Feasibility of Hungarian Algorithm based Scheduling

Feasibility of Hungarian Algorithm based Scheduling

Shinsuke Tamura, Yuki Kodera, Shuji Taniguchi and Tatsuro Yanase
Graduate school of Engineering, University of Fukui
3-9-1, Bunkyo, Fukui, Japan
tamura@u-fukui.ac.jp

***Abstract*—An optimal resource allocation algorithm, Hungarian algorithm, is not directly applicable to manufacturing scheduling problems, because solutions of resource allocation problems may violate precedence constraints among processes that constitute individual manufacturing jobs. To apply Hungarian algorithm to scheduling problems, in this paper, several strategies for assigning prices to time slots of individual machines, which are allocated to processes, are proposed. Preliminary experimentation results showed that these strategies can generate near optimal schedules, i.e. when lengths of scheduling horizons were larger than 3 times of the maximum lengths of jobs, generated schedules could complete given jobs while maintaining the deterioration of the efficiency less than 5% from optimal schedules.**

Keywords—**scheduling, hungarian algorithm, near-optimal schedules**

I. Introduction

The responsibility of manufacturing scheduling is increasing rapidly with the shift from low variety high volume to high variety low volume productions. Schedulers are required to quickly generate more efficient schedules for larger amount of manufacturing activities in environments where both customers' requirements and manufacturing facilities change frequently. Although many scheduling systems had been developed based on various approaches, e.g. heuristic based and mathematical optimization based schedulers [1][2][3][4], yet they are not powerful enough. Heuristics based schedulers are not efficient enough because they do not ensure any optimality, and mathematical optimization based ones cannot be applied to complicated manufacturing systems because of various restrictions caused by their simple formulations. As one of approaches to develop schedulers that satisfy the requirements in high variety low volume manufacturing, this paper discusses the feasibility of a scheduling method based on Hungarian algorithm.

A manufacturing scheduling problem is a kind of resource allocation, i.e. when individual time slots of machines are considered as resources, scheduling is a problem just to allocate these time slots to processes that constitute given manufacturing jobs. Therefore it is natural to apply Hungarian algorithm, an optimal resource allocation algorithm that allocates resources to jobs so that the cost functions are minimized, to scheduling problems. However, because of precedence constraints that are essential in manufacturing scheduling, Hungarian algorithm cannot be applied in a straightforward way. Namely, usually in manufacturing scheduling, a set of processes to be completed to accomplish a job should be executed in the predefined order; however these orders cannot be preserved when Hungarian algorithm is applied without any modification.

To make Hungarian algorithm suffice these precedence constraints, this paper proposes strategies for assigning prices to individual time slots of machines for executing individual processes. Intuitively, the strategies assign prices to time slots to execute contiguous processes P_1 and P_2, so that prices for executing P_1 increase more rapidly as the slot beginning time increases than for P_2 that succeeds P_1. By these strategies, the sum of execution cost of P_1 and P_2 becomes smaller when earlier time slots are assigned to P_1 than the case where they are assigned to P_2, i.e. precedence constraints are satisfied. Preliminary experimentation results showed that the proposed strategies can generate near optimal schedules, i.e. when lengths of scheduling horizons exceeded more than 3 times of the maximum length of jobs generated schedules could complete given jobs while maintaining the deterioration of the efficiency less than 5% from optimal schedules.

II. Resource Allocation and Scheduling Problems

Table 1 shows an example of a resource allocation problem. In the table, the 0-th row and column of the matrix represents resources and jobs, respectively, and (i, j) element represents the price for allocating the j-th resource to the i-th job. Here, it is assumed that the numbers of jobs and resources are the same; this assumption is satisfied always by defining dummy jobs or resources if necessary. Then, optimal resource allocation is a problem to allocate resources to jobs so that the total allocation cost becomes minimal, i.e. to select one element in each row and column, so that the sum of prices of selected elements becomes minimal. In the figure a set of selected elements {(1, 5), (2, 1), (3, 4), (4, 2), (5, 3)} represents the optimal allocation, i.e. the minimal allocation cost is 240, the sum of prices of selected elements.

Hungarian algorithm finds the above optimal resource allocations efficiently, i.e. it can calculate the optimal allocation with the computation volume of $O(n^3)$. Here n is the number of resources or jobs.

TABLE I. OPTIMAL RESOURCE ALLOCATION PROBLEM

	R_1	R_2	R_3	R_4	R_5
J_1	60	70	50	90	**40**
J_2	**40**	80	90	70	60
J_3	80	90	90	**60**	70
J_4	70	**50**	70	60	60
J_5	50	50	**50**	80	80

Manufacturing scheduling problems are a kind of resource allocation ones. For example, let M_1 and M_2 be manufacturing machines, and $\{P_{11}, P_{12}, P_{13}\}$ and $\{P_{21}, P_{22}, P_{23}\}$ be sequences of processes that constitute jobs J_1 and J_2, respectively. Then, when resources are defined as time slots of individual machines, scheduling manufacturing jobs J_1 and J_2 is just to allocate these time slots to processes $\{P_{11}, P_{12}, P_{13}, P_{21}, P_{22}, P_{23}\}$ as shown in Table 2. However, Hungarian algorithm cannot be directly applied to scheduling problems, because generally there are precedence constraints among processes that constitute individual jobs. In the case shown in Table 2, to complete job J_k (k = 1, 2), processes P_{k1}, P_{k2}, P_{k3} must be accomplished in this order, but Hungarian algorithm allocates time slots to these processes without any constraints. As the consequence, generated schedules may violate these precedence constraints, e.g. in the table, elements (1, 6), (2, 1), (3, 5), (4, 2), (5, 4) and (6, 3) are selected, i.e. J_1 and J_2 are completed by executing processes in the order $\{P_{12}, P_{13}, P_{11}\}$ and $\{P_{21}, P_{23}, P_{22}\}$, respectively.

TABLE II. MANUFACTURING SCHEDULING PROBLEM

		M_1			M_2		
			Slot-2 (R_2)	Slot-3 (R_3)	Slot-1 (R_4)	Slot-2 (R_5)	Slot-3 (R_6)
J_1	P_{11}	60	70	50	60	50	**40**
	P_{12}	**40**	80	90	80	70	60
	P_{13}	80	90	90	100	**60**	70
J_2	P_{21}	70	**50**	70	80	60	60
	P_{22}	50	60	80	**50**	80	80
	P_{23}	50	70	**50**	80	90	100

III. HUNGARIAN SCHEDULER

Hungarian scheduler proposed in this paper consists of the following steps,

1) N = 0 (initialize the number of price modification times),
2) assign initial prices to individual slots of machines for processing individual processes,
3) execute Hungarian algorithm,
4) if there is no constraint violation, then terminate calculation, else go to step 5,
5) if $N = N_{\max}$, then set N to 0 and go to step 7, else go to step 6,
6) $N = N + 1$, modify prices of selected slots for executing selected processes, and go to step 3,
7) add time slots and go to step 1.

In the above procedure, prices of time slots for executing individual processes are defined and modified according to the strategies described later, so that the precedence constraints among processes are satisfied. When the precedence constraints cannot be satisfied even prices are modified predefined number of times, an extra slot is added, and the same procedure is iterated. Namely, original scheduling horizon is expanded until the constraint violations are removed completely. Here, because the number of processes is finite, it is apparent that the constraints violations can be removed eventually, provided that prices of individual time slots are defined and modified appropriately.

Strategies for assigning and modifying prices of slots for executing individual processes are described in the following subsections. In the following, it is assumed that jobs are accomplished through sequences of processes that do not have either branches or junctions, i.e. each job J consists of a sequence of processes P_1, P_2, ---, P_N, that should be executed in this order.

A. Type-1: Equal Length Processes and Equal Performance Machines

This subsection considers scheduling problems, in which machines may have different functions but they execute functions with the same performance, and all jobs consist of processes that require the same processing time. In this case if the length of slots is defined as same as the processing times of processes, all processes can be executed within single time slots of machines. Therefore, the only deference between scheduling and resource allocation problems is the existence of precedence constraints, and simple price assignment strategy described below works effectively. In the following, n(P) represents that process P is the n(P)-th process of some job J, e.g. for $\{P_{11}, P_{12}, P_{13}\}$ in the previous section, $n(P_{11}) = 1$, $n(P_{12}) = 2$, $n(P_{13}) = 3$.

Strategy-1: For executing process P, define the price of a slot that begins from time t as ∞ when $t < n(P)$ and $c(n(P))(t - n(P))$ when $n(P) \le t$, where $c(n(P))$ is positive, therefore, the price increase linearly as slot beginning time increases. Also coefficient $c(n(P))$ is defined so that the prices increase less rapidly as n(P) increases.

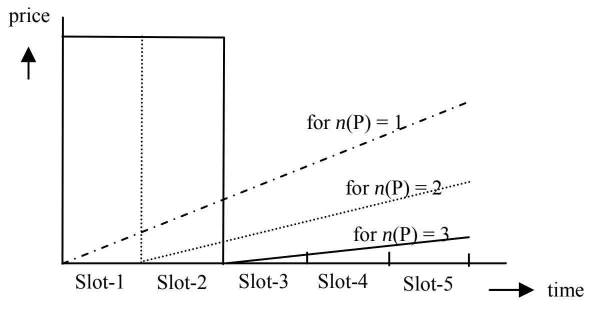


Figure 1. Prices of slots for executing process P

Fig. 1 depicts how coefficient *c*(n(P)) is defined. Let P and Q be processes of job J, and $n(P) = p$ and $n(Q) = q$ ($p < q$). Then according to Strategy-1, it is apparent that the slot assignment cost of a schedule, in which Q is executed before P, decreases when slots that execute P and Q are exchanged. Therefore it is expected that the strategy generates a schedule which executes P earlier than Q that succeeds P, i.e. the strategy is prone to satisfy precedence constraints. In the figure, the price of Slot-1 for executing process P with $n(P) = 2$ and $n(P) = 3$ are defined as ∞, because P cannot be executed until processes that precedes it had been completed.

However it is not ensured that precedence constraints are satisfied always; therefore Strategy-2 that modifies prices so that preceding processes are forced to be processed earlier is necessary.

Strategy-2: When Hungarian algorithm generates a schedule that executes process Q before completing P that precedes Q (i.e. P and Q are processes of same job J and $n(P) < n(Q)$), define prices of slots for executing Q so that the beginning time of Q is forced to be greater than or equal to $t_F(P)$, or define prices of slots for executing P so that the beginning time of P is forced to be less than $t_B(Q)$. Here $t_F(P)$ and $t_B(Q)$ represent the originally scheduled completion time of P and the beginning time of Q, respectively.

Strategy-2 can be implemented by 2 different ways, i.e. by *inhibited slots*, or by *early start* or *late completion penalties*. An inhibited slot for process X is a slot, of which price for executing X is infinite; therefore if slots that begin earlier/later than the scheduled completion time of P/Q are defined as inhibited slots for Q/P, Q/P is rescheduled to be executed after/before P/Q, unless P/Q is moved to a further later/earlier slot in the rescheduling. An early start/late completion penalty is the additional cost to be paid when a process begins/terminates before/after the designated time; and can be achieved by modifying prices of slots. When prices of slots for executing P decreases when $t < t_S$, or they increases more rapidly when $t > t_F$ as shown in Fig.2, P is expected to be executed after t_S or before t_F, because the cost for executing P before t_S or after t_F increases more than executing other processes during that period. Obviously inhibited slots can implement strategy-2 in a stricter way, however early start/late completion penalties leave Hungarian algorithm larger number of possibilities to be evaluated as candidate schedules, i.e. it is expected that the algorithm can find more efficient schedules.

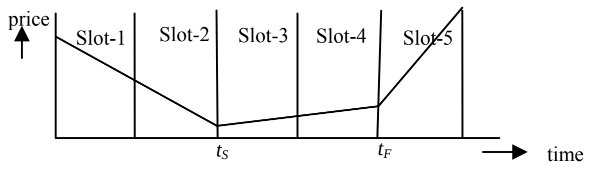


Figure 2. Early start and late completion penalty

B. *Type-2: Different Length Processes and Equal Performance Machines*

When processing times of individual processes that constitute jobs are not the same, the length of slots cannot be defined so that every process can be accomplished within single slot duration; there are processes that require multiple slots for their completions. To schedule a process that cannot be accomplished within a single slot, it is necessary to divide it into multiple sub-processes so that they can be accomplished within single slots, and to introduce *adjacency constraints* to these sub-processes, i.e. these sub-processes must be executed in a sequence of mutually adjacent slots of the same machines as shown in Fig.3.

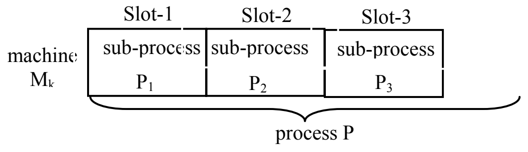


Figure 3. Sub-processes and an adjacecy constraint

Although adjacency constraints are more difficult to satisfy than usual precedence constraints, they are not so difficult to satisfy when alternative machines of all processes have the sane functions and equal performances as follows. Here, *alternative machines* of process P are a set of machines that can execute P. Then let machines M_1 and M_2 be alternative machines of process P and Q, and slots S_1, --, S_K of M_1 be ones that are assigned to sub-processes from P_1 to P_K of process P, except slot S_j, as shown in Fig. 4. Slot S_j of M_1 is assigned to sub-process Q_x of process Q, and sub-process P_j is scheduled to be executed by slot S_j of M_2, i.e. the adjacency constraint on P is not satisfied. However, if M_1 and M_2 have the same functions and the equal performance, slots assigned to P_j and Q_x, i.e. slot S_j of M_1 and M_2, can be exchanged each other without changing the completion time of either of P and Q. Therefore, simple local adjustments can generate a schedule that satisfies the adjacency constraint on P.

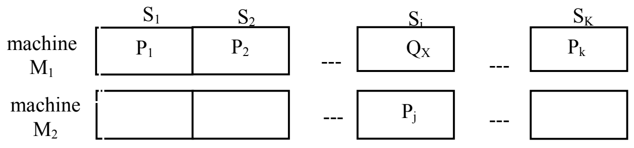
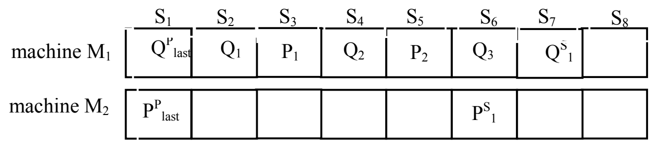


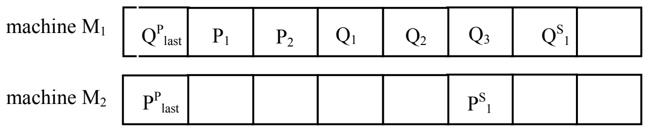
Figure 4. Exchanging assigned slots

Also, even when sub-processes of P (i.e. P_1 and P_2) are separated by those of Q (i.e. Q_1, Q_2 and Q_3) as shown in Fig. 5 (a), if P^S (process succeeds P) is scheduled to start before Q^S (process succeeds Q), sub-processes of P and Q can be exchanged so that all sub-processes of P precede those of Q without changing start times of P^S,Q^S, P^P and Q^P (P^P and Q^P are processes precedes P and Q, respectively). In Fig. 5(b), P_1 and P_2 are moved to the slots that precede all sub-processes of Q, however, this exchange does not delay the completion time of Q.

In the above case, P^P_{last} (the last sub-process of P^P that precedes P) is scheduled to be completed before the start time of Q_1 from the beginning. However, when P^P_{last} is scheduled to be completed after Q_1 starts, the completion time of Q is delayed, and consequently not only the start times of P and Q bust also that of Q_S must be changed, i.e. simple local adjustments cannot generate feasible schedules.



(a) Original slot allocation



(b) Modified slot allocation

Figure 5. Separated sub-processes

C. *Type-3: Different Performance Machines*

When individual alternative machines have different performances, processing times of a process differ depending on machines that execute it, and it becomes difficult to keep optimality of schedules. Namely, when the slot length is defined so that single slots of lower performance machines can complete individual processes (or sub-processes), higher performance machines cannot show their full ability, because they complete these processes before the end of their slots, and remaining durations of the slots are left unused. On the other hand, it is not possible to define the slot length as the duration, by which higher performance machines can complete individual processes, because lower performance machines cannot complete these processes within their single slots.

Introduction of *dummy time slots* and *dummy processes* enable the scheduler to generate efficient schedules also in these situations, provided that performance ratios between alternative machines do not depend on processes. This condition means that, when process P and Q are accomplished by 3 and 6 slots of machine M_1 respectively, Q must be accomplished in 10 slots by machine M_2, if P is accomplished in 5 slots by M_2, for example. Here a dummy time slot is defined for lower performance machines to adjust performance difference between alternative machines, and they are assigned only to dummy processes, i.e. prices of dummy slots are infinite for all processes except dummy processes, and performance differences between machines are adjusted by lower performance machines' dummy process executions. For example, when machine M_1 can accomplish a function, which is accomplished by n time slots of M_2, by its m time slots ($n > m$), (n-m) dummy slots are placed before every m slots of M_2 as shown in Fig. 6. In the figure D_j ($1 \leq j \leq n-m$) represents dummy slots, therefore, M_2 must execute (n-m) dummy processes before completing m processes, and consequently, it can execute only m processes during the period where M_1 executes n processes.

A drawback of the above dummy slot arrangement is that it allocates fixed number of dummy slots regardless of process length (execution time of the process), although the difference between process completion times of the higher and the lower performance machines becomes small when the process length is small. Therefore in Fig. 6, dummy slots may delay start times of Q, when lower performance machine M_2 can complete P_1, P_2, ---, P_{last} (sub-processes of P that precedes Q) within less than m slots. These delays can be removed when dummy slots are allocated after every m slots of M_2 as shown in Fig. 7. In Fig.7, Q_1 begins just after slot S_{j+m-1} that is assigned to P_{last}. However, in this case, generated schedules may become infeasible, i.e. succeeding processes may start before the completions of processes that precede them. A possible solution is to distribute (n-m) dummy slots between m non-dummy slots.

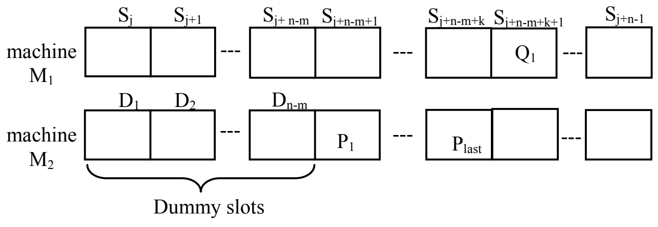


Figure 6. Dummy slots

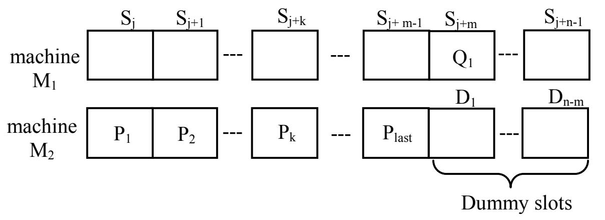


Figure 7. Inconsistent schedule

It must be noticed that different from equal performance machine cases, slot S_j of 2 machines with different performances that are assigned to 2 different processes cannot be exchanged. When they are exchanged, the slot of the lower performance machine may not be able to accomplish the newly assigned process. Therefore, for cases including different performance machines, the scheduler may require more additional slots to generate consistent schedules.

IV. Preliminary Experimentation Results

To evaluate the feasibility of the Hungarian algorithm based scheduler, times required for the scheduling executions and the optimality of scheduling results were measured for simple scheduling problems of types 1, 2 and 3, while comparing job completion times of the scheduler with that of the optimal schedules. Here, optimal schedules were generated by filling all slots of all machines by processes, and generating jobs by randomly selecting all allocated processes so that they can be arrayed in their start time increasing order, and defining these sequences of processes as processes or sub-processes of jobs. In all evaluation cases, inhibited slots were used for implementing strategy-2.

A. *Type-1*

The Hungarian algorithm based scheduler showed the good performance for type-1 cases. Namely, for all cases where 5 machines executed jobs consists of 40-60 processes during 100 time slots scheduling horizon, the proposed method had generated schedules that completed the jobs within durations as same as the optimal ones. However, in cases where there were not enough alternative machines for individual processes, the numbers of iterations of the algorithm executions required to exclude all constraint violations became large, e.g. when more than 1 alternative machines were prepared for each process it required less than 20 iterations, on the other hand, when only single alternative machines are available for some processes, more than 200 iterations were required.

B. *Type-2*

Performance of the proposed method was evaluated in an environment consists of 4 identical machines while randomly generating jobs. Where individual jobs consisted of 9-15 processes each of which required 2 time slots of the machines in average, and scheduling horizons were varied from 30 to 90 time slots.

Table III shows the results, where "extra slots" means the number of slots that were added to complete jobs without violating the precedence and the adjacency constraints, i.e. they represent the efficiency deteriorations from optimal schedules (the smaller is better). "Constraint violations" represents the maximum numbers of constraint violations generated during the iterations of Hungarian algorithm executions, and "scheduling time" represents the computation time for generating schedules (of course it is proportional to the number of iterations). According to these tables, it is expected that when the scheduling horizons are long enough (e.g. more than 3 times of the maximum length of jobs) the proposed scheduler generate efficient schedules that complete jobs almost as same as the optimal ones do. Namely, when the length of scheduling horizon was 90 slots, Hungarian algorithm based scheduler had generated feasible schedules without reducing efficiency more than 5% of optimal schedules as shown in Fig. 8.

TABLE III. SCHEDULING RESULTS FOR TYPE-2 CASES

(a) Scheduling horizon = 30 slots

	Case 1	Case 2	Case 31
Extra slots	11	13	11
Constraint violations	56	75	13
Scheduling time	20	26	20

(b) Scheduling horizon = 60 slots

	Case 4	Case 5	Case 6
Extra slots	5	5	8
Constraint violations	20	18	12
Scheduling time	255	207	539

(c) Scheduling horizon = 90 slots

	Case 7	Case 8	Case 9
Extra slots	3	4	4
Constraint violations	8	15	26
Scheduling time	880	962	1304

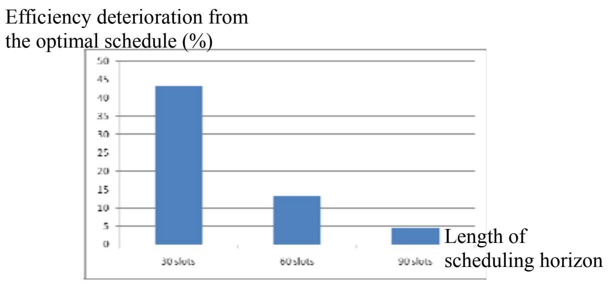


Figure 8. Efficiency of schedules of type-2

C. *Type-3*

An environment consists of 3 machines that can execute same functions with different performances were assumed, where the performances of individual machines were defined so that the 1st, the 2nd and the 3rd machines could complete 2, 3 and 4 unit processes within their 4 time slots, respectively. Jobs consist of 1-8 processes each of which requires 2 time slots of the least performance machine in average were randomly generated.

Table IV shows the results. As same as in type-2 cases, Hungarian scheduler performed well when the length of scheduling horizons were long enough. As shown in Fig. 9, when the length of scheduling horizons were more than 3 times of job length, Hungarian algorithm based scheduler had generated feasible schedules without reducing efficiency more than 5% of optimal schedules.

TABLE IV. SCHEDULING RESULTS FOR TYPE-3 CASES

(a) Scheduling horizon = 16 slots

	Case 10	Case 11	Case 12
Extra slots	2	1	1
Constraint violations	4	4	3
Scheduling time	0.5	0.3	0.3

(b) Scheduling horizon = 32 slots

	Case 13	Case 14	Case 15
Extra slots	2	2	2
Constraint violations	4	2	2
Scheduling time	5	7	7

(c) Scheduling horizon = 64 slots

	Case 16	Case 17	Case 18
Extra slots	1	3	0
Constraint violations	2	41	0
Scheduling time	41	94	21

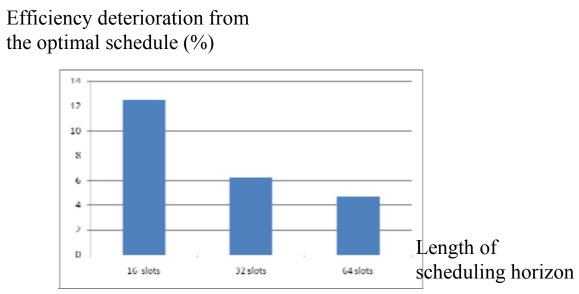


Figure 9. Efficiency of schedules of type-3

V. CONCLUSION

According to the preliminary evaluations, it is expected that Hungarian algorithm based scheduler can generate efficient schedules when machine constraints are not complicated and scheduling horizons are long enough compared with lengths of jobs. However, computation volumes necessary for scheduling must be reduced. As future works, firstly to generate schedules for more complicated cases, price assignment and modification strategies must be enhanced, and secondly to reduce computation times, mechanisms for dividing scheduling horizons and/or for parallelizing iterations must be devised.

REFERENCES

[1] J. W. Herrmann, et al.: "Hierarchical Production Planning with Part, Spatial and Time Aggregation," Proc. of 4th International Conference on Computer Integrated Manufacturing and Automation Technology, 1994

[2] P. Kanchanasevee, et al.: "Contract-Net based Scheduling for Holonic Manufacturing Systems," Proc. of SPIE: Architectures, networks and Intelligent Systems for manufacturing Integration, 1997

[3] P. B. Luh, et al.: "Schedule Generation and Reconfiguration for Parallel Machines," Trans. on Robotics and automation, Vol.6, No.6, 1999

[4] M. E. Aydin, et al.: "Dynamic job-shop scheduling using reinforcement learning agents," Robotics and Autonomous Systems, Vol.33, No.2-3, 2000