan and the time used increase slightly faster than in Table X with the increase of jobs. This is because the last 30 jobs in Table XIII require more operations than the corresponding jobs in Table X.

E. Results Obtained by Changing the Number of Machines

In the above studies, it was assumed that there are only five workstations in the FMS. To investigate the impact of changing the number of machines on the performance of the new approach, the following numerical experiment was conducted. First, an identical set of the five machines defined in Tables VII and VIII is added to the FMS. In this new system, the 40 jobs defined in Table XIII were scheduled. Good legal schedules with a makespan of 173 were found in 28.02 min. In a similar way, the number of workstations is increased to 15 and 20. The results are summarized in Table XII.

The final scheduling problem has 40 jobs and 20 workstations. The number of legal schedules for this problem can be estimated as follows:

$$\begin{aligned} &144!/\left[(2!)^3\times(3!)^{13}\times(4!)^{21}\times(5!)^3\right]\\ &\quad\times\left(3^{32}\times3^{31}\times4^{28}\times3^{23}\times4^{30}\right)\times4^{144}\\ &\approx 2.38\times10^{347}. \end{aligned}$$

TABLE X
NUMERICAL RESULTS FOR IDENTICAL SETS OF JOBS

Number of Jobs	10	20	40	80
Total Makespan	60	124	248	505
Time Used (minute)	1.05	1.95	4.08	8.37

TABLE XI
NUMERICAL RESULTS FOR DIFFERENT JOBS

Number of Jobs	10	20	30	40	80
Total Makespan	60	132	204	305	610
Time Used (minute)	1.05	1.98	3.17	4.48	9.73

TABLE XII
NUMERICAL RESULTS FOR A DIFFERENT NUMBER OF WORKSTATIONS

Number of Workstations	5	10	15	20
Total Makespan	305	173	118	95
Time Used (minute)	4.48	28.02	72.27	161.8

TABLE XIII
OPERATIONS REQUIRED BY JOBS

Job	Required Operations				
	O_1	O_2	O_3	O_4	O_5
J_1	1	2	3	0	0
J_2	1	0	2	0	0
J_3	1	0	0	2	3
J_4	3	1	0	2	4
J_5	0	1	0	0	2
J_6	0	1	3	2	4
J_7	1	0	0	0	2
J_8	1	3	4	2	0
J_9	1	0	3	2	0
J_{10}	0	1	2	0	3
J_{11}	2	1	3	0	4
J_{12}	1	2	3	0	0
J_{13}	1	0	2	3	4
J_{14}	3	2	0	1	4
J_{15}	3	1	0	0	2
J_{16}	4	3	2	1	0
J_{17}	2	0	3	0	1
J_{18}	1	4	3	0	2
J_{19}	0	1	2	3	0
J_{20}	1	0	2	0	3
J_{21}	0	2	3	4	1
J_{22}	1	3	2	0	0
J_{23}	2	4	0	1	3
J_{24}	1	3	0	2	4
J_{25}	3	1	0	0	2
J_{26}	0	2	3	1	4
J_{27}	3	0	1	0	2
J_{28}	4	3	1	2	0
J_{29}	2	0	3	0	1
J_{30}	0	1	2	4	3
J_{31}	1	2	3	4	5
J_{32}	2	3	1	0	4
J_{33}	1	2	0	4	3
J_{34}	2	1	0	3	4
J_{35}	4	1	5	3	2
J_{36}	3	1	0	2	4
J_{37}	1	2	4	0	3
J_{38}	2	3	4	1	5
J_{39}	4	1	3	2	0
J_{40}	0	1	2	4	3

Obviously it is impossible to enumerate all the legal schedules. In Table XII, one can see that the time needed to solve the problem increases nearly quadratically with the increase of workstations. This mirrors the analysis conducted in Section IV-A.

VI. CONCLUDING REMARKS

The GA-based discrete dynamic programming (GA-DDP) approach proposed in this paper provides an alternative way of generating good quality schedules in an FMS environment. The quality results from the evolutionary generation of job sequences in the GA step and the local optimization of partial schedules in the DDP step. The distinctive features of this GA-DDP approach include its flexibility of accommodating multiple performance criteria and its potential for implementation in a parallel computing environment. The results from this research support the strategy of combining traditional algorithmic procedures with heuristics or adaptive search techniques to develop hybrid FMS scheduling approaches. Preliminary numerical studies reported in this paper have demonstrated the implementation and potential of the GA-DDP approach. However, no adaptive or heuristic search approach could guarantee to generate global optimal schedules. The GA-DDP approach is no exception. Further numerical and comparison studies are needed to explore the approach in order to provide useful guidelines as to the circumstances in which the approach can be properly used.

APPENDIX
SPECIFICATION FOR A 40 JOB SCHEDULING PROBLEM

See Table XIII.

ACKNOWLEDGMENT

The author would like to thank Dr. D. Todd of the University of Newcastle upon Tyne for providing a general-purpose GA-based sequence generation program in C++. He would also like to thank the anonymous referees whose comments contributed to the improvement of the paper.

REFERENCES

[1] D. D. Bedworth and J. E. Bailey, *Integrated Production Control System*. New York: Wiley, 1987.
[2] V. Belton and M. D. Elder, "Exploring a multicriteria approach to production scheduling," *J. Oper. Res. Soc.*, vol. 47, no. 1, pp. 162–174, 1996.
[3] J. Blazewicz *et al.*, "Vehicle scheduling in 2-cycle flexible manufacturing systems," *Math. Comput. Modeling*, vol. 20, no. 2, pp. 19–31, 1994.
[4] S. De and A. Lee, "Flexible manufacturing system (FMS) scheduling using filtered beam search," *J. Intell. Manufact.*, vol. 1, pp. 165–183, 1990.
[5] M. P. Fanti *et al.*, "Genetic multi-criteria approach to flexible line scheduling," *Int. J. Approx. Reasoning*, vol. 19, no. 1–2, pp. 5–21, 1998.
[6] S. French, *Sequencing and Scheduling*. New York: Wiley, 1982.
[7] Y. P. Gupta, G. W. Evans, and M. C. Gupta, "A review of multi-criterion approaches to FMS scheduling problems," *Int. J. Prod. Econ.*, vol. 22, no. 1, pp. 13–31, 1991.
[8] N. A. J. Hastings, *Dynamic Programming with Management Application*. London, U.K.: Butterworth, 1973.
[9] C. W. Holsapple *et al.*, "A genetics-based hybrid scheduler for generating static schedules in flexible manufacturing contexts," *IEEE Trans. Syst., Man, Cybern.*, vol. 23, pp. 953–972, Apr. 1993.
[10] R. Hohzaki, S. Fujii, and H. Sandoh, "Rescheduling for AGV's by time windows concept," *JSME Int. J. Series C-Dyn. Contr. Robot. Design Manufact.*, vol. 38, no. 4, pp. 818–823, 1995.
[11] H. Hwang and J. U. Sun, "Production sequencing problem with re-entrant work flows and sequence dependent setup times," *Int. J. Prod. Res.*, vol. 36, no. 9, pp. 2435–2450, 1998.
[12] N. Jawahar, P. Aravindan, and S. G. Ponnambalam, "A genetic algorithm for scheduling flexible manufacturing systems," *Int. J. Adv. Manufact. Technol.*, vol. 14, no. 8, pp. 588–607, 1998.
[13] C. Y. Lin and P. Hajela, "Genetic algorithms in optimization problems with discrete and integer design variables," *Eng. Optim.*, vol. 19, pp. 309–327, 1992.
[14] S. Shen and Y. Chang, "Scheduling generation in a flexible manufacturing system: A knowledge-based approach," *Dec. Support Syst.*, vol. 4, pp. 157–166, 1988.
[15] M. C. South, G. B. Wetherill, and M. T. Tham, "Hitch-hiker's guide to genetic algorithms," *J. Appl. Statist.*, vol. 20, no. 1, pp. 153–175, 1993.
[16] G. Ulusoy, F. SivrikayaSerifoglu, and U. Bilge, "A genetic algorithm approach to the simultaneous scheduling of machines and automated guided vehicles," *Comput. Oper. Res.*, vol. 24, no. 4, pp. 335–351, 1997.
[17] J. B. Yang and M. G. Singh, "An evidential reasoning approach for multiple attribute decision making with uncertainty," *IEEE Trans. Syst., Man, Cybern.*, vol. 24, pp. 1–18, Jan. 1994.
[18] J. B. Yang, "Gradient projection and local region search for multiobjective optimization," *Eur. J. Oper. Res.*, vol. 112, no. 2, pp. 432–459, 1999.