Machine Assembly/Disassembly Planning by Cooperative Agents*

Satoshi IMAMURA**

A machine assembly/disassembly planning system based on agent cooperation has been developed. It can create an optimal assembly and/or disassembly sequence for any goal, such as disassembling/assembling all parts or replacing a broken-down part. The planning system is reactive, i.e., it can cope with unpredictable incidents, such as changing goals while the system is under operation. We define twenty six class agents for machine assembly/disassembly tasks and use an improved algorithm whose original one was proposed by Maes. The framework of the proposed planning system and detail definition of the planning algorithm are introduced and the results of experiments are discussed.

**Key Words**: Assembly, Disassembly, Process Planning, Production System, Multi Agent Architecture, Topological Relations of Assembly Parts

1. Introduction

Machine assembly planning systems play important roles of verifying assemblability, generation of appropriate machine assembly/disassembly sequences for manufacturing and maintenance. In this paper an assembly planning system based on cooperation of agents is reported.

Numerous studies with regard to assembly planning have been reported. Kitajima et al.(1) used directed graphs, Ito and Sobukawa(2), Uchiyama et al.(3), and De Mello and Sanderson(4) used assembly plan trees, Sekiguchi et al.(5) used information of part connective relations, and Seow and Devanathan(6) applied a temporal logic to disassembly planning. All of them generate plans based on the initially provided static assembly data, therefore they lack planning flexibility, such as setting a variety of goals or changing goals while the system is under operation.

The proposed system can create a wider variety of plans including both assembly and disassembly sequences, such as, replacing a broken-down mechanical part. Furthermore, the proposed planning algorithm creates step-by-step plans while observing current states and goals, therefore it can cope with unpredictable incidents, such as changing goals, while the system is under operation.

The original idea of the cooperative planning scheme came from Maes' "The Dynamics of Action Selection(7)." It proposed a society of agents that can create a sequence of robot hand motions by cooperation of the society members. It exhibited desirable properties of flexibility (adapting to new or unforeseen situations), robustness (graceful degradation of performance) and modularity (it is easy to introduce new agents or modify agents). However, this scheme has several impasses; (1)loops: the same sequence of agents is activated over and over again, (2)deadlocks: situations where no agent is active enough to escape from a local optimal, (3)instability: system parameters need delicate tuning which is dependent on situations and goals. To apply Maes' scheme to machine assembly planning, we improved the algorithm to evade the impasses and introduced some ideas, such as, (1)class agents: we introduced class agents that can generate necessary instances which depend on problems, (2)structured planning: agents
to assemble and disassemble units (subassembly) are introduced to achieve better planning, (3) bias activity: each agent gets some predefined bias activity that controls the order of preference of activation.

The proposed assembly planning scheme will be introduced, and the results of the experiment on a spindle head will be discussed in the following sections.

2. Framework of Cooperative Planning

The machine assembly/disassembly planning system consists of a cooperative planning shell and classes of agents for particular assembly/disassembly tasks, with an instance agent generator. The cooperative planning shell is developed to provide a general environment for agents to make cooperative decisions. Class agents are described by the format shown in Fig. 1. Examples of class agents are shown in Fig. 2.

The list of base conditions is used as the criteria to create agents from their class agents. The list of preconditions is used as the criteria to become executable. The list of facts after the prefix ‘negate’ is used to represent the effect of the agent action to the current state. In addition to these lists, agents have their activity level. The real value after ‘bias’ is used to jack up the agent’s activity level for the purpose of controlling the order of preference of its activation. The agents can refer to their global fact base and their action history. The global fact base consists of facts of goals and current status. The goals and current status may change unpredictably at any time.

The global fact base is represented by the following predicates:

- **fit** (Direction, Part1, Part2, FitType, Truth)
- **face** (Direction, Part1, Part2, FaceType, Truth)
- Direction = (x, y, z), Truth = (true, false)
- FitType = {press, position, movable, taper, screw}
- FaceType = {clamping, contact, gap, taper}

The fit/5 expresses that the “Part2” fits in the hole of the “Part1,” whereas the face/5 expresses that the left side surface of the “Part1” faces the right side surface of the “Part2.” The “Truth” is used to indicate if the predicates hold or not. The “Truth” of fit/5 and face/5 predicates that match the negate list are rewritten, from ‘t’ to ‘f’ or from ‘f’ to ‘t’, when the agents are activated.

Twenty six classes of agents, shown in Table 1, are defined for assembly/disassembly tasks. The agents in the left side column are for assembly tasks, whereas the agents in the right side column are for disassembly tasks. Agents (1)–(8) are to fasten or loosen the parts that have a screw type fit, agents (9)–(12) are to put or pull out unbolted parts. Agents (13)–(16) are to assemble/disassemble parts that have internal fit relations, whereas agents (17)–(23) are for parts that have only external relations when the agents are activated. Agents (24)–(26) are to put or remove keys that prevent rotation between internal and external fits. The pair of agents related by lines are for opposite tasks and each pair shares the same base-condition (instance generation condition) definition. The twelve pairs of agents that are for the same type but opposite direction tasks ((1)–(3), (5)–(7), (9)–(11), (13)–(15), (17)–(19), (21)–(23), (24)–(26), (2)–(4), (6)–(8), (10)–(12), (14)–(16), (18)–(20)) have mirror reflection image relations, i.e., the form of predicates appearing in the pair agents is the same except that the face (Dir, P1, P2) or face (Dir, P1, P2, Type, Truth) predicates in one
partner appear as the face (Dir, P2, P1) or face (Dir, P2, P1, Type, Truth) predicates in the other partner.

A particular set of instance agents is automatically created by the instance generator when new global fact data are loaded. Expressions of instance agents are the same as class agents with variables “Dir” and “Name” being instantiated. An example of a set of instance agents created is shown in Table 2 in section 4.

A notion of unit, a subassembly of parts, is introduced to set a constraint to assembly/disassembly sequences. It is expressed by a predicate unit (Unit, List), where the “List” is a list of parts (and units) that compose the “Unit”. If the “Unit” is defined, the system automatically rewrites the global fact base as:

\[
\begin{align*}
\text{fit}(\text{Dir}, \text{Part1}, \text{Part2}, \text{Type}, \text{Truth}) & \rightarrow \text{fit}(\text{Dir}, \text{Unit}, \text{Part2}, \text{Type}, \text{Truth}) \\
\text{fit}(\text{Dir}, \text{Part2}, \text{Part1}, \text{Type}, \text{Truth}) & \rightarrow \text{fit}(\text{Dir}, \text{Part2}, \text{Unit}, \text{Type}, \text{Truth}) \\
\text{face}(\text{Dir}, \text{Part1}, \text{Part2}, \text{Type}, \text{Truth}) & \rightarrow \text{face}(\text{Dir}, \text{Unit}, \text{Part2}, \text{Type}, \text{Truth}) \\
\text{face}(\text{Dir}, \text{Part2}, \text{Part1}, \text{Type}, \text{Truth}) & \rightarrow \text{face}(\text{Dir}, \text{Part2}, \text{Unit}, \text{Type}, \text{Truth}) \\
\text{where Part1}\in\text{List}, \text{Part2}\in\text{List}
\end{align*}
\]

Then, the disassembly of the “Unit” always precedes the disassembly of any part in the “List”, and the assembly of the “Unit” always succeeds the assembly of any part in the “List”. Therefore the unit/2 predicate can be regarded as a constraint to the sequence of agent activation.

Agent activity is computed by external inputs from the global fact base and activity spreading among agents. The external inputs are provided to the agents that realize one of the goals, and to the agents of which the preconditions partially match the current status. Each agent gives away a part of its own activity through spreading links, which extend from a successor agent to some of its predecessors, the agent that realizes one of the preconditions of the successor.

The existential conditions of spreading links are defined by a set of predicates:

\[
\begin{align*}
(1) & \quad \text{screen_a_bolt_to_right(Dir, Name)} \\
(2) & \quad \text{loosen_a_bolt_to_left(Dir, Name)} \\
(3) & \quad \text{screen_a_bolt_to_left(Dir, Name)} \\
(4) & \quad \text{loosen_a_bolt_to_right(Dir, Name)} \\
(5) & \quad \text{screen_a_nut_to_right(Dir, Name)} \\
(6) & \quad \text{loosen_a_nut_to_left(Dir, Name)} \\
(7) & \quad \text{screen_a_nut_to_left(Dir, Name)} \\
(8) & \quad \text{loosen_a_nut_to_right(Dir, Name)} \\
(9) & \quad \text{pull_unbotted_part_to_left(Dir, Name)} \\
(10) & \quad \text{pull_unbotted_part_to_right(Dir, Name)} \\
(11) & \quad \text{put_unbotted_part_to_right(Dir, Name)} \\
(12) & \quad \text{pull_out_unbotted_part_to_left(Dir, Name)} \\
(13) & \quad \text{push_in_inner_fit_to_right(Dir, Name)} \\
(14) & \quad \text{pull_out_inner_fit_to_left(Dir, Name)} \\
(15) & \quad \text{push_in_outer_fit_to_left(Dir, Name)} \\
(16) & \quad \text{pull_out_outer_fit_to_right(Dir, Name)} \\
(17) & \quad \text{push_in_outer_fit_to_right(Dir, Name)} \\
(18) & \quad \text{pull_out_outer_fit_to_left(Dir, Name)} \\
(19) & \quad \text{push_in_outer_fit_to_left(Dir, Name)} \\
(20) & \quad \text{pull_out_outer_fit_to_right(Dir, Name)} \\
(21) & \quad \text{push_in_outer_fit_to_either(Dir, Name)} \\
(22) & \quad \text{pull_out_outer_fit_to_either(Dir, Name)} \\
(23) & \quad \text{push_in_outer_fit_to_either2(Dir, Name)} \\
(24) & \quad \text{put_a_key_to_shaft(Dir, Name)} \\
(25) & \quad \text{take_a_key_from_shaft(Dir, Name)}
\end{align*}
\]

link(Term1, Term2). Spreading links are asserted between an agent that has a term matching Term 1 in its negate list and an agent that has a term matching Term 2 in its precondition list. This predicate is introduced to assert only necessary spreading links and avoid unnecessary spreading links that deteriorate the overall system performance. For example, spreading links between the pair agents in the same row of Table 1 are unnecessary, as they tend to have iterative actions between them.

Three global parameters are used to tune the external input and the activity spreading. (1) The percentage of the agents’ activity spreading towards their predecessors. (2) The percentage of the external input coming from the goals. (3) The total activation level. The following computation is executed for all the agents at every time step. (1) The external input to an agent is computed. (2) The spreading of activity is computed. (3) The overall activity is adjusted to a constant level. (4) The activity level of each agent is baked up by its own bias activity value. (5) The agent that satisfies the following two conditions becomes activated: (a) It is executable, (b) Its activity level is higher than that of any other executable agent. If there are two or more than two executable agents that have the same highest activity level, then no agent is activated and the computation is forwarded to the next time step. Steps (1) through (5) are repeated until all the goals are satisfied.

3. Algorithm of Activity Computation

This section describes the cooperative planning algorithm in detail. The following definitions are given:

- a set of agents \( i = 1 \cdots n \)
- a function \( S(j, t) \), which returns 1 if the fact \( j \) is observed to be true (a member of the current state) at time \( t \), and 0 otherwise.
- a function \( G(j, t) \), which returns 1 if the fact \( j \) is a member of the goals at time \( t \), and 0 otherwise.
- a function \( B(s, i, j) \), which returns 1 if (1) the fact \( j \) is a member of the preconditions of the agent \( s \), (2) the fact \( j \) matches one of the negate list of the

*JSME International Journal*
agent $i$ by changing the "Truth" to an opposite value, and (3) the fact $j$ is observed to be true. It returns 0 otherwise.

A function $\text{bias}(i)$, which returns the bias level of the agent $i$ defined after the prefix 'bias', $0 \leq \text{bias}(i) \leq 1$.

\[ \pi : \text{the total level of activation (real number)}, \quad 0 \leq \pi \leq 1. \]

\[ \beta : \text{the percentage of backward spreading}, \quad 0 \leq \beta \leq 1. \]

\[ \gamma : \text{the percentage of the external input from the goals, versus the current state}, \quad 0 \leq \gamma \leq 1. \]

The external input of activity to agent $i$ at time $t$ is:

\[ \text{external}(i, t) = \frac{d}{dt} \sum_{j \in P_i} a(j) \cdot S(j, t) \]

\[ + \frac{1}{g_i} \sum_{k \in g_i} b(i) \cdot G(k, t) \quad (1) \]

where $j$ ranges over the predicates $1 \ldots P_i$ in the precondition of agent $i$ and $k$ ranges over the predicates $1 \ldots g_i$ in the negate list of agent $i$. The $a(t)$ and $b(t)$ are coefficients that are chosen to satisfy the following:

\[ a(t) \sum_j S(j, t) = \pi (1 - \gamma) \quad (2) \]

\[ b(t) \sum_k G(k, t) = \pi \gamma \quad (3) \]

where $i$ ranges over the set of agents $1 \ldots n$.

The backward spreading from agent $s$ to agent $i$ is specified by the following equation, where $s$ is the successor of $i$.

\[ b(s, i, t) = c(t) \sum_{j \in s_i} B(s, i, s_i) \quad (4) \]

where $s_i$ ranges over the preconditions of agent $s$. The $c(t)$ is chosen such that:

\[ c(t) \sum_{j \in s_i} B = a(s, t) \cdot \beta \quad (5) \]

where $p$ ranges over the predecessor of agent $s$ and $a(i, t)$ stands for the activity level of agent $i$ at time $t$.

The following equations specify what an agent $i$ retains and what it receives from other agents, where $s$ ranges over the successors of agent $i$:

\[ \text{receives}(i, t) = \sum_s b(s, i, t) \quad (6) \]

The activity level of agent $i$ at time $t$ is defined by the following equations:

\[ a(i, t_0) = 0 \]

\[ a(i, t_1) = \text{external}(i, 1) \cdot (1 - \beta) + \text{receives}(i, t) + \text{bias}(i) \]

\[ a(i, t) = ( \text{external}(i, t) + a(i, t-1) - \text{bias}(i)) \cdot (1 - \beta) + \text{receives}(i, t)) / 2 + \text{bias}(i) \quad (7) \]

4. Experiments of Machine Assembly Planning

As an example we have experimented the proposed scheme on a few machines. Figure 3 shows a spindle head, used as one of the examples for the experiments. It consists of 31 parts, whose assembly structure is expressed by 48 fit/5, 42 face/5 predicates and a unit (unit1, [2, 3, 4, 5, 14, 15, 16, 17, 18, 19, 25, 26]) predicate. Table 2 shows 63 instance agents created from class agents (Table 1) for the assembly and disassembly tasks for the spindle head. If there were no restriction of the instance creation, the number of created instance agents for this example could be the number of parts(31) x the number of class agents = 806. As a result of adequate definition of base condition, the number of created agents is reduced to 63.

The system could generate adequate sequences for goals of assembly/disassembly of whole parts and goals of replace/replace of part 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 17, 20, 21, 22, 23, 24, 27, 28, 29, 30 or 31. However removal of part 16, 19, 25, 16 was done after removal of the unit1 which is defined by the unit/2 predicate, while the removal of those parts can be done without removing the unit1. We need to change the contents of the unit/2 in case it prevents generation of an adequate sequence of assembly or disassembly.

To evaluate the stability of the system, we varied $\beta$ (the percentage of backward spreading) and $\gamma$ (the
percentage of the external input from the goals\(^1\) in the range \(0.0 - 1.0\) by 0.1 steps. According to the evaluation, the system was sufficiently stable in the area of \(0.5 < \beta < 1.0, \ 0.0 < \gamma < 1.0\).

Figure 4 shows the assembly/disassembly simulator. It can show assembly or disassembly of parts by an animation. The goal is given by inputting a target part number (or 'all' in case the target consists of whole parts) and clicking the 'Disassembly' or 'Assembly' button. The experiment was performed using Sicstus Prolog\(^{20}\) on a Sparc 10 workstation. About 1 min was required for creating instance agents and setting spreading links and an average of about 30 sec was required for assembly/disassembly planning for various goals.

5. Conclusion

A machine assembly/disassembly planning system based on cooperation of agents has been developed. The computer experiments performed on a few machines showed that the system could cope with unpredictable incidents such as changing goals while the system is under operation, and it could create a variety of optimal plans including assembly or disassembly of whole or part of the machines, although in some exceptional cases inadequate plans were generated because of inadequate unit definitions for goals.

We need further experiments and development to enforce the proposed planning scheme. Also we plan to develop an interface between a CAD system and the proposed planning system to directly obtain assembly structure data (fit/5, face/5) from CAD drawing data and show planning results by an animation of the drawing. The application area of the system could be the instruction of machine disassembly sequence for maintenance, checking of assembling, and calculation of assembly or disassembly cost.

The proposed planning scheme is also useful for other applications such as robot hand motion planning\(^{20}\).

Appendix: Supporting Predicates Used for Agent Definitions

1) positioned\((P)\)
   It states that the part \(P\) is assembled to other parts, i.e., there are fit/5 and face/5 predicates that contain the \(P\) and whose Truth is true.

2) assembled\((P)\)
   It states that the part \(P\) is either a single part or a unit whose components are all assembled.

3) dismantled-unit-member\((P)\)
   It states that the part \(P\) is either a single part or a member of a unit that is dismounted from the whole assembly.

JSME International Journal
4) bolt\((P)\)
   It states that the part \(P\) is a bolt, i.e., there is one
   fit\((\_\_\, P, sc\_\_)\) and one or more than one fit\((\_\, P, mo\_\_)\) and no fit\((\_\, P, s\_\_\_)\).
5) nut\((P)\)
   It states that the part \(P\) is a nut, i.e., there is only
   one fit\((\_\, P, sc\_\_)\) and no other fit/5 that contains the
   \(P\).
6) fit\((\text{Dir, P1, P2})\)
   It states that the part P2 fits into the hole of the
   part P1 directly or indirectly.
7) face\((\text{Dir, P1, P2})\)
   It states that the left side surface of the part P1
   faces the right surface of the part P2 directly or
   indirectly.

References

(1) Kitajima, K., Nishiyama, T. and Yoshikawa, H.,
Development of an Interactive Machine Assembl-
ability Check System based on Oriented Con-
nectivity Graph on Machine, Bull. of JSPE, (in
(2) Ito, M. and Sobukawa, H., Automatic Generation
of Assembly Plans of Machines(2nd Report),
(3) Uchiyama, N., Arai, E. and Igoshi, M., Path
Generation for Disassembly in order to Verify the
Assemblability of Mechanical Products, Trans.
p. 1300-1307.
(4) De Mello, L. and Sanderson, A., And/Or Graph
Robotics Autom., Vol. 6, No. 2 (1990), p. 188-199.
(5) Sekiguchi, H., Imamura, S., Kojima, T. and Inoue,
K., Method of Developing Part Specification from
Assembly Drawing of Machine Unit(2nd Report),
(6) Seow, K.T. and Devanathan, R., Temporal Logic
Programming for Assembly Sequence Planning,
(7) Maes, P., The Dynamics of Action Selection,
(8) Imamura, S., Sakakibara, H. and Horie, Y,
Agents' Cooperative Actions Generating Optimal
Robot Hand Motion Sequences, Journal of
Mechanical Engineering Laboratory, (in
(9) Andersson, J., Andersson, S., Boottz, K., Carlsson,
M., Nilsson, H., Sjöland, T. and Widén, J., SICStus
Prolog User's Manual, SICS Technical Report,
T93 : 01,(1993), Swedish Institute of Computer
Science