**PAPER** *Special Section on Concurrent Systems Technology*

# Timed Petri Net Based Scheduling for Mechanical Assembly——Integration of Planning and Scheduling——

**Akio INABA**[†], ***Member,*** **Fumiharu FUJIWARA**[††], **Tatsuya SUZUKI**[††], ***and*** **Shigeru OKUMA**[††], ***Nonmembers***

**SUMMARY** In scheduling problem for automatic assembly, planning of task sequence is closely related with resource allocation. However, they have been separately carried out with little interaction in previous work. In assembly planning problem, there are many feasible sequences for one mechanical product. In order to find the best assembly sequence, we have to decide the cost function for each task a priori and make decision based on summation of costs in sequence. But the cost of each task depends on the machine which executes the allocated task and it becomes difficult to estimate an exact cost of each task at planning stage. Moreover, no concurrent operation is taken into account at planning stage. Therefore, we must consider the sequence planning and the machine allocation simultaneously. In this paper, we propose a new scheduling method in which sequence planning and machine allocation are considered simultaneously. First of all, we propose a modeling method for an assembly sequence including a manufacturing environment. Secondly, we show a guideline in order to determine the estimate function in $A^*$ algorithm for assembly scheduling. Thirdly, a new search method based on combination of $A^*$ algorithm and supervisor is proposed. Fourthly, we propose a new technique which can take into consider the repetitive process in manufacturing system so as to improve the calculation time. Finally, numerical experiments of proposed scheduling algorithm are shown and effectiveness of proposed algorithm is verified.

***key words:*** *scheduling, Timed Petri Net, assembly*

## 1. Introduction

In scheduling problem for automatic assembly, we need to decide sequence of tasks and allocation of machines. In most of previous researches, these problems have been studied separately. In conventional scheduling problem, for example, Job Shop Scheduling (JSS) [1]–[4] has widely been discussed, however, only resource allocation problem has mainly been focused in JSS. On the other hand, in assembly planning problem, there are many feasible sequences for one mechanical product. In order to find the best assembly sequence, we have to decide the cost function for each task a priori and make decision based on summation of costs in sequence. However, the cost of each task depends on the machine which executes the allocated task and it becomes difficult to estimate an exact cost of each task at planning stage. Moreover, no concurrent operation is taken into account at planning stage. Figure 1 shows an example on this point. The solution in planning problem is sequence (a), because the summation of cost is smaller that of (b). However, production time of sequence (b) is smaller than that of sequence (a), when we can use machine M1 and M2 concurrently. Figure 2 shows another example. The solution in planning is sequence (a), but production time of sequence (b) is much smaller than that of sequence (a), when we can use machine M1 and M2 concurrently. (Here, M○ : ○○ in

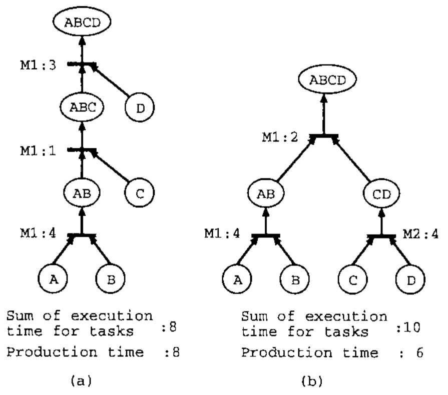


**Fig. 1** An example of planning (1).

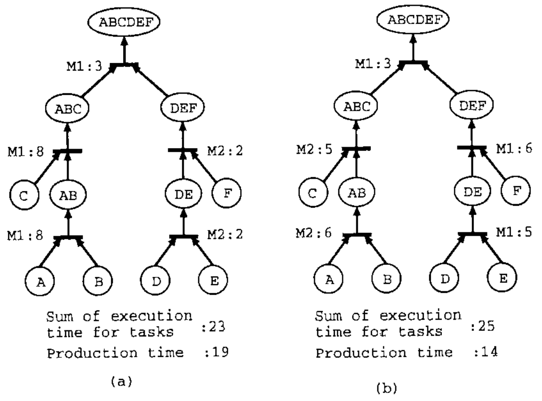


**Fig. 2** An example of planning (2).

Manuscript received September 8, 1997.
Manuscript revised November 10, 1997.

[†]The author is with the Gifu Prefectural Metal Research Institute, 1288 Oze, Seki-shi, 501–3265 Japan.
[††]The authors are with Department of Electrical Engineering, Nagoya University, Furo-cho, Chikusa-ku, Nagoya-shi, 464–8603 Japan.

Figs. 1 and 2 denote machine and execution time of each task in case of using assigned machine.) Therefore, we must consider the sequence planning and the machine allocation simultaneously. On the other hand, in few previous researches, sequence planning and machine allocation were discussed simultaneously. For example, M.F. Sebaaly et al. [8] have been proposed a method based on GA. However, they study the case that the number of assembled product is only one for each kind of product. In this case, the solution may not be good solution for case that the number of assembled product is greater than one.

In this paper, we propose a new scheduling method for assembly problem in which sequence planning and machine allocation are considered simultaneously based on Timed Petri net modeling. Proposed method has also a feature that the solution can be implemented in logical controller because the controlled object is represented by Timed Petri Net.

## 2. Modeling of Assembly Sequences

We construct the model of assembly sequence including a manufacturing environment by doing stepwise refinement of assembly network which consists only of mechanical parts. Homem de Mellow et al. have shown an efficient method for representation of assembly network using AND/OR graph. A Petri Net representation of assembly network is given by

$$N_S = (P_S, T_S, I_S, O_S), \qquad (1)$$

where, $P_S, T_S, I_S$ and $O_S$ are as follows.

$$P_S = \{\text{A set of subassemblies}\}$$
$$T_S = \{t_i\} = \{\text{A set of assembly tasks}\}$$
$${}^\bullet t_i = \{\text{two subassemblies before task } t_i\}$$
$$t_i^\bullet = \{\text{the subassembly after task } t_i\}$$
$$I_S(p_i, t_j) = \begin{cases} 1 & : \; p_i \in {}^\bullet t_j \\ 0 & : \; \text{otherwise} \end{cases}$$
$$O_S(t_i, p_j) = \begin{cases} 1 & : \; p_j \in t_j^\bullet \\ 0 & : \; \text{otherwise} \end{cases}$$

Figure 4 shows a Petri Net representation of assembly network about a ball-point pen shown in Fig. 3.

In [7], the authors have shown a technique to construct AND/OR Petri Net from geometric information of mechanical product.

In this paper, we assume that the manufacturing environment is given as follows (Fig. 5).

- A production system consists of semi-universal assembly machines, a common stack, a product station and a transferring robot.
- Each assembly machine has its own parts feeder to supply parts for machine.

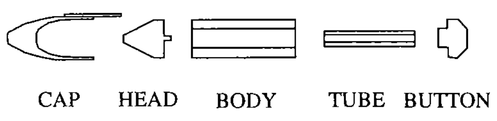


**Fig. 3** A ball-point pen.

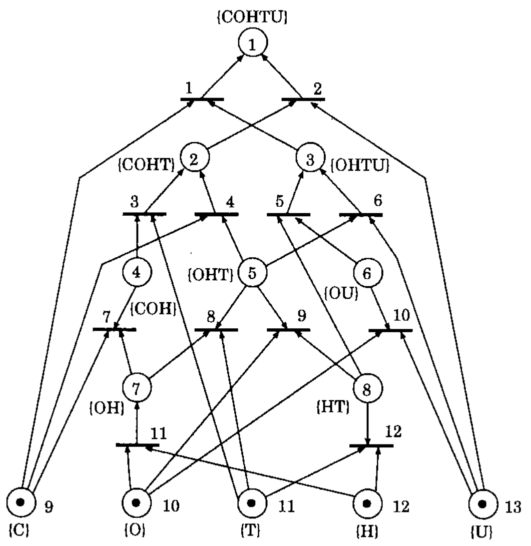


**Fig. 4** A Petri net representation of assembly network about a ball-point pen.

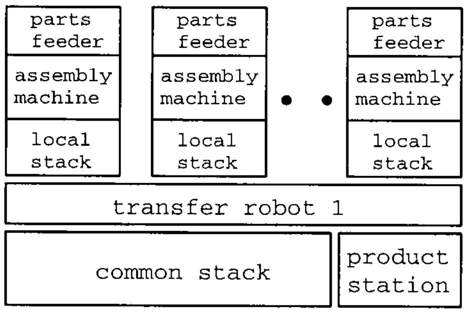


**Fig. 5** Production system.

- Each assembly machines has its own local stack and subassemblies are transferred between machine and common stack through this local stack.
- The capacity of a common stack is limited.

Under this manufacturing environment, each assembly task consists of following steps.

- A transferring robot transfers subassemblies from the common stack to the local stack in assembly machine.
- An assembly machine assembles subassemblies in a local stack (or parts loaded from parts feeders).
- A transferring robot transfers the assembled subassembly from the local stack to the common stack.

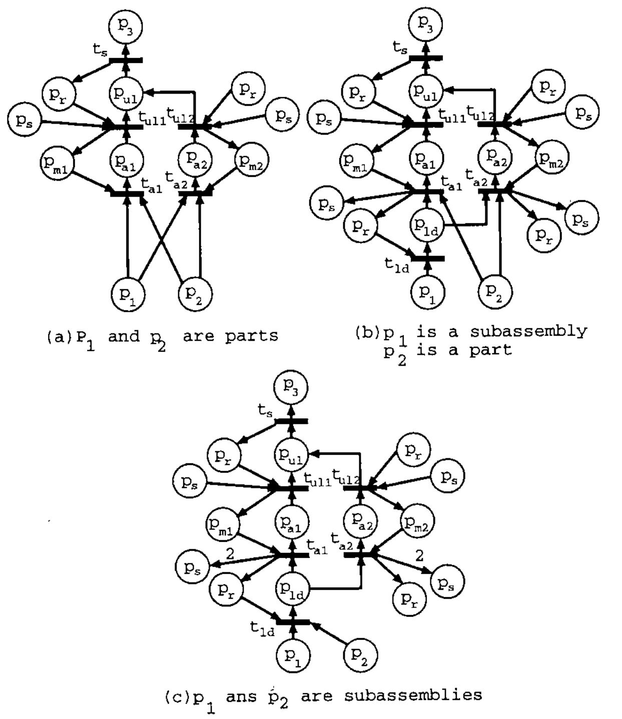


**Fig. 6** Assembly tasks.

Figure 6 shows a Petri Net expression of these steps A mathematical formulation for them is given as follows.

○ A set of places

• A set of places for subassemblies

$$P_{TS} = \{p_1, p_2, p_3\}, \tag{2}$$

where, $p_1$ and $p_2$ are subassemblies before assembling, $p_3$ is the assembled subassembly for $p_1$ and $p_2$.

• A set of places for steps

$$P_{TT} = \begin{cases} \{p_{aj}, p_{ul} | j = 1, N_M\} \\ \qquad : p_1, p_2 \text{ is parts} \\ \{p_{ld}, p_{aj}, p_{ul} | j = 1, N_M\} \\ \qquad : \text{otherwise} \end{cases} \tag{3}$$

where, $p_{ld}$ is a step to load subassemblies from the common stack to a assembly machine. $p_{aj}$ is a step to assemble subassemblies at machine $j$. $p_{ul}$ is a step to unload subassemblies from the assembly machine to the common stack. $N_M$ is number of feasible machines in the environment.

• A set of places for resources.

$$P_{TR} = \{p_r, p_s, p_{mj} : j = 1, N_M\} \tag{4}$$

where, $p_r$ is the transferring robot. $p_s$ is the common stack. $p_{mj}$ is a feasible machine $j$ for an assembly task. Each of these places has a token if it is available.

After all, a set of places for an assembly task is given as follows.

$$P_T = P_{TS} \cup P_{TT} \cup P_{TR} \tag{5}$$

○ A set of transition

$$T_T = \{t_j\} = \begin{cases} \{t_{ak}, t_{ulk}, t_s : k = 1, N_M\} \\ \qquad : p_1, p_2 \text{ are parts} \\ \{t_{ld}, t_{ak}, t_{ulk}, t_s : k = 1, N_M\} \\ \qquad : \text{otherwise} \end{cases} \tag{6}$$

where, $t_{ld}, t_{ak}, t_{ulk}$ and $t_s$ are given as follows.

$${}^\bullet t_{ulk} = \{p_{ak}, p_r, p_s\}, t_{ulk}^\bullet = \{p_{ul}, p_{mk}\}$$
$${}^\bullet t_s = \{p_{ul}\}, t_s^\bullet = \{p_3, p_r\}$$

In case that $p_1$ and $p_2$ are parts.

$${}^\bullet t_{ak} = \{p_1, p_2, p_{mk}\}, t_{ak}^\bullet = \{p_{ak}\}$$

In case that $p_1(p_2)$ is a subassembly and $p_2(p_1)$ is a part.

$${}^\bullet t_{ld} = \{p_1(p_2), p_r\}, t_{ld}^\bullet = \{p_{ld}\}$$
$${}^\bullet t_{ak} = \{p_2(p_1), p_{ld}, p_{mk}\}, t_{ak}^\bullet = \{p_{ak}, p_r, p_s\}$$

In case that $p_1$ and $p_2$ are subassemblies.

$${}^\bullet t_{ld} = \{p_1, p_2, p_r\}, t_{ld}^\bullet = \{p_{ld}\}$$
$${}^\bullet t_{ak} = \{p_{ld}, p_{mk}\}, t_{ak}^\bullet = \{p_{ak}, p_r, p_s\}$$

○ Weight of arcs

$$I_T(p_j, t_k) = \begin{cases} 1 & : \ p_j \in {}^\bullet t_k \\ 0 & : \ \text{otherwise} \end{cases} \tag{7}$$

$$O_T(t_j, p_k) = \begin{cases} 2 & : \ p_k \in t_j^\bullet, p_k = p_s \\ & \quad p_1 \text{ and } p_2 \text{ are parts} \\ 1 & : \ p_k \in t_j^\bullet, (p_k \neq p_s \ \text{ or} \\ & \quad p_1 \text{ or } p_2 \text{ is a part.}) \\ 0 & : \ \text{otherwise} \end{cases} \tag{8}$$

○ A set of time assigned to place

$$A_T = \{a_j\} \tag{9}$$

$$a_j = \begin{cases} \text{execution time} & : \ p_j \in P_{TT} \\ 0 & : \ \text{otherwise} \end{cases} \tag{10}$$

After all, Petri Net of an assembly task is given by

$$N_T = (P_T, T_T, I_T, O_T, A_T). \tag{11}$$

We extend assembly network shown in Fig. 4 to assembly system including resources as follows (Fig. 7). Here, we denote task $i$ by $N_T^i$ in order to distinguish it from other tasks.

(1) We replace each transition of $N_S$ by $N_T^i$.

(2) If successive assembly in one machine is possible, bypass nets are added. Because, we do not need the unloading and reloading steps between assembly tasks.

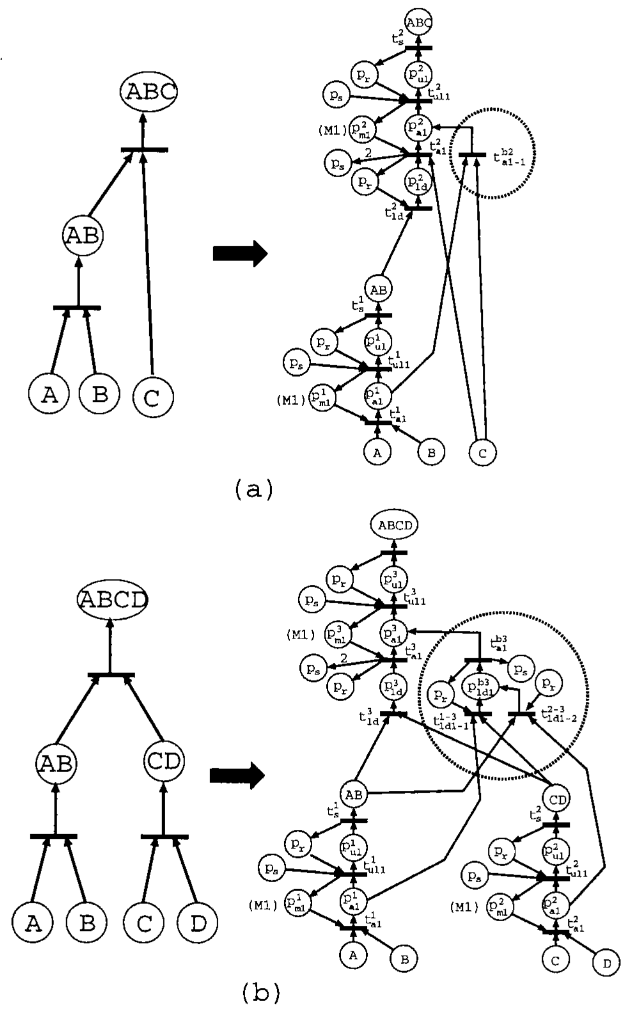


**Fig. 7** Refinement of assembly networks.

Based on the above procedure, we consider the complete Petri Net representation for assembly system including resources. First of all we give the bypass net to the assembly task $p^i_{aj}$ executed with the machine $j$.

○ In case that one of $p^i_1$ and $p^i_2$ is a part

We add the transition which represents the beginning of successive assembly task for the subassembly $p^i_1$ ($p^i_2$) in the machine $p^i_{mj}$, as shown in Fig. 7 (a).

$$t^{bi}_{aj}(k,l) = \begin{cases} \{t^{bi}_{aj-k}\} & : \quad p^i_{mj} = p^k_{ml}, p^i_1(p^i_2) = p^k_3 \\ \phi & : \quad \text{otherwise} \end{cases} \tag{12}$$

where,

$${}^{\bullet}t^{bi}_{aj-k} = \{p^k_{al}, p^i_2\}(\{p^k_{al}, p^i_1\}), t^{bi\bullet}_{aj-k} = \{p^i_{aj}\}$$

• Sets of places and transitions along the bypass net

$$T^i_{pj} = \bigcup_{k=1}^{N_{TA}} \bigcup_{l=1}^{N^k_M} t^{bi}_{aj}(k,l) \tag{13}$$

$$P^i_{pj} = \phi \tag{14}$$

where, $N_{TA}$ is number of all tasks.

○ In case that $p^i_1$ and $p^i_2$ are subassemblies

We add the bypass net (pass 1) which represents pass for the successive assembly task for the subassembly $p^i_1$ in the machine $p^i_{mj}$, and add the bypass net (pass 2) which represents pass for the successive assembly task for the subassembly $p^i_2$ in the machine $p^i_{mj}$, as shown in Fig. 7 (b).

$$t^{1-i}_{ldj}(k,l) = \begin{cases} \{t^{1-i}_{ldj-k}\} & : \quad p^i_{mj} = p^k_{ml}, p^i_1 = p^k_3 \\ \phi & : \quad \text{otherwise} \end{cases} \tag{15}$$

$$t^{2-i}_{ldj}(k,l) = \begin{cases} \{t^{2-i}_{ldj-k}\} & : \quad p^i_{mj} = p^k_{ml}, p^i_2 = p^k_3 \\ \phi & : \quad \text{otherwise} \end{cases} \tag{16}$$

where, ${}^{\bullet}t^{1-i}_{ldj-k} = \{p^k_{al}, p^i_2, p_r\}$, ${}^{\bullet}t^{2-i}_{ldj-k} = \{p^k_{al}, p^i_1, p_r\}$, $t^{1-i\bullet}_{ldk} = t^{2-i\bullet}_{ldk} = \{p^{bi}_{ldj}\}$, $p^{bi}_{ldj}$ is a step to load a subassembly from the common stack.

• Sets of places and transitions along bypass net

$$T^i_{pj} = \begin{cases} T'^i_{pj} \cup \{t^{bi}_{aj}\} & : \quad T'^i_p \neq \phi \\ \phi & : \quad \text{otherwise} \end{cases} \tag{17}$$

$$P^i_{pj} = \begin{cases} \{p^{bi}_{ldj}\} & : \quad T'^i_p \neq \phi \\ \phi & : \quad \text{otherwise} \end{cases} \tag{18}$$

where, $T'^i_{pj} = \bigcup_{k=1}^{N_{TA}} \bigcup_{l=1}^{N^k_M} (t^{1-i}_{ldj}(k,l) \cup t^{2-i}_{ldj}(k,l))$, ${}^{\bullet}t^{bi}_{aj} = \{p^{bi}_{ldj}\}$, $t^{bi\bullet}_{aj} = \{p^i_{aj}, p_r, p_s\}$.

Therefore, Petri Net for an assembly system including resources is specified as follows.

$$N_A = (P_A, T_A, I_A, O_A, A_A), \tag{19}$$

where, $P_A, T_A, I_A, O_A$ and $A_A$ are follows.

$$P_A = P_S \cup \left( \bigcup_{i=1}^{N_{TA}} \left( P^i_{TT} \cup P^i_{TR} \cup \left( \bigcup_{j=1}^{N^i_M} P^i_{pj} \right) \right) \right)$$

$$T_A = \bigcup_{i=1}^{N_{TA}} \left( T^i_T \cup \left( \bigcup_{j=1}^{N^i_M} T^i_{pj} \right) \right)$$

$$I_T(p_i, t_j) = \begin{cases} 1 & : \quad p_i \in {}^{\bullet} t_j \\ 0 & : \quad \text{otherwise} \end{cases}$$

$$O_T(t_i, p_j)$$
$$= \begin{cases} 2 : p_j \in t^{\bullet}_i, p_j = p_s, {}^{\exists} p^k_{ld} \in {}^{\bullet} t_i, \\ \quad p^k_{ld} = \{\text{a task to load two subassemblies from the common stack.}\} \\ 1 : p^i_k \in t^{i\bullet}_j, (p^i_k \neq p_s \text{ or } {}^{\exists} p^k_{ld} \in {}^{\bullet} t_i, \\ \quad p^k_{ld} \neq \{\text{a task to load two subassemblies from the common stack.}\} \\ 0 : \text{otherwise} \end{cases}$$

$$A_A = \{a_i\}$$

$$a_i = \begin{cases} \text{execution time} : p_i \in \cup_{i=1}^{N_{TA}} \left( P^i_{TT} \cup \left( \cup_{j=1}^{N^i_M} P^i_{pj} \right) \right) \\ 0 \qquad\qquad\quad : \text{otherwise} \end{cases}$$

The reason why we have adopted the Timed Place Petri Net (TPPN) is that the marking in TPPN is always determined uniquely. On the contrary, if we use the Timed Transition Petri Net (TTPN), tokens disappear when transitions are firing, then marking may not be determined uniquely.

## 3. Scheduling Algorithm

A scheduling algorithm executed on the Petri Net model has been proposed by Doo Yong Lee et al.[5]. This technique is modified version of $A^*$ algorithm. The details of this algorithm are as follows.

**Algorithm L1**

```
STEP1. Put the initial marking $m_0$ on the list $OPEN$.
STEP2. If $OPEN$ is empty, terminate with failure.
STEP3. Remove the first marking $m$ from $OPEN$ and put it on the list $CLOSED$.
STEP4. If $m$ is final marking, construct the path from the initial marking to the final marking and terminate.
STEP5. Find the enable transition of the marking $m$.
STEP6. Generate the next marking, or successor, for each enabled transition, and set pointers from the next marking to $m$. Compute $g(m')$ for every successor $m'$.
STEP7. For every successor $m'$ of $m$, do the following.
    a: If $m'$ is already on $OPEN$, direct its pointer along the path yielding the smallest $g(m')$.
    b: If $m'$ is already on $CLOSED$, direct its pointer along the path yielding the smallest $g(m')$. If $m'$ requires pointer redirection, move $m'$ to $OPEN$.
    c: If $m'$ is not on either $OPEN$ or $CLOSED$, compute $h(m')$ and $f(m')$ and put $m'$ on $OPEN$.
STEP8. Reorder $OPEN$ in the increasing magnitude of $f$.
STEP9. Go to STEP2.
```

Here, $f(m)$ is an estimate of the cost, i.e., the makespan from the initial marking to final marking along an optimal path which goes through the marking $m$. $f(m) = g(m) + h(m)$. $g(m)$is the current lowest cost obtained from the initial marking to the current marking $m$. $h(m)$ is an estimate of the cost from the marking $m$ to the final marking along an optimal path which goes through the marking $m$.

In scheduling algorithm, it always finds an optimal path if h($m$) satisfies the following condition.

$$h(m) \leq h^*(m) \text{ for all } m \tag{20}$$

where, $h^*(m)$is the actual cost of the optimal path from $m$ to the final marking. The estimate function based on number of remaining tasks was presented in literature[5]. However, no method to estimate the number of remaining tasks has been shown in it. In assembly problem, we can give the estimate as follows. We need $n-1$ tasks to assemble a product from $n$ subassemblies. In this case, number of remaining tasks is $n-1$. We extend this idea to case that the number of each part is $\alpha$. At the intermediate state of assembly process, if the number of subassemblies are $l$, then the number of remaining tasks is $l-\alpha$. Therefore, we define the estimate function $h(m)$ as follows.

$$h(m) = \frac{\sum_{i=1}^{l} f(i) \cdot m(i) - \alpha}{N_M} \cdot C_{min}, \tag{21}$$

where, $\alpha$, $l$ and $N_M$ are the number of products, places and machine, respectively. $m(i)$ is the number of token in place $i$ at marking $m$. $C_{min}$ is minimum cost in all assembly tasks.

$$f(i) = \begin{cases} 1 & : \text{ place } p_i \text{ expresses a subassembly} \\ 0 & : \text{ otherwise} \end{cases}$$

On the other hand, in algorithm L1, it is well known that when $h(m)$ is much smaller than $h^*(m)$, search time becomes bigger. Generally speaking, $h(m)$ shown in the Eq. (21) is much smaller than $h^*(m)$ because (21) does not take into account the cost of transferring task and few tasks with near minimum cost may be performed. In order to take into account this point We redefine $h(m)$ as follows.

$$h(m) = \frac{\sum_{i=1}^{l} f(i) \cdot m(i) - a}{N_M} \cdot (C_a + T_a) \tag{22}$$

where, $C_a$ is mean cost of assembly tasks, $T_a$ is mean cost of transfer tasks.
Although Eq. (22) does not satisfy the condition (20), in many cases, the optimal solution can be obtained. The simulation results on this point will be shown in Sect. 7.

## 4. Reduction of Search Space Using a Supervisor

When we apply the algorithm L1 to a scheduling problem in which many numbers of products are assembled, search space becomes enormous and we need much time to solve it. In order to overcome this difficulty and obtain a quasi optimal solution with small calculation time, we propose a new algorithm adopting the following three strategies.

1) We add state feedback (a supervisor) to assembly Petri Net in order to reduce the size of state space.

2) We try to search a quasi optimal solution which has repetitive operation.

3) We make a limitation for the capacity of list OPEN.

As for 3), Limitation of capacity for list OPEN enables us to search a quasi optimal solution with small calculation time (See [6]). In the remaining part of this chapter we state about 1). Also we state about 2) in the next chapter. Supervisor controls a firing of each transition based on markings of assembly system and reduce the search space (Fig. 8). Control place is attached to each transition of assembly Petri Net. Control place and transition are linked by dual arcs and each control place has a token at an initial marking (Fig. 9). Supervisor controls a token in control place based on state of marking and closed loop specification.

For example, when execution of a task is forbidden, the token in the control place connected to corresponding transition is removed. In the same way, fire of transitions can be controlled by supervisor. On the other hand, this proposed method can be viewed as a new search algorithm based on the combination of $A^*$ algorithm and supervisor. In other words, we can say that a 'rule based algorithm' (supervisor) is built in the $A^*$ algorithm in order to reduce the search space.

In this paper, we introduce a supervisor which handles a problem of parts allocation to parts feeders.

Here, we consider the following control specification.

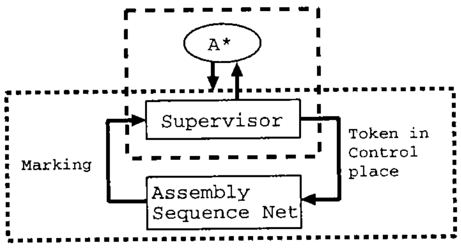


**Fig. 8** Search Algorithm.

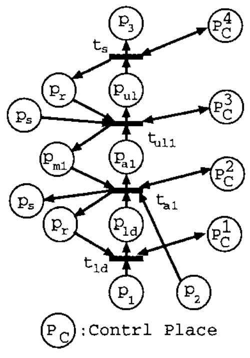


**Fig. 9** Control place.

- Same kind of part is loaded from same parts feeder.

This is natural specification because, in practical manufacturing, there are few cases in which same kind of parts are loaded from different parts feeders. We show how to construct the control logic of supervisor for this problem.

## 4.1 The Control of Allocation of Parts Feeders

The supervisor controls tokens in control places based on Task List table (Table 1) and Control List table (Table 2). Task List table includes information on the relation of machine number, loaded parts and places corresponding to the parts loading step. Control List table includes information on the relation of parts, loaded machine and transitions for loading task. For example, there is a token in place $p^1_{a1}$ at a marking $\boldsymbol{m}$ in an assembly system shown in Fig. 10. Supervisor detects that parts A and B are allocated to machine $M_1$ by Task List table, and forbids the firing of transition about steps loading machines except for machine $M_1$ with parts A and B by Control List table. In this case, transition $t^1_{a2}$ is forbidden.

## 4.2 Avoidance of Deadlock Occurring from Control of Parts Allocation

In an assembly system shown in Fig. 10, when $t^2_{a2}$ fires and a token is provided in $p^2_{a2}$ (An assemble step for part C and D by machine $M_2$ is performed), parts C and D are allocated to the parts feeder in machine $M_2$. Then, supervisor introduced in previous section forbids

**Table 1** Task List table.

| Place | Machine No | Parts |
|---|---|---|
| $p^1_{a1}$ | 1 | A,B |
| $p^1_{a2}$ | 2 | A,B |
| $p^2_{a1}$ | 1 | C,D |
| $p^2_{a2}$ | 2 | C,D |
| $p^3_{a1}$ | 1 | C |
| $p^3_{a2}$ | 2 | C |
| $p^5_{a1}$ | 1 | D |

**Table 2** Control List table.

| Parts | Machine No | Transition |
|---|---|---|
| A | 1 | $t^1_{a1}$ |
| A | 2 | $t^1_{a2}$ |
| B | 1 | $t^1_{a1}$ |
| B | 2 | $t^1_{a2}$ |
| C | 1 | $t^2_{a1}, t^3_{a1}, t^{b3}_{a1-1}$ |
| C | 2 | $t^2_{a2}, t^3_{a2}, t^{b3}_{a2-1}$ |
| D | 1 | $t^2_{a1}, t^5_{a1}, t^{b5}_{a1-3}$ |
| D | 2 | $t^2_{a2}$ |

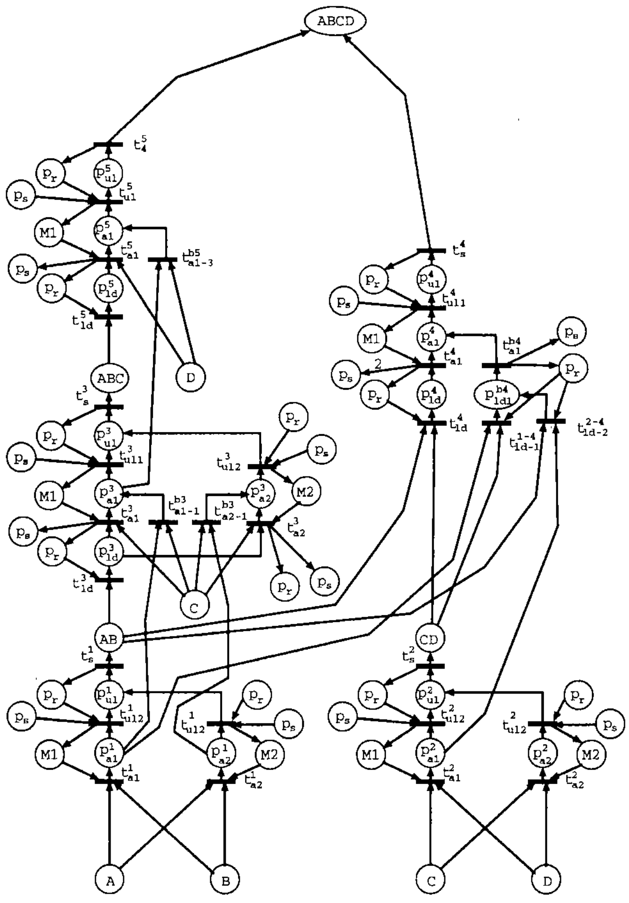


**Fig. 10** An assembly system.

firing of transition $t^2_{a1}$, $t^5_{a1}$, $t^{b5}_{a1-3}$. This leads to that execution of the assembly step for subassembly ABC and part D is impossible. However, when one of transitions $t^3_{ld}$, $t^{b3}_{a1-1}$ and $t^{b3}_{a2-1}$ is fired, it starts to make subassembly ABC and after all, deadlock will occur. In order to avoid this type of deadlock, the supervisor has to forbid firing of transitions $t^3_{ld}$, $t^{b3}_{a1-3}$ and $t^{b3}_{a2-1}$, when firing of transitions $t^5_{a1}$ and $t^{b5}_{a1-3}$ are forbidden. Therefore, we make Subassembly List table (Table 3) and Task Group List table (Table 4). Subassembly List table includes information on the relation of subassembly and group of assembly task including **s**. $T_{out}$ is the group of task to assemble **s** and another subassembly or part. $T_{in}$ is the group of task to make **s**. Task Group List table includes information on the relation of task group, controlled transition, entry transition and entry place. The information on the controlled transition is used in order to detect forbidden tasks. Entry transitions express beginning task in a task group and entry places are output places of entry transitions. Entry transitions are used to forbid all steps in task group and entry places are used to detect the beginning of a task in the task group. Based on this consideration, supervisor controls tokens in control places as follows.

**Table 3** Subassembly List table.

| Subassembly | $T_{in}$ | $T_{out}$ |
|---|---|---|
| AB | A+B | AB+CD,AB+C |
| CD | C+D | AB+CD |
| ABC | AB+C | ABC+D |

**Table 4** Task Group List table.

| Task group | controlled transition | entry transition | entry place |
|---|---|---|---|
| A+B | $t^1_{a1}(p^1_{a1})$ $t^1_{a2}(p^1_{a2})$ | $t^1_{a1}$ $t^1_{a2}$ | $p^1_{a1}, p^1_{a2}$ |
| C+D | $t^2_{a1}(p^2_{a1})$ $t^2_{a2}(p^2_{a2})$ | $t^2_{a1}$ $t^2_{a2}$ | $p^2_{a1}, p^2_{a2}$ |
| AB+C | $t^3_{a1}(p^3_{a1})$ $t^3_{a2}(p^3_{a2})$ | $t^3_{ld}, t^{b3}_{a1-1}$ $t^{b3}_{a2-1}$ | $p^3_{ld}, p^3_{a1}$ $p^3_{a2}$ |
| AB+CD | $t^4_{a1}(p^4_{a1})$ | $t^4_{ld}, t^{1-4}_{a1-1}$ $t^{2-4}_{a1-2}$ | $p^4_{ld}, p^{b4}_{ld1}$ |
| ABC+D | $t^5_{a1}(p^5_{a1})$ | $t^5_{ld}, t^5_{a1-3}$ | $p^5_{ld}, p^5_{a1}$ |

1) When all tasks in $T_{out}$ are disabled, forbid the execution of all tasks in $T_{in}$
2) When a task in $T_{in}$ is executed, and only one task in $T_{out}$ can be performed, disable the execution of all tasks of which execution disable it.
3) If all tasks in task group are forbidden, disable the firing of all entry transitions of it.

For example, supervisor introduced in previous section forbids $t^5_{a1}$ and $t^{b5}_{a1-3}$, when a token is in $p^2_{a2}$. The supervisor identifies that $t^5_{a1}$ for ABC+D is forbidden referring Task Group List table, and forbid the execution of $t^5_{ld}$ by execution procedure 3. Moreover, the supervisor identifies that all tasks in $T_{out}$ for subassembly ABC are forbidden referring Subassembly List table, and forbid AB+C in $T_{in}$ based on procedure 1 and forbids $t^3_{ld}$, $t^{b3}_{a1-1}$ and $t^{b3}_{a2-1}$ referring Task Group List table.

When $t^3_{ld}$ is fired and a token is provided into $p^3_{ld}$, supervisor identifies the execution of task AB+C referring Task Group List table. Supervisor executes procedure 2 because only $p^5_{a1}$ belongs to $T_{out}$ of subassembly ABC of which $T_{in}$ includes AB+C, referring Subassembly List table. We assume that there is a token in $p^5_{a1}$, supervisor forbids $t^2_{a2}$ according to procedure in previous section.

The control of this deadlock doesn't guarantee that all deadlock can be avoided because supervisor uses only local information. However, supervisor can remove many deadlocks and improve efficiency of search.

## 5. Speedup Based on Searching a Quasi Optimal Solution Including Repetitive Process

There are several case that the optimal solution includes repetitive process in manufacturing system, when infinite number of products are assembled. In this paper, paying attention to this point, we try to reduce calculation time by searching a quasi optimal solution including repetitive process.

We detect repetitive process as follows. First, we define marking time, sojourn time of a token and equivalent marking.

- **Marking time**

Marking time is time, when a transition becomes enable in marking.

- **Sojourn time of a token in a marking**

Sojourn time of a token is time interval from marking time to a time when the token comes to be able to get out. If sojourn time is negative, it is defined as 0.

- **Equivalent marking for $m$**

Equivalent marking for $m$ is a marking in which number of token for each place except for places corresponding to parts and products is equal to that of $m$ and in which sojourn time of each token in place except for places corresponding to parts and products is smaller than that of $m$ or equal to that of $m$.

Figure 11 shows equivalence of markings. There is no place corresponding to parts and products in Fig. 11. Each number assigned place represents a time when the token comes to be able to get out. Each number in parenthesises represents sojourn time. In Fig. 11 (a), marking time is 10 because transition $t_2$ becomes enable at time 10. Sojourn time of the token in place $p_1$ is $15 - 10 = 5$. The marking (b) is equivalent marking for the marking (a) because number of token for each place in marking (b) is equal to that of marking (a) and sojourn time of each token for each place in marking (b) is smaller than that of marking (a) or equal to that of marking (a). The marking (c) is not equivalent marking for the marking (a) because sojourn time of token for place $p_3$ is longer than that of marking (a). The marking (d) is not equivalent marking for the marking (a), because number of token for place $p_2$ in marking (d) is not equal to that of marking (a).

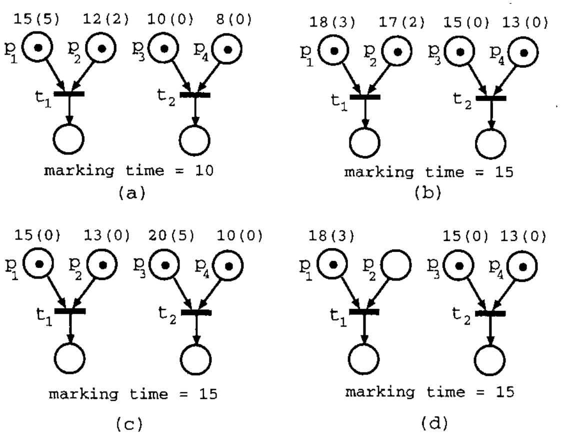


**Fig. 11** Equivalence of markings.

When a marking $m$ at step 3 in algorithm L1 is compared with its parent marking and number of tokens in places corresponding to products changes, we examine repetitive process. Repetitive process is detected by searching a marking $m'$ satisfying following conditions on list along the path from $m$ to initial marking $m_0$.

- **Condition 1**

$m'$ is a equivalent marking of $m$.

- **Condition 2**

Number of tokens in every place corresponding to products and parts changes between $m'$ and $m$.

Here condition 1 guarantees that we can perform firing sequence between $m'$ and $m$ from $m$ with same firing timing. Condition 2 expresses that loading of all parts and production of product are done. The reason why we detect repetitive process using marking is that we do not have to consider length of repetitive process. A scheduling algorithm considering repetitive process is given as follows.

**Algorithm L2**

**STEP1.** Put the initial marking $m_0$ on the list $OPEN$.

**STEP2.** If $OPEN$ is empty, terminate with failure.

**STEP3.** Remove the first marking $m$ from $OPEN$ and put $m$ on the list $CLOSED$.

**STEP4.** When marking $m$ satisfies following condition, examine a repetitive process.

a: Number of token in the place for major product is equal to maximum value for one of marking in OPEN.

b: Number of tokens in each place for products about marking $m$ is different from one of parent marking.

If repetitive process is detected, generate marking $m^r$ firing transition according to repetitive process. Set pointers from $m^r$ to $m$. Compute $g(m^r)$ and put $m^r$ on the list $CLOSED$. Regard $m^r$ as $m$. Empty list $OPEN$.

**STEP5.** If $m$ is final marking, construct the path from the initial marking to the final marking and terminate.

**STEP6.** Find the enable transition of the marking $m$

**STEP7.** Generate the next marking, or successor, for each enabled transition, and set pointers from the next marking to $m$. Compute $g(m')$ for every successor $m'$. Control tokens in control places by the supervisor.

**STEP8.** For every successor $m'$ of $m$, do the following.

**Table 5** Production time of tasks.

| Task | $P_1$ | $P_2$ | Task | $P_1$ | $P_2$ |
|---|---|---|---|---|---|
| 1 (M1) | 2 | 2 | 5 (M2) | 4 | 4 |
| 1 (M2) | 4 | 3 | 6 (M1) | 3 | 4 |
| 2 (M1) | 3 | 3 | 6 (M2) | 4 | 5 |
| 2 (M2) | 2 | 4 | 7 (M1) | 3 | 4 |
| 3 (M1) | 3 | 3 | 7 (M2) | 4 | 5 |
| 3 (M2) | 2 | 4 | 8 (M1) | 3 | 3 |
| 4 (M1) | 4 | 4 | 9 (M1) | 3 | 3 |
| 4 (M2) | 2 | 5 | 10 (M1) | 2 | 2 |
| 5 (M1) | 2 | 3 | | | |

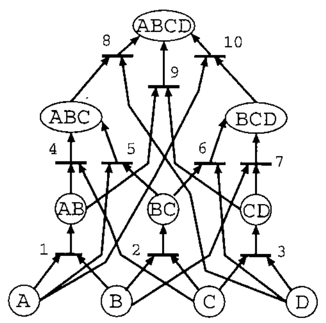


**Fig. 12** An assembly network for a product.

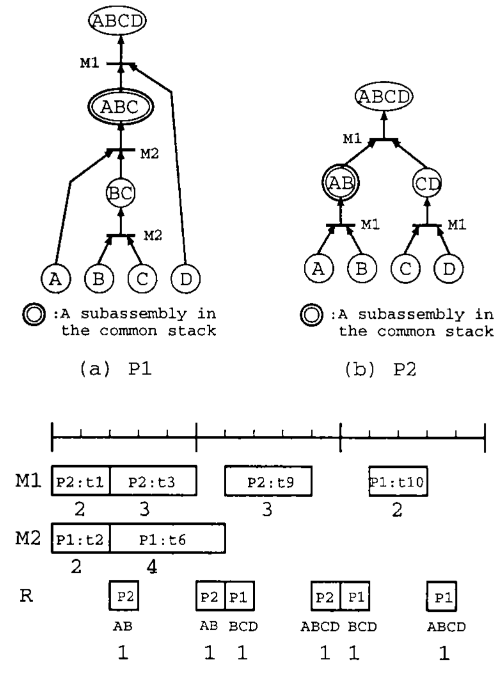


**Fig. 13** Assemble schedule of example.

a: If $m'$ is already on $OPEN$, direct its pointer along the path yielding the smallest $g(m')$.

b: If $m'$ is already on $CLOSED$, direc its pointer along the path yielding the smallest $g(m')$. If $m'$ requires pointer redirection, move $m'$ to $OPEN$.

c: If $m'$ is not on either $OPEN$ or $CLOSED$, compute $h(m')$ and $f(m')$ and put $m'$ on $OPEN$.

**STEP9.** Reorder $OPEN$ in the increasing magnitude of $f$. If number of elements in list $OPEN$ is greater than $L_{open}$, eliminate the markings of which value $f$ is larger, so that number of elements is smaller than $L_{open}$.

**STEP10.** Go to **STEP2.**

Here, $L_{open}$ is capacity of list $OPEN$.

This algorithm does not detect the repetitive operation with a marking including less number of products in OPEN, since this kind of solution is likely to far from optimal solution.

## 6. Example

We show an example of scheduling for assembly sequence with two assembly machines $M_1$ and $M_2$. Table 5 shows execution time of tasks for products $P_1$ and $P_2$. $P_1$ and $P_2$ consist of four parts and their assembly network is given in Fig. 12. In Table 5, we assume that task 8, 9 and 10 can be executed by only $M_1$. Cost of transferring task is 1 per one subassembly. The characteristics of machines for each product is follows.

$P_1$: There is no explicit relation between execution time by machine 1 and 2.

$P_2$: Execution time of $M_1$ is smaller than that of $M_2$ at each task.

We consider to assemble a product $P_1$ and a product $P_2$ concurrently using same set of common stack, assembly machine and transferring robot. Figure 13 shows the solution of this scheduling problem. We can identify the effectiveness of scheduling considering both sequence planning and machine allocation concurrently. All tasks of $P_2$ are allocated to $M_1$, because execution time of $M_1$ is smaller than that of $M_2$ at each task.

## 7. Evaluation of Computational Amount

### 7.1 Effect of Estimate Function

In order to confirm the effectiveness of proposed estimate function we compare the solution and calculation time based on (22) with them based on (21) in case of assembling two products of which assembly network is given by Fig. 12. Costs of assembly step in the products take random value of which range is from 2 to 7. Cost of transferring task is 1 per one subassembly. Calculation times listed on Table 6 are average time of 50 trials. The computer used for calculation is CELEBRIS GL 6200 (pentium pro 200 MHz).

**Table 6** Execution time (1).

| | Equation (21) | Equation (22) |
|---|---|---|
| production time | 15.1 | 15.4 |
| execution time | 12.2 s | 3.9 s |

**Table 7** Execution time (2).

| | consideration of repetitive process | consideration without repetitive process |
|---|---|---|
| production time | 215.3 | 215.2 |
| execution time | 33.9 s | 1108.6 s |

In Table 6, we can not identify difference of solution between Eqs. (21) and (22). But calculation time using Eq. (22) is smaller than the solution using Eq. (21). Thus, effectiveness of Eq. (22) is verified.

### 7.2 Effect of Searching a Repetitive Solution

In order to confirm the effectiveness of detecting the repetitive solution, we compare the solution and calculation time taking into account the repetitive process with the case not considering the repetitive process in case of assembling 30 products of which assembly network is given by Fig. 12. Costs of assembly task in the products take random value of which range is from 2 to 7. Cost of transferring task is 1 per one subassembly. Limitation of capacity for list OPEN is 200 and the supervisor is used in both cases. Calculation times listed on Table 7 are average time of 30 trials in which we can calculate solutions within two hours. We calculated solutions using the computer CELEBRIS GL 6200 (pentium pro 200 MHz).

In Table 7, we can not identify difference of solution between two algorithms, but calculation time considering repetitive process is much smaller than that of normal search. Thus, effectiveness of considering repetitive process is verified.

## 8. Conclusion

In this paper, we have proposed a new scheduling method in which sequence planning and machine allocation are considered simultaneously. First of all, we have proposed a modeling method for an assembly sequence including a manufacturing environment. By using the model based on timed Petri Net, we can always obtain feasible sequences without any consideration. Secondly, we have shown a guideline in order to determine the estimate function in $A^*$ algorithm for assembly scheduling. By using proposed estimate function the calculation time would be reduced with little loss of accuracy of solution. Thirdly, a new search method based on combination of $A^*$ algorithm and supervisor has been proposed. Fourthly, we have proposed a new technique which can take into consider the repetitive process in manufacturing system so as to improve the calculation time. Finally, numerical experiments of proposed scheduling algorithm has been shown and effectiveness of proposed algorithm has been verified. Our future work is evaluation of computational amount about size of assembly systems.

## References

[1] S.M. Johnson, "Optimal two-and-three-stage production schedules with setup times included," Nav. Res. Log. Quart., vol.1, no.1, pp.61–68, 1954.

[2] J.R. Jackson, "An extension of Johnson's results on job-lot Scheduling," Nav. Res. Log. Quart, vol.3, no.3, pp.201–203, 1956.

[3] J.H. Blackstone, D.T. Phillips, and G.L. Hogg, "A State-of-the-art survey of dispatching rules for manufacturing job shop operations, international journal of production research," vol.20, no.1, pp.27–45, 1982.

[4] K. Tanaka and N. Ishii, "Scheduling and Simulation," SICE, 1955.

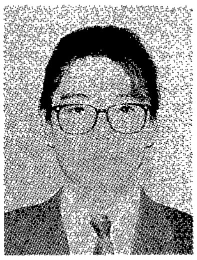
[5] D.Y. Lee and F. DiCesare, "Petri Net-Based Heuristic Scheduling for Flexible Manufacturing," Petri Nets in Flexible and Agile Automation, ed. MengChu Zhou, Kluwer Academic Publishers, Boston, pp.149–188, 1995.

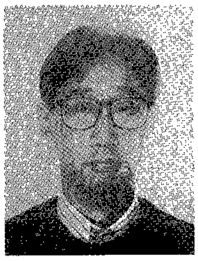
[6] D.Y. Lee and F. DiCesare, "Scheduling flexible manufacturing systems using Petri Nets and heuristic search," IEEE Trans. Robotics and Automation, vol.10, no.2, pp.123–132, 1994.

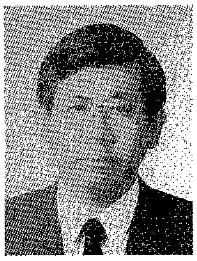
[7] A. Inaba, T. Suzuki, and S. Okuma, "Generation of assembly or disassembly sequences based on topological operations," Trans. of the Japan Society of Mechanical Engineers (C), vol.63, no.609, pp.1795–1802, 1997.

[8] M.F. Sebaaly and H. Fujimoto, "Integrated planning and scheduling for multi-product job-shop assembly based on genetic algorithms," Proc. of the 6th IFIP TC5/WG5.7 International Conference on Advances in Production Management Systems-APMS '96, Kyoto, Japan, pp.557–562, 1996.

**Akio Inaba** was born in Gifu, Japan, in 1961. He received the BE degree in Electronic Engineering from Gifu University, Japan in 1983. From 1984 to 1989, he was with Gifu Prefectural Industrial Technology Research Center. Since 1990, he has been with Gifu Prefectural Metal Research Institute. His current research interests are in discrete-event-system and the areas of robotics. Mr. Inaba is a member of the Information Processing Society of Japan, the Society of Instrument and Control Engineers of Japan, and the Robotic Society of Japan.


**Fumiharu Fujiwara** was born in Osaka, Japan, in 1963. From 1982 to 1988, he was in the Department of Call Physiology, National Institute for Physiological Sciences, Okazaki. Since 1988, he has been an Research Assistant of the Department of Electrical Engineering of Nagoya University. He received the BE in Electrical-Electronic Engineering from Meijo University, Japan in 1991. His main works are management and maintenance of computers and development of programming.


**Tatsuya Suzuki** was born in Aichi, Japan, in 1964. He received the BE, ME and PhD degrees in Electronic-Mechanical Engineering from Nagoya University, Japan in 1986, 1988 and 1991, respectively. Since 1991, he has been an assistant Professor of the Department of Electrical Engineering of Nagoya University. His current research interests are in motion control, discrete-event-system and learning systems. Dr.Suzuki is a member of the Institute of Electrical and Electronics Engineers, the Institute of Electrical Engineers of Japan, the Society of Instrument and Control Engineers of Japan, the Robotic Society of Japan, and Institute of System Control and Information Engineers.


**Shigeru Okuma** was born in Gifu, Japan, in 1948. He received the BE, ME and PhD degrees in Electrical Engineering from Nagoya University, Japan in 1970, 1972 and 1978, respectively. He also received the ME degree in Systems Engineering from Case Western Reserve University, Cleveland, OH in 1974. From 1977 to 1984, he was with the Department of Electrical Engineering of Nagoya University as an Assistant Professor. From 1985 to 1990, he was with the Department of Electronic-Mechanical Engineering of Nagoya University as an Associate Professor. Since December 1990, he has been a Professor of the Department of Electrical Engineering of Nagoya University. His research interests are in the areas of robotics, power electronics and emergent soft computers. Dr.Okuma is a member of the Institute of Electrical Engineers of Japan, the Society of Instrument and Control Engineers of Japan, the Robotic Society of Japan, Institute of System Control and Information Engineers, and IEEE.