Fully Automated Robotic Assembly Cell : Scheduling and Simulation

Ham-Huah Hsu and *Li-Chen Fu*

Department of Computer Science and Information Engineering
National Taiwan University, Taipei, Taiwan, R.O.C.

Abstract

In this paper, we propose the scheduling methodology for a multi-robot assembly cell, which is then integrated into a simulation environment. Since modeling a multi-robot assembly system is difficult and tedious, a systematic method is proposed to transfer the AND/OR product assembly graph and the domain knowledge to assembly rules. Given this rule-base knowledge, an inference engine first generates all possible subsequent assembly tasks, and then a search algorithm finds the optimal ones. These promising tasks thus become the operation commands assigned to the assembly system. A simulation environment of a two-robot assembly cell is built and an experiment is performed using the proposed scheduling strategy. Satisfactory performance has been demonstrated.

1 Introduction

Assembly scheduling plays an important role in assembly production system. Since robots are widely recognized to be highly flexible and manipulable, they are the most suitable for undertaking those assembly tasks. Generally, automatically generating assembly tasks and scheduling it for a multi-robot assembly system can be divided into the following three steps. The first step is to represent the state for multi-robot assembly; the second step is to transfer the assembly graph representing feasible assembly sequences into various assembly rules; the third step is to infer a suitable schedule of assembly tasks by searching over the rule base. One possible strategy to generate an optimal assembly schedule efficiently is to integrate assembly scheduling and simulation of a multi-robot assembly system. By applying a simulation tool, such as CimStation, the simulation executes assembling tasks from assembly planning. Given the assembly rules and via simulation, an optimized schedule of assembly tasks can be generated. Of course, the motion planning problem to avoid collision in a multi-robot assembly cell also needs to be considered, and the solution method should be embedded in the simulation.

The work by Kanehara, et al. [8] presented an assembly planning method using Petri-Net. Based on assembly AND/OR graph [2-4], they modeled an assembly system with Petri-Net, and proposed some state shift equations to manipulate the Petri-Net model. Based on this Petri-Net and equations, a heuristic search, AO*, is used to solve the optimization problem which minimizes a summation of weights assigned to all tasks in the assembly sequence. Unfortunately, it lacked consideration of the configuration of assembly cell (e. g. fixture, tool, machine, and so on) and treated such a problem as only with processing machines. The work by Huang and Lee [7] also proposed a knowledge-based approach, but the method failed to include the run-time information of an assembly cell, like the assembly time of each assembly task which may not be fixed. This scheduling methods with fixed assembly time may be more proper for the assembly tasks by flow line machine. The other researchers such as Shin and Zheng[5] and Sriskandarajah et al.[6] all have treated such a problem but without taking robots as processing machines. Also, there they formulated the problem more like a flow line machines, which is quite different from the one considered in this paper. Here, we are the first to integrate assembly scheduling into the simulation of multi-robot assembly. To mimic the actual assembly cell in our laboratory, a simulation environment of an actual multi-robot assembly system is constructed. Besides, a motion planning algorithm adapted from [1] is applied here to prevent the robots from running into collision.

2 Cell Description

Consider a two-robot assembly cell dedicated to assembling various types of mechanical parts serially sent in through a conveyor belt. The assembled products are then serially sent out again to an automated storage device. The cell is equipped with the several hardwares such as robots, conveyor belt, CCD cameras, assembly table, fixture, rotary buffer, part feeder, etc.

Each product to be assembled contains a number of parts, and all its assembly cell are assumed to be explicitly analyzed and to be stored in a knowledge base via an AND/OR graph representation. Within each assembly sequence for a type of assembly task, each assembly operation to form a subassembly is associated with a time cost, which is the time to finish the assembly operation. To avoid two robots colliding with each other in the common working cell, a robot must have a motion plan before its movement.

In order to describe task of assembly clearly, suppose that there are k types of assembly tasks, and each type has to be performed N_i times so that there are totally $N = N_1 + N_2 .. + N_k$ products to be assembled. Let the parts be fed into the cell in a serial order at a constant rate, and let each part move when staying on the conveyor belt at a constant speed. Furthermore, if we name those robots as robot a and robot b, respectively, we assume that both of them will

IEEE International Conference
on Robotics and Automation
0-7803-1965-6/95 $4.00 ©1995 IEEE

be independently handling m_a and m_b assembly sites, respectively, on the assembly table. This implies that there can be at most $m_a + m_b$ products undergoing concurrent assembly processes. For simplicity again, we will not allow any of the assembly processes to be preemptive, which means that any sub-assembly sitting in some assembly site will not be moved out of that site before it becomes a complete assembly.

Assembly scheduling in a multi-robot assembly system would render the system highly efficient for assembly tasks. The AND/OR graph for assembly is convenient for representation of the assembly products. Assume all the mechanical assembly sequences of each product are explicitly analyzed and stored a priori via AND/OR graph representation. According to the AND/OR graph for assembly, assembly modeling automatically generates assembly rules for robots in the multi-robot assembly cell. Those rules describe all of the assembly tasks in the multi-robot assembly workcell. (e.g., if the AND/OR graph have a node $part_k$ which connects subnodes $part_i$ and $part_j$, then there have some assembly rules expressing that $part_i$ and $part_j$ are assembled into $part_k$ by any robot in the assembly workcell). The assembly scheduling generates all possible assembly tasks, according to those assembly rules. And a searching algorithm is applied to find the optimal assembly tasks among those possible assembly tasks.

3 Assembly Scheduling

The configuration of assembly scheduling is shown in Figure 3 . First, modeling of the assembly systems is that transferring the product assembly AND/OR graphs into the assembly rules for the assembly scheduling. The figure is shown in Figure 4. The detailed modeling method is explained later. According to the assembly rules and the state of the multi-robot system, the inference engine then generates all of the possible assembly tasks, among which, the search algorithm searches for the optimal assembly tasks for the multi-robot assembly system. In order to infer over the assembly system status, the state is used for description of the present assembly status. Finally, the robots will execute this optimal command from the generated schedule. Whenever there is a robot ready for assembly, the assembly scheduling mechanism will repeats the above scheduling procedure again to generate now command for that robot.

3.1 State Representation

The state in fact represents the situation with assembly tasks, whose state information will be used for the assembly scheduling to generate all possible subsequent assembly tasks for robots. Some necessary information to express assembly tasks in a multi-robot assembly system includes the following :

1. **Robot** : which expresses the status (ready or busy) of robots in the multi-robot assembly system.
2. **Part** : which expresses the part type and its identity.
3. **Hand of robot** : which expresses whether the robot hand is empty or not. For examples, if there is a part whose type is $type_i$ and identity is $identity_i$ on the hand of the robot Adept, then we have
   (hand Adept $type_i$ $identity_i$)
4. **Parts located on the sites of the buffer** : which expresses the status about how parts (or subassemblies) are located on the sites of the buffer in the multi-robot assembly system. For example, there is a part whose type is $type_i$ and identity is $identity_i$ on the first site of the buffer but nothing is on the second site of the buffer, then we have
   (site buffer 1 $type_i$ $identity_i$) (site buffer 2 empty)
5. **Part located on the sites of the robot** : which expresses the status about how parts (or such assemblies) are located on the sites of the robots in the multi-robot assembly system. For example, there is a part whose type is $type_i$ and identity is $identity_i$ on site 1 of the robot Adept but nothing is on its site 2, then we have
   (site Adept 1 $type_i$ $identity_i$) (site Adept 2 empty)
6. **Time of assembly operation** : which expresses the time of assembly operation. For examples, if there is an assembly operation, named op_x, which takes $time_x$ seconds to finish, then we have
   (time op_x $time_x$)

The state of a multi-robot assembly system defined here must contain the above six items of information. For example, the state of the multi-robot assembly system is the following:
(robot Adept ready) (robot A-arm busy)
(site Adept 1 $type_i$ $identity_i$)
(site A-arm 2 $type_j$ $identity_j$)
(site buffer empty) (time op_x 10)

3.2 Assembly Rules

In this subsection, we will start from the AND/OR assembly graph representation and look for all the relevant assembly rules to facilitate the task of assembly scheduling.

The nodes in the AND/OR graph are the subsets of the all of stable subassemblies. The hyper arcs correspond to the geometrically and mechanically feasible assembly tasks. The AND/OR graph shown in the Figure 1, here, are three hyper arcs, which correspond to op_1, op_2, and op_3, and there are seven nodes, which correspond to subassemblies $part_1$, $part_2$, $part_3$, $part_4$, $part_A$, $part_B$ and $part_C$. In particular, $part_1$ and $part_2$ are assembled to $part_A$ using operation op_1, $part_A$ and $part_3$ are assembled to $part_B$ using operation op_2, and $part_C$ and $part_4$ are assembled to $part_C$ using operation op_3. First, the structure of a rule is described in the following paragraph. Each rule is composed of five components :

- Precond part: which expresses the condition under which this rule will be executed.
- Delete part: which expresses some conditions of the system state that will be deleted when the rule is executed.
- Add part: Which expresses some condition of the system that will be added when rule is executed.
- Delay part: which expresses the delay time of adding condition to the system state.
- Action part: which generates actual command to robots.

Assume there is sub-graph of the AND/OR graph shown in the Figure 1 which indicates that $part_i$ and $part_j$ is assembled to $part_k$ using operation op_x. Assume the following rule correspond to this op_x operation, then all the relevant components as mentioned above are given below.
Precond:

(robot ?rid ready) (site ?rid ?num1 $part_i$?pid1)
(hand ?rid $part_j$?pid2) (time ?rid op-x ?time)
Delete:
(robot ?rid ready) (site ?rid ?num1 $part_i$?pid1)
(hand ?rid $part_j$?pid2)
Action:
(?rid op-x ?num1 ?pid1 ?pid2)
Delay:
(?time)
Add:
(robot ?rid ready) (site ?rid ?num1 $part_k$?pid1)

This rule can be interpreted as follows: In the beginning, robot "rid" is ready for assembly, and the "?num1" site of the robot has a part whose type is $part_i$ and identity is "?pid1", and the robot hand has a part whose type is $part_j$ and identity is "?pid2". These conditions are described in "Precond" of this rule. If these conditions are satisfied, then the rule is executed, and the following conditions will thus be deleted from the system state: robot "?rid" is not ready for assembly, and both the part located on "?num1" site of robot "?rid" and the part located on the hand of the robot are not available again for assembly. Those conditions are described in "Delete" of the rule. If the operation op_x takes "?time" seconds to finish, this "?time" is included in "Delay" of this rule. In "?time" seconds later, assembly of $part_k$ on the "num1" site is then finished and robot "?rid" is ready for assembly again. The symbol "?" in the rule stands for the binding word that can be substituted by any word in the state. After the symbol is bound, it is substituted with a word in the rule. For examples, let the state of multi-robot assembly system be
(robot Adept ready) (site Adept 1 $part_i$ id_{i1})
(site Adept 2 $part_i$ id_{i2}) (hand Adept $part_j$ id_{j1})
(time Adept op-x 10)

After the precondition (robot ?rid ready) is checked to see whether it is satisfied or not, the symbol "?rid" is bound to Adept word and this word is put into the binding table. According to the binding table, the precondition (site ?rid ?num1 $part_i$?pid1) is then transferred to (site Adept ?num1 $part_i$?pid1). Next, symbol "?num1" is bound to 1 and "?pid1" is bound to id_{i1}, which are then put to the binding table. Alternatively, the symbol "?num1" is bound to 2 and the symbol "?pid1" is bound to id_{i2}, such that the binding table must be divided into two tables. At last, the symbol "?num2" is bound to 2 , the symbol "?pid2" is bound to id_{j1}, and the symbol "?time" is bound to 10. Those are then put into the binding table as shown shown in Fig 5.

According to the left sub-tree of the binding table, the rule is expressed
Precond:
(robot Adept ready) (site Adept 1 $part_i$ id_{i1})
(hand Adept $part_j$ id_{j1}) (time Adept op-x 10)
Delete:
(robot Adept ready) (site Adept 1 $part_i$ id_{i1})
(hand Adept $part_j$ id_{j1}) (hand Adept empty)
Action:
(Adept op-x 1 id_{i1} 3 id_{j1})
Delay:
(10)
Add:
(robot Adept ready) (site Adept 1 $part_k$ id_{i1})

According to the right sub-tree of the binding table, the rule is changed. By now, it is clear that any AND/OR graph can be systematically transformed to a set of assembly rules.

Obviously, the way we propose here allows each rule containing the binding words which can be mapped to the other words. This provides a easy modeling method for a multi-robot assembly system. Since definitions of those rules does not depend on the knowledge about the number of sites of a robot and the number of robots in the system. This is an advantage of applying this rule format for modeling such a dynamical robotic assembly system.

The assembly rules for a multi-robot assembly system not only contain those involving the assembly tasks directly from the AND/OR graph of a product assembly but also contain the domain knowledge about assembly. The following four rules for assembly belong to the latter type.

- Rule 1: put the grasped part down to a site of a robot.
  Precond:
  (robot ?rid ready) (hand ?rid ?type ?pid)
  (site ?rid ?num empty) (time ?rid place-site-from-hand ?time)
  Delete:
  (robot ?rid ready) (hand ?rid ?type ?pid)
  (site ?rid ?num empty)
  Action:
  (?rid place-site-from-hand ?num ?pid)
  Delay:
  (?time)
  Add:
  (robot ?rid ready) (site ?rid ?num ?type ?pid)
  The other there rules listed below have similar constituents and hence are omitted for limited space.
- Rule 2: Put the grasped part down to the buffer.
- Rule 3: Put a part sitting on a site of robot to a site of the buffer
- Rule 4 : Put a part sitting in a site of the buffer to a site of the robot.

The assembly rules suggested above belong to the domain knowledge about part's movement within a multi-robot assembly system. But there may be a repeated movement existing in this set of rules. For example, robot Adept perform the operation "place-site-from-buffer" and then perform the operation "place-buffer-from-site". This kind of problem can be solved by searching, which will not only find the best effective assembly tasks but also avoid this problem.

3.3 Inference Engine

The configuration of the inference engine and the searcher is shown in Figure 6. According to the assembly rules and the present state of the multi-robot assembly system, the inference engine generates all possible assembly tasks and their associated states. Then, searching is employed over those candidate states. Each state of the multi-robot assembly system is internally represented as a node. The information of a node includes the state of the multi-robot assembly system, the system time, and the delayed event. The state event list of a multi-robot assembly system contains state information of the system. The system time is taken as the time of the system. The delayed event list corresponds to the "add" part of a rule which must be delay for some time before being put into the system.

The scenario of an inference engine is the following: First, the inference engine generates a node of the present system

state. According to the assembly rules, the inference engine generates all possible states of the system, and the searcher place those states into the priority queue. Then, inference engine takes out a state from this queue and repeats the process again, and simultaneously the search will find a system state with the minimal cost. This will be kept on till the searcher find a goal or a sub-goal. For examples, let a state of a multi-robot assembly system be:
(robot adept ready) (site adept 1 $part_i$ id_{i1})
(hand adept empty) (site adept 2 $part_j$ id_{j1})
(robot arm ready) (site arm 1 $part_i$ id_{i2})
(hand arm empty) (site arm 2 $part_j$ id_{j2})
(buffer 1 $part_j$ id_{j3}) (buffer 2 empty)
(time adept-site-op-x 10) (time adept-buffer-op-x 18)
(timearm-site-op-x 15) (time arm-buffer-op-x 28)

The information of node 0 includes :
1. system time:0
2. state event list:
(robot adept ready) (site adept 1 $part_i$ id_{i1})
(hand adept empty) (site adept 2 $part_j$ id_{j1})
(robot arm ready) (site arm 1 $part_i$ id_{i2})
(hand arm empty) (site arm 2 $part_j$ id_{j2})
(buffer 1 $part_j$ id_{j3}) (buffer 2 empty)
(time adept-site-op-x 10) (time adept-buffer-op-x 18)
(time arm-site-op-x 15) (time arm-buffer-op-x 28)
3. delayed event list: NULL

There are four assembly tasks for node 0, which are adept-site-op-x, adept-buffer-op-x, arm-site-op-x, and arm-buffer-op-x. And, each assembly operation creates the next node. Node 1 is generated by the adept-site-op-x operation, whose information contains:
1. system time :0
2. state event list:
(hand adept empty)
(robot arm ready) (site arm 1 $part_i$ id_{i2})
(hand arm empty) (site arm 2 $part_j$ id_{j2})
(buffer 1 $part_j$ id_{j3}) (buffer 2 empty)
(time adept-site-op-x 10) (time adept-buffer-op-x 18)
(time arm-site-op-x 15) (time arm-buffer-op-x 28)
3. delayed event list: (10 (robot adept ready) (site adept 1 $part_k$ id_{i1}))

Generally, the delayed event list contains the information about which assembly tasks that have not yet been finished. When the state can not be inferred by any rule, the delayed event list is put into the state event list and the system is changed. If there are two or more delayed event lists, the delayed event list which is with the smallest delay time is put into the system state.

3.4 Search Based Scheduling Algorithm

The searching strategy is important for inference engine since it can affect the system performance considerably. This strategy is based on A* search algorithm in this system. To clearly formulate this search strategy, first, we explain the goal in the following.

Suppose that there are k types of assembly tasks, and each type has to be performed N_i times so that there are totally $N = (N_1 + N_2 + .. + N_k)$ products to be assembled. Let this be the goal of the system. But the inference engine can not provide inference up to the goal, simply because of the lack of the parts ready for assembly in the system. It is impractical that the inference engine should regard this condition as the goal, and therefore should select a good sub-goal as an alternative. The sub-goal of the system is defined as follows: The sub-goal is defined to be the state which takes the minimum amount of time to finish any type of product from the present state.

In order to analyze this problem, we first define the following functions:

- H(x) function: H(x) is an estimated minimal time from the present state x to any sub-goal state .
- G(x) function: G(x) is the assembly time from the initial state to the present state x, where the initial state is the state after finishing a sub-goal state.
- F(x) function (cost function): F(x) = G(x) + H(x).
- B(x) function (bonus function) : This function is used for evaluating the state. If the state is preferable, then the bonus function of the state will be higher. The bonus function of a state is the sum of all part bonus functions within the state, namely,
  B(x) = Σ_p BP(p) : p is a part on the system, where BP(p) is a part bonus function explained below
- BP(p) function : The part bonus function must satisfy the following conditions.
  If a $part_i$ and a $part_j$ are assembled to form a $part_k$, then
  BP($part_i$) < BP($part_k$)
  BP($part_j$) < BP($part_k$)
  BP($part_i$) + BP($part_j$) < BP($part_k$)
  In order to consider the positions of parts, the same part type may have different part bonus value. For example, $part_i1$ and $part_i2$ belong to the same part type, but $part_i1$ is on a site of the robot whereas $part_i2$ is on the buffer. Therefore, we should have
  BP($part_i2$) < BP($part_i1$)

The following is the search algorithm.
Searching Algorithm.

Step 1 Start with OPEN containing only the initial node (node 0) . Set CLOSED to the empty list. Set GOAL to the empty node.

Step 2 Until a OPEN is empty, repeat the following procedure:

Step 3 Pick the node x on OPEN with the lowest F(x) value (cost function). Call it BESTNODE. Remove it from OPEN. Place it on CLOSED. See if BESTNODE is a goal node, a sub-goal node or a node which can not be inferred. If so, record this node to GOAL. If GOAL node is not empty, then compare their F functions (cost functions). Set the node with the lowest F function value to GOAL. If their F function value are the same, then compare their B function (bonus function). Set the node B with the largest function value to GOAL. Otherwise, put this node back to the inference engine. And, the inference engine generates the SUCESSOR of the BESTNODE, wherefrom the searcher does the following:

1. Set SUCESSOR to point back to BESTNODE. Those backwards links will make it possible to recover the path once a solution is found.
2. The G(SUCCESSOR) is the system time of the node SUCESSOR.

3. See if SUCCESSOR is the same as any node on OPEN (i.e. it has already been generated but not processed). If so, call that node OLD. Since this node already exists in the graph, we can throw SUCCESSOR away and add OLD to the list of BESTNODE's successor. Now we must decide whether OLD's parent link should be reset to point to BESTNODE. It should be through if the path we have just found to SUCCESSOR is cheaper than the current best path to OLD (since SUCCESOR and OLD are really the same node).
4. If SUCCESSOR was not on OPEN, see if it is on CLOSED. If so, call the node on CLOSED OLD and add OLD to the list of BESTNODE's successors.
5. In order to warrant that searching time not exceed a limit time, the number of nodes which were put into OPEN must be limited. If the number of nodes which were put into OPEN does not exceed the limited number and SUCCESSOR was not already on either OPEN or CLOSED, then put it on OPEN and add it to the list of BESTNODE's successors and compute F(SUCESSOR)=G(SUCESSOR)+H(SUCESSOR).

Step 4 When OPEN is empty, and GOAL is not an empty node, backtrack GOAL to the initial node to get the command list of assembly tasks, and then generate the first command to the system. If GOAL is an empty node, then do not generate any command to the system.

4 CimStation Simulation

Modeling and programming of a multi-robot assembly system can be completely written on CimStation. Here, we build a multi-robot assembly cell by CimStation, and use it to simulate all kinds of activities in the multi-robot assembly cell, which consists of a robot ADEPT model, a robot A-ARM model (a new type of robot), a conveyor belt model, a part loader, and a common buffer model. Each model includes information about its geometric shape, kinematics description and path planning algorithms. CimStation provides a tool to create those models, or use SIL language to generate those models. The details is described in [20].

The scenario of the simulation of the multi-robot assembly system is: First, a part is automatically put onto the conveyor by the part loader unit. Next, the conveyor sensor unit would detect the coming of the part on the conveyor and notify the control unit about the coming part; the control unit then decide a robot which is ready for assembly to pick up the coming part; finally, the control unit call the scheduling unit to assign the assembly tasks to respective robots. According to the present state of the assembly system, the scheduling unit decides the best assembly tasks for respective robots, and then turn to the control unit to implement this assignment.

There are two robot units: ADEPT unit and A-ARM unit. Each robot unit supports the task-level robot assembly commands, which is assembling the base part on a site and another part on the hand of the assembling robot. In the robot unit, each task-level assembly command is transferred to a set of primitive robot commands. which include the following:

- Picking up a part from workcell.
- Placing a part in workcell.
- Moving from one position to another.

The primitive buffer commands include the following:

- Rotating the buffer clockwise.
- Rotating the buffer anti-clockwise.
- Stopping the buffer rotation.

Different task-level commands (assembly tasks) correspond to the different set of primitive commands. For example, assembling the base part1 on site 1 and the added part2 on site 2 is an assembly task-level command. Thus, the robot unit transfers that command to a set of primitive robot commands listed below:

Step 1 Moving the robot to site 2.

Step 2 Having the robot pick up the part2.

Step 3 Moving the robot move to site 1.

Step 4 Having the robot place part2.

Even for the same assembly tasks, the parameters of each primitive operation may not be the same, simply because the assembly site can be different. The configuration of the primitive robot operations like "pick up", "move", and "place on", which are quite fundamental to assembly system. In a multi-robot assembly system, motional collision problem will be inevitable if a sound motion planning is not enforced. So, the move module in the primitive operation unit carefully considers the problem of collision. The methodology of motion planning is adapted from [1].

5 Experiments

In this section, we will express how the scheduling method proposed in the previous section can be applied to such an flexible two-arm assembly cell and its simulation environment built on CimStation. Totally, there are four parts, denoted as part 1 - 4 respectively, to be assembled into a product in our experiment, whose assembly AND/OR graph is shown in the Figure 7. The objective of the simulation experiment is demonstrated the efficiency of the proposed assembly scheduleing strategy for assembling this kind of product.

First, the part-load unit randomly places different types of part on the conveyor, and the time at which a part is placed is called the arrival time of that part hereafter. The part arrival-time chart for each part type is shown in Figure 8 - 11. The robot utilization vs. time represents how often the robot is commanded (either for transferring or assemblying) in the simulation throughout the total period of assembly. The utilization of the robot Adept is shown in Figure 12, and that of the robot A-arm is shown in Figure 13. The number of the robot assembly sites used during the assembly period, indicates how many assembly tasks are active in that period. The number of the used sites of robot Adept is shown in Figure 14, and that of the robot A-arm is shown in Figure 15. Moreover, the number of sites on the buffer used expresses how many parts are stored in the buffer during the assembly period. The number of used sites on the buffer is shown in Figure 16. Finally, time chart for completing all product assemblies is shown in Figure 17.

6 Conclusion

As the fully automated assembly system in multi-robot environment played an important role, the scheduling problem become more and more important than before. But the complexity of this kind of system grows so rapidly that its modeling and scheduling become more and more difficult. Here, we proposed a new methodology for modeling this kind of assembly system. It is more intuitive that representation of this kind of assembly system should use the ruler-base knowledge generated from the AND/OR assembly graph and the domain knowledge. Based on these rules, the assembly scheduling deduces all of the possible assembly tasks via an inference engine and and then finds the optimal ones through applying a search algorithm.

In addition, integration of a scheduling method into a simulation environment is another novel achievement of this paper. Here, the experiment of a multi-robot assembly system was performed. In order to avoid collision in the multi-robot assembly system, a motion planning strategy is adopted in this simulation. It has been demonstrated that the assembly system runs quite efficiently.

References

[1] L.C. Fu and Y.J. Hsu, "Fully Automated Two-Robot Flexible Assembly Cell," Proc. IEEE Int. Conf. on Robotics and Automation, pp. 332-338, 1993.

[2] L.S. Homem de Mello and A.C. Sanderson, "A Correct and Complete Algorithm for the Generation of Mechanical Assembly Sequences," IEEE Trans. Robotics and Automation, Vol. 7, No. 2, pp. 228-240, 1991.

[3] L.S. Homem de Mello and A.C. Sanderson, " Representation of Mechanical Assembly Sequences," IEEE Trans. Robotics and Automation, Vol 7, No. 2, pp. 211-227, 1991.

[4] L.S. Homem de Mello and A.C. Sanderson, " AND/OR Graph Representation of Assembly Plans," IEEE Trans. Robotics and Automation, Vol. 6, No. 2, pp. 188-199, 1990.

[5] K.G. Shin and Q. Zheng, "Scheduling Job Operations in an Automatic Assembly Line," IEEE Trans. Robotics and Automation, Vol. 7, No. 3, pp. 333-341, 1991.

[6] C. Sriskandrajah, P. Ladet, and R. Germain,"Scheduling Methods for a Manufacturing System," Elsevier Science Publishers B.V. (North Holland), pp. 173-189, 1986.

[7] Y.F. Huang and C.S.G. Lee, " A Frame Work of Knowledge-based Assembly Planning," Univ. of Purdue, Dept. of Electrical Engineering, Proc. of Int. Conf. Robotics and Automation, pp. 599-604, 1992.

[8] T. Kanehara, T. Suzuki, A. Inaba, S. Okuma, "On Algebraic and Graph Structural Properties of Assembly Petri Net: Search By Linear Programming," Proc. Int. Conf. Robotics and Automation, pp. 2286-2293, 1993.

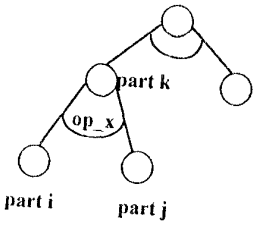

Figure 1: An AND/OR assembly sub-graph

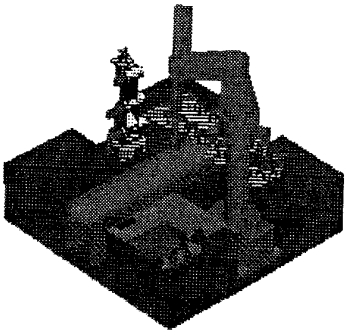

Figure 2: System picture

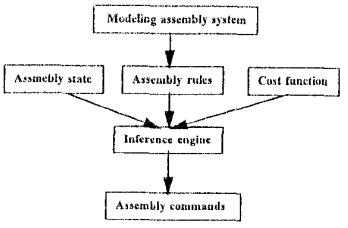

Figure 3: The hierarchical view of the assembly scheduling unit

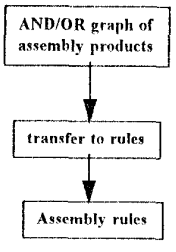

Figure 4: The procedure of modeling assembly system

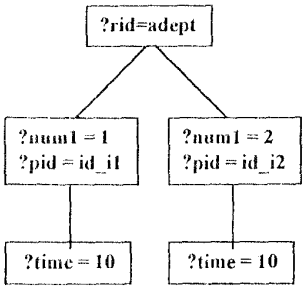

Figure 5: The rule binding table

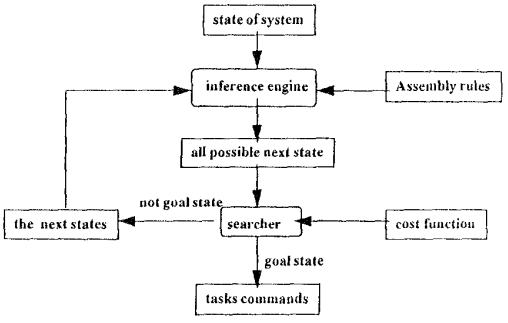

Figure 6: The configuration of inference engine and search

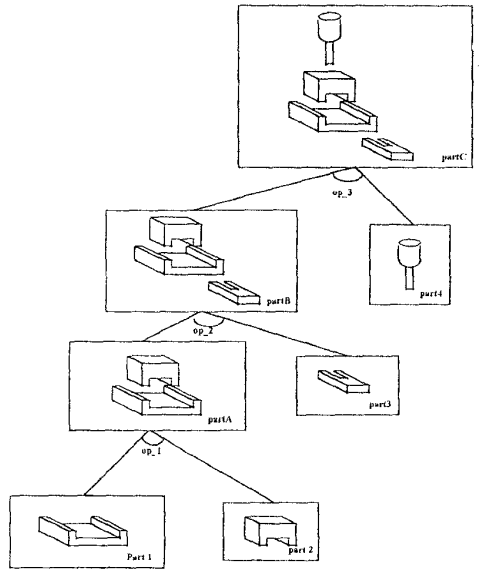

Figure 7: An AND/OR assembly graph of the product

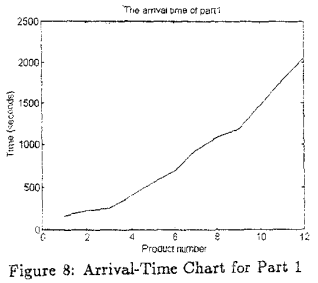

Figure 8: Arrival-Time Chart for Part 1

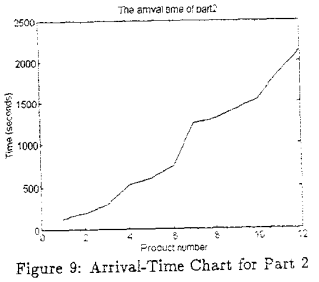

Figure 9: Arrival-Time Chart for Part 2

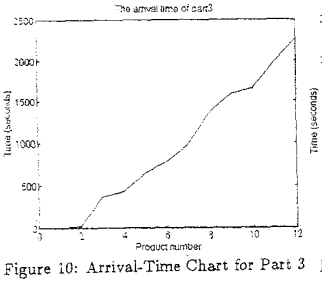

Figure 10: Arrival-Time Chart for Part 3

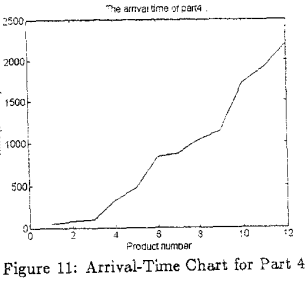

Figure 11: Arrival-Time Chart for Part 4

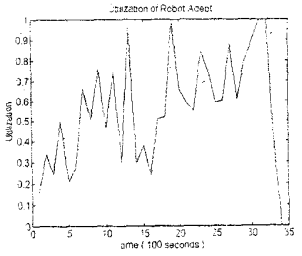

Figure 12: Utilization of the Robot Adept

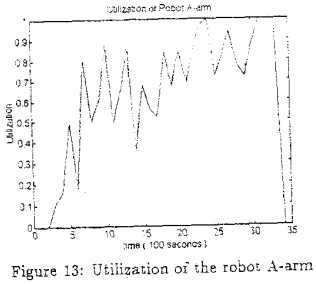

Figure 13: Utilization of the robot A-arm

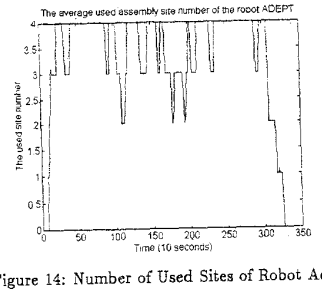

Figure 14: Number of Used Sites of Robot Adept

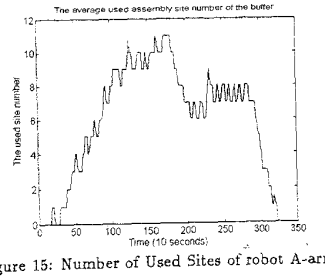

Figure 15: Number of Used Sites of robot A-arm

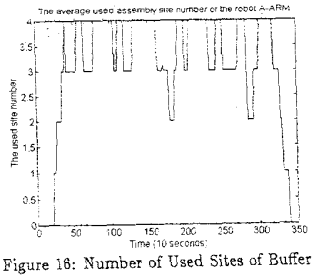

Figure 16: Number of Used Sites of Buffer

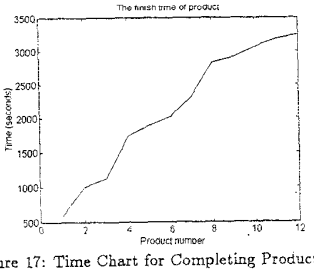

Figure 17: Time Chart for Completing Product Assemblies