7

Empirical Assembly Sequence Planning: A Multistrategy Constructive Learning Approach

Heedong Ko

ABSTRACT

Assembly sequence planning is a rich domain to explore learning and planning issues with a strong prospect for real world application in manufacturing industry. The problem is to find a sequence of assembly steps so that a goal assembly structure is built by carrying out the sequence. Unfortunately, a weak method that systematically explores all the alternatives is not practical because the number of possible sequences grows exponentially with the number of components, and there are few constraints or heuristics to tame the search process. Furthermore, the final assembly sequence is generated and selected in the context of a manufacturing environment that differs from one industry to another and from one job shop to another. As a result, the assembly sequence planning problem is solved by an experienced manufacturing engineer who is familiar with the production environment. Hence, the key to building an assembly sequence planner is assimilating the assembly episodes in memory which reflect the production environment. Here, the final assembly sequence is derived by bringing the previously experienced assembly episodes in memory by a multistrategy constructive learning approach in order to realize the planning methodology.

7.1 INTRODUCTION

It is a common practice in manufacturing industry to create and maintain a product model in a computer using a Computer-Aided Design (CAD) system. The product model is shared by many activities before producing the final product. For example, it is a common practice that a designer creates a model of a mechanical component using

Machine Learning and Data Mining: Methods and Applications
Edited by R.S. Michalski, I Bratko and M. Kubat

a CAD system, and the commands for a Numerical Controlled (NC) machine are directly generated to manufacture the component in the NC machine by a Computer-Aided Manufacturing (CAM) system. The CAD and CAM systems are linked by a Computer-Aided Process Planning (CAPP) system which derives the cut regions of a blank stock material, their sequences, cutter locations and machining conditions, as well as fixture set ups. Commercial CAPP systems for such tasks have recently been introduced.

By analogy with NC machining of a component, a CAPP system for assemblies would benefit the manufacturing process even more, since most products are assemblies. With the CAPP system for assemblies, the commands for an industrial robot to assemble the product are generated as a designer creates a model of an assembly using a CAD system. As industrial robots come into wider use in assembly applications, there is a growing interest in systems that can automatically generate a robot program for an assembly task. The development of robot programming languages has moved consistently toward higher levels of abstraction and greater use of world knowledge. From joint level, manipulator level, to task level, robot programming languages have been proposed. To realize a CAPP system for assemblies, it is necessary to solve a still higher level of planning problem, an assembly sequencing problem (De Fazio and Whitney, 1988).

The assembly sequencing problem is how to generate a sequence of mating operations whose initial subsequence does not construct a subassembly structure that prevents a collision-free path for a mating component belonging to the latter subsequence. To create such a sequence, the planner must follow precedence constraints when generating an assembly sequence. When one cannot assemble a component because of some blocking components already in place (a base subassembly), the planner must follow a precedence constraint that some of the blocking components in the base subassembly must be assembled after the mating component. Detecting such precedence constraints requires checking whether there is no collision-free path for the component to maneuver among the components of the base subassembly. This maneuvering problem is called a FIND-PATH problem, with the base subassembly as obstacles. The algorithm for solving the FIND-PATH problem in 3D with all six degrees of freedom is not polynomially bound (Donald, 1984). Because the moving component can be an assembly structure like the base subassembly, the FIND-PATH problem must be solved for every pair of the power sets of the components in the assembly (Ko, 1989), an exponential number of FIND-PATH problems to solve.

A naive state space search approach is of little use to solve the doubly exponential assembly sequencing problem. We cope with the complexity by grouping the base and the moving subassemblies into an assembly hierarchy. Hierarchical structuring of an assembly sequencing problem is powerful world knowledge to make the sequencing problem more tractable. First of all, components within a subassembly are assembled before the components of its parent subassembly. Furthermore, sibling subassemblies are assembled independently: that is, the components across neighboring sibling subassemblies do not interfere during their respective assembly operations within each subassembly. In short, an assembly hierarchy imposes additional precedence constraints as well as locality of interactions to reduce the number of FIND-PATH problems to be solved.

In some manufacturing industries, there are long established traditions. In the automobile industry, an engine subassembly is separately designed and manufactured

from the transmission subassembly. They are even manufactured in different factories. The separation may be due to the distinct roles played by the engine and transmission in the operation of the automobile. Their interactions are mediated by a cam shaft. The engine, transmission, and the cam shaft form sibling subassemblies in the final assembly hierarchy. Then, one might conclude that a functional decomposition of a product corresponds to its assembly hierarchy.

Unfortunately, this is not the case: some interactions are intentionally allowed to optimize the design. Some components are designed to play multiple roles to reduce the component count while maintaining the same functionality (Ulrich and Seering, 1988). These shared components cause interactions between their respective subassemblies. Furthermore, unexpected subassembly interactions may arise depending on the use of the hierarchy. For example, when a car was manufactured, no significant problem was encountered but the car may be hard to maintain, e.g., changing an oil filter may require the mechanic to dismount the engine subassembly to access the oil filter.

When creating a new assembly hierarchy, there is no stable heuristic or world knowledge to avoid all these interactions beforehand. Instead, through a long history of production experience in the automotive, aerospace, and ship building industries, there are widely accepted hierarchical structures, reflecting the collective knowledge of the respective industrial practices. Likewise, the world knowledge for constructing an effective assembly hierarchy brings previous assembly planning episodes to bear in a new assembly sequencing problem because they have already been subject to field tests and these design bugs have been filtered out. We hypothesize that the designer's experience and industrial practices are fundamental sources of knowledge for a planner to make the assembly sequencing problem tractable.

Hence, the key to building an effective planner is a learning issue: how to assimilate previously experienced assembly episodes, and how to bring them to bear in solving a new assembly problem. The former is a learning issue while the latter is a planning issue, and their coupling is a fundamental leverage towards solving the assembly sequence planning problem. We called such a planning paradigm an empirical assembly planning methodology to emphasize the heavy dependence on the learning mechanism to construct a plan.

Here, the learning situation is a multi-concept learning where the memory contains both assembly structure and sequencing concepts. Furthermore, success or failure of applying each assembly episode is recorded into memory incrementally, a closed-loop learning. In this chapter, we explain the Multistrategy Constructive Learning (MCL) (Michalski, 1993) system to address the learning situation using an empirical assembly planning system called NOMAD (NOrmative Mechanical Assembling Device). Before introducing the learning element, we introduce how NOMAD is given an assembly problem to solve, how to represent the assembly planning experience in memory, and the planning process.

7.2 REPRESENTATION AND PLANNING IN NOMAD

An assembly structure in CAD systems consists of components and mating conditions. A solid model of a component in a CAD system consists of faces at their boundaries between the inside and outside of a component. Then, a face is bounded by edges, and an edge is bounded by a pair of vertices. In addition, these bounding entities are

associated with geometric entities, e.g. a planar face is associated with a plane. From the component models, an assembly is represented by assigning mating conditions between a pair of components at their respective bounding entities: an "against" mating condition specifies that a planar face of a component is in contact with a planar face of another component (their respective faces are facing each other and are coplanar); an "aligned" mating condition specifies that a cylindrical rod of a component is "co-cylindrical" with a cylindrical hole of the mating component. An assembly structure is represented by a labeled graph, called a mating graph (Ko and Lee, 1987), where nodes represent components and links represent mating conditions.

Figure 7.1 illustrates a Bell Head assembly structure with three components, the Bell Head, the Pin, and the Ring, while the mating graph is shown in Figure 7.2 (a). In this chapter, the examples are illustrated in 2D where the bounding elements of the components are lines and points, although NOMAD has a 3D solid modeling kernel for the assembly modeling. The Global Frame is a reference coordinate system of all the components in the environment: the z axis is pointing vertically upward. Each component has its own local reference frame, called a base frame. Figure 7.1 shows the base frame of the Pin.

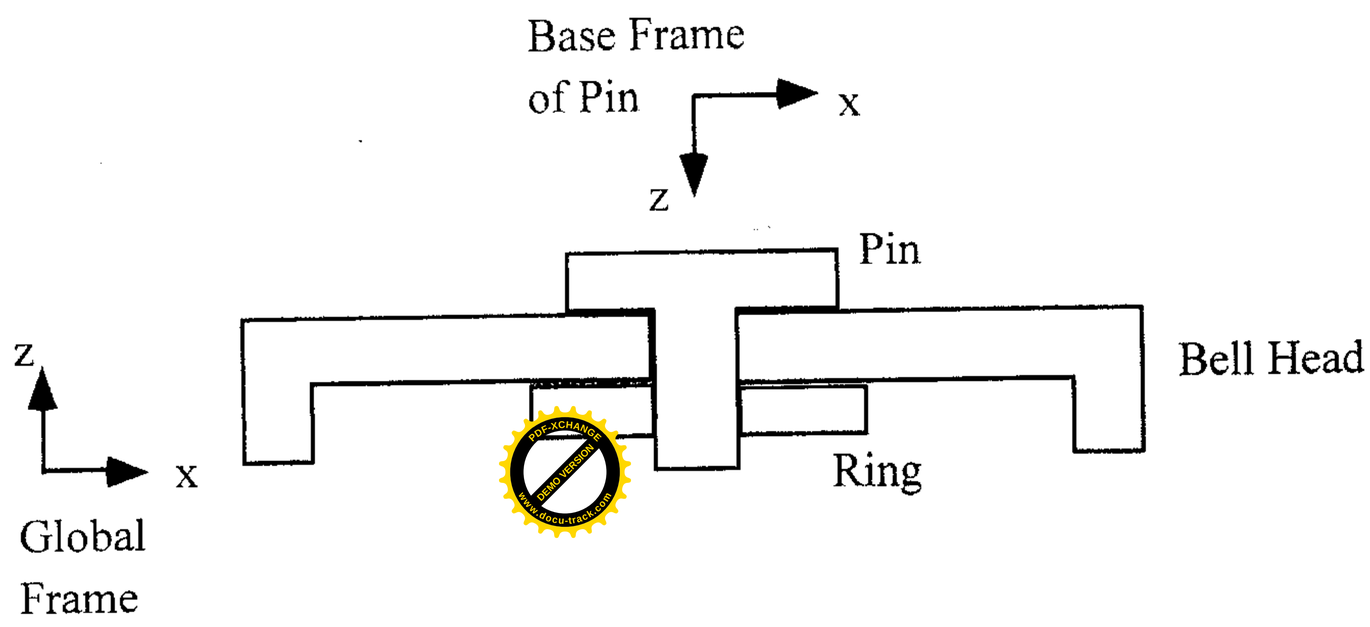


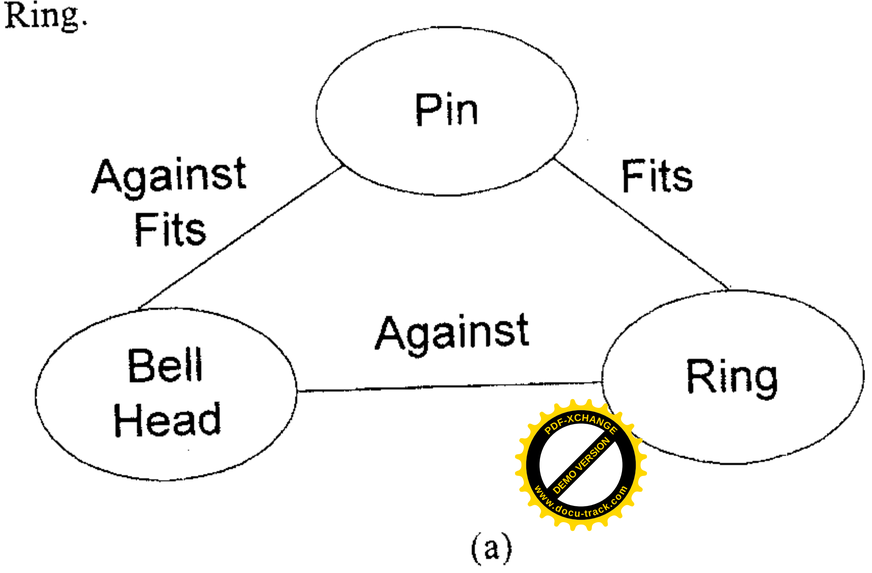
Figure 7.1 Bell Head assembly.

The assembly structure is represented by a mating graph, shown in Figure 7.2 (a). To assemble the Bell Head, the mating graph is used as a goal specification by the planner to determine the sequence of assembly steps. In general, each assembly step may consist of grasping, positioning, mating and various sensor commands to a robot. This makes it difficult to compare among the plan structures. Here, each assembly step may satisfy one or more mating conditions in the mating graph, assuming that each mating operation includes positioning, grasping and other robotic operations as its sub-operations during the operationalization process (Mostow, 1983).

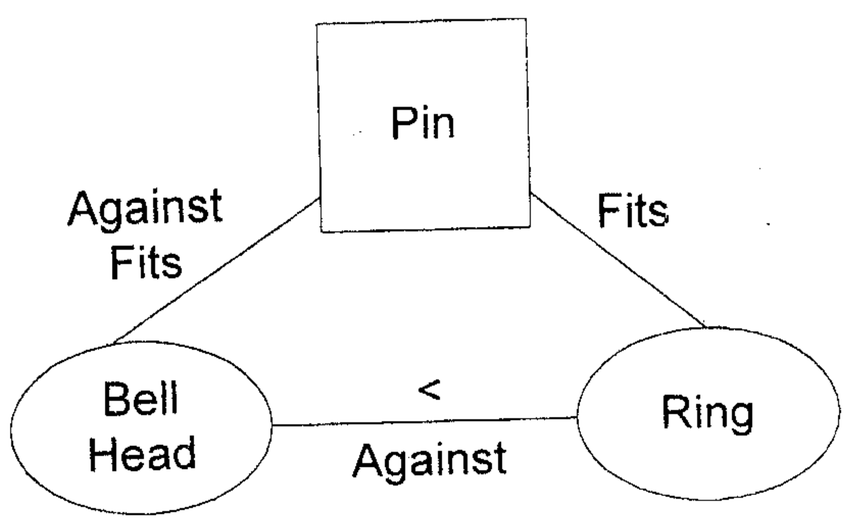
In addition to the simplification of each assembly step, the plan structure must be canonicalized into elementary chunks in memory to store and compare multiple plan structures, an essential step of an empirical learning element. These elementary chunks are called *assembly episodes*. An assembly episode is a plan segment consisting of a sequence and an assembly structure. Corresponding to the assembly structure, we

define a grouping as a canonical chunk from which the memory structure is built. The grouping is a set of components with a single distinguished component, called a base component, and others as satellite components. When assembling the grouping, the base component stays fixed while the satellite components are moved to mate with the base component.

In the context of a grouping, it is sufficient to represent an assembly sequence as a sequence of satellite components. As a result, an assembly episode is a combination of a grouping and a sequence of satellite components. Figure 7.2 (b) shows an assembly episode after planning for the mating graph in Figure 7.2 (a), where the Pin is the base component (a square node). The Bell Head and the Ring as satellite components are assembled to the Pin in sequence, which is represented by "<" notation between the Bell Head and the Ring.



(a)



(b)

Figure 7.2 Mating graph with two assembly episodes.

The final assembly plan may consist of one or more assembly episodes. The results of carrying out the final plan are categorized as either positive or negative applications of the assembly episodes in memory. These exemplars are input to a learning element that generates a new schema using an instance-to-class generalization method (Michalski, Ko and Chen, 1987). A schema is a generalized description of a set of domain objects, e.g. in the assembly domain, a component schema for a set of components and a grouping schema for a set of groupings. The grouping schema consists of a base component schema and satellite component schemas. The ordering

relations among the satellite component schemas are described by one or more precedence constraints. They are stored in a precedence schema using a part-to-whole generalization method (Michalski, Ko and Chen, 1987). Because there can be multiple ways to assemble the satellite components of a grouping, a grouping schema may have multiple precedence schemas. In short, the memory structure of NOMAD consists of two types of interrelated domain concepts: precedence and grouping schemas.

With these domain concepts in memory, the planning process proceeds in three phases: recollection, combination, and *post mortem* analyses. The recollection process decomposes the mating graph into assembly episodes. An assembly episode is identified in two steps: first by a grouping schema where the base and the satellite component schemas are identified with the components in the mating graph; second by a precedence schema where the assembly steps between the base and satellite components identified by the grouping schema in the first step are ordered into a sequence. These sequence segments are posted as candidate sequence segments to be considered as a part of the final assembly sequence.

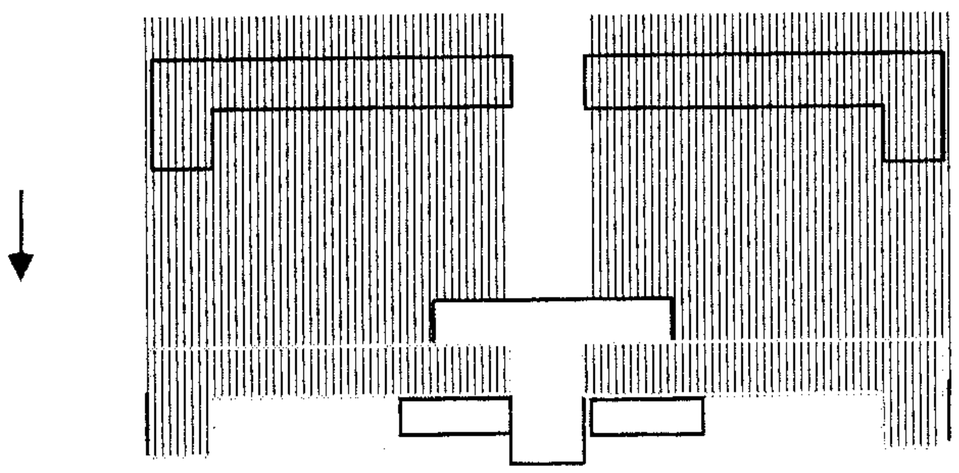
After the recollection step, the planner combines the candidate assembly episodes into a hierarchical structure that is consistent and complete. An assembly hierarchy is complete if it contains assembly steps involving all the components in the mating graph and consistent if all the precedence relations imposed by the candidate sequence segments are not contradictory with each other. An assembly hierarchy represents an abstraction hierarchy for the assembly planner in deriving an assembly sequence. That means, when deriving an assembly sequence for the root of a hierarchy, only its immediate children subassemblies are considered for ordering with an assumption that an assembly step of a subassembly may not interact with the assembly steps of a sibling subassembly (subassembly independence law, SIL). Furthermore, assembly steps within a subassembly must be assembled before the assembly steps of the parent subassembly (subassembly precedence law, SPL). The constraints imposed by SPL and SIL, as well as assembly episode[s] in[vo]ked from memory, are used by the credit assignment process during the *post mortem* analysis.

After an assembly hierarchy is constructed, the final assembly sequence is derived from the precedence relations of the sequence segments as well as SPL and SIL constraints from the hierarchy. An assembly step in the sequence may fail if there is no collision-free trajectory in reaching the desired configuration of the step. For example, there is no collision-free path for the Bell Head in reaching the mating configuration after the Ring is already assembled because of spatial interference, as shown in Figures 7.3 (a) and (b) with alternative approach directions. Hence, the application of this assembly episode is a negative experience, and then the planner succeeds with the alternative episode of Figure 7.2 (b).

To learn from the planning experience involving multiple concepts in memory, the learner conducts *post mortem* analysis of which concept in memory is responsible for the failure or success in applying the assembly episodes. Then, the plan structure must be a dependency structure (de Kleer, 1986), where each precedence relation in the assertional database is supported by an assembly episode, SPL, SIL constraints or their combination.

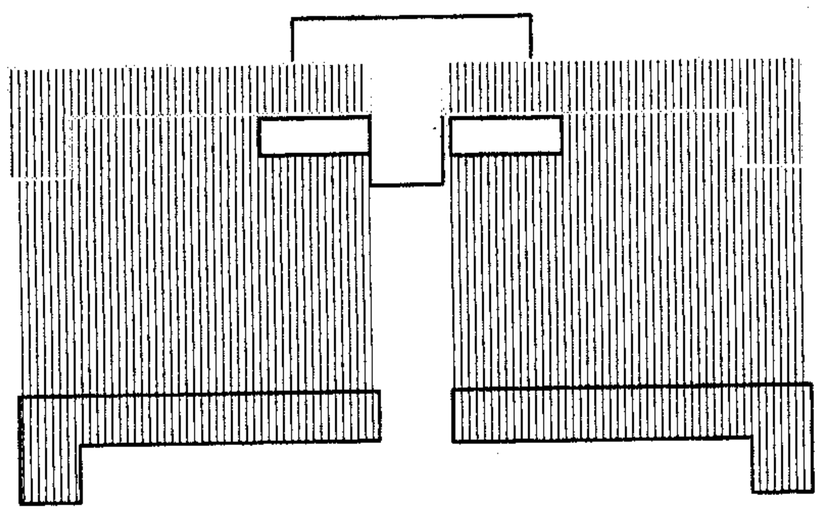
Because both precedence and grouping schema's are used to create the assembly episode, the *post mortem* analysis must identify which of the two types of concepts are responsible. In the previous example, a precedence schema is identified as the responsible concept when the failure or success is localized within the assembly episode. On the other hand, when a subassembly of the hierarchy violates SPL or SIL

constraints, or both, the grouping used to construct the subassembly is identified as the culprit. The credit assignment process identifies which concept in memory is responsible for the outcome, and hence is to be learned. The credit assignment problem is made possible by the dependency support structure between the assembly precedence relation of the final assembly sequence and the assumptions introduced by instantiating the schema in memory. The next section introduces multistrategy constructive learning as an extension of previous learning concepts to cope with the multi-concept learning situation of NOMAD.



(a)

Interference between Bell Head and P



(b)

Interference between Bell Head and Ring

Figure 7.3 Spatial interference.

7.3 MULTISTRATEGY CONSTRUCTIVE LEARNING

Learning from planning experience is multi-concept learning: the planning experience may highlight more than one concept to learn. For assembly sequence planning, there are two types of concepts to learn, grouping and precedence schemas. Then, the learner must determine which concept to learn, as well as the corresponding training instances. Previous learning systems have mostly been single-concept incremental learning systems. For example, it is reported that INDUCE (Larson, 1977) is given multiple

exemplars of west-bound or east-bound trains as two classes of a single concept (directionality of the train heading). In this case, indexing the memory for which concept to learn is trivial, e.g. this is determined by the class of the exemplar in INDUCE.

In a multi-concept learning situation, the system should determine which concept or concepts in memory to learn. The *post mortem* analysis using the dependency support structure of the assembly sequence in the previous section, identifies which concept (grouping or precedence schema) is responsible for the failure or success and hence which to learn. Here, the planning experience is decomposed into assembly episodes as applications of grouping and precedence schemas in memory, and their application results as training instances for the corresponding schemas. In the previous example, the precedence schema used to construct the assembly episode in Figure 7.3 is identified as a concept to learn, and the precedence constraints imposed by the assembly episode as training instances to incrementally refine the concept description of the chosen schema. Such an incremental learning in a multi-concept learning situation is called *closed-loop* learning, where the concept to learn is determined before, and incrementally updating the concept.

Initially, the concept description with little background knowledge (schemas) in memory is generated by inductive generalization of experimental observations. In empirical inductive learning, the learner generalizes examples observed to form their consistent and complete description. In machine learning, programs constructing empirical generalizations typically uses only descriptive concepts that are selected from among those used in describing the original observations. Such surface induction is called *selective induction*. In constructive induction (Michalski, 1983), the learner uses domain dependent as well as domain independent background knowledge to elaborate the input observations with new descriptors in order to search for inductive hypotheses in the preferred description space of the domain concepts. Here, the learner elaborates the training instances with relative spatial relations such as "higher-than" or "lower-than" among the components.

Finally, the system should evaluate the constructed knowledge in order to decide if it is to be stored in memory or not. Using the dependency structure, the learning episodes are combined into "more" useful knowledge before storing the experience. This step is similar to a constructive induction step, but it is called *deductive restructuring* to distinguish the fact that this step is to generate new "interesting" instances by combining multiple training instances rather than inserting new descriptors to each training instances. To learn from planning experience, these three learning steps must be combined. We call the integrated learning mechanism Multistrategy Constructive Learning (MCL):

MCL = Constructive Induction + Closed-loop + Deductive Restructuring

The next section illustrates the concepts introduced in this section with a learning scenario.

7.4 LEARNING SCENARIO BY NOMAD

Here, the multistrategy constructive learning approach of NOMAD is described as follows:

1. generate candidate grouping and precedence schemas inductively;
2. maintain the credibility of each candidate schema's by their applications; and
3. apply promising schema(s) in new assembly situations.

Suppose a user specifies the Bell Head assembly structure of Figure 7.1 when NOMAD has no experience. The Global Frame is a reference coordinate system of all the components in the environment: the z axis is pointing vertically upward. Initially, NOMAD has no schemas for the Bell Head assembly when given it as a task. So, the user may specify the grouping in Figure 7.2 (a): the Pin as a base and the Ring and the Bell Head as satellites. Then, the system may experiment with alternative plans randomly because it does not have any experience to guide the planning process.

From the successful planning episode of Figure 7.2 (b), using a constructive induction rule, the following descriptors are generated from the locations of individual satellite components with respect to the two reference frames, the global frame (dloc-g) and the base frame of the base component, the Pin (dloc-b):

- the location of the Bell Head is higher than the Ring with respect to the Global Frame: [dloc-g (Ring, Bell Head) < 0]
- the location of the Ring is higher than the Bell Head with respect to the base frame of the base component, Pin: [dloc-b(Ring, Bell Head) > 0)]

In addition, the base component, Pin, is T-shaped and the satellite components, Bell Head and Ring, are ring-shaped.

Then, NOMAD generates alternative schemas from the planning episode, using the "dropping a conjunct" inductive generalization rule (Michalski, 1983). Among alternative generalizations, two candidate precedence schemas are presented here, P1 and P2, with respect to the grouping schema as shown in Figure 7.4 and 7.5, respectively. In Figure 7.4, the base component schema, $p1, is aligned with satellite component schemas, $p2, and $p3, with respect to the global reference frames (dloc-g). Like the assembly episode, the square denotes the base component schema.

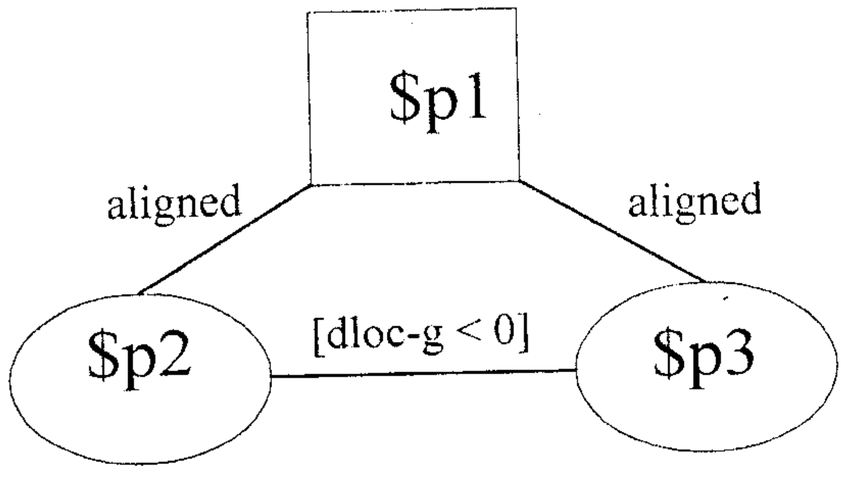


Figure 7.4 Candidate precedence schema P1.

Figure 7.5 shows that the base component schema, $p1, is aligned with satellite schemas, $p2 and $p3, and $p2 is assembled before $p3 because $p2 is lower than $p3

with respect to $p1. In addition, the component schemas of Figure 7.4 and 7.5 store the component shapes, $p1 is pin-shaped and $p2 and $p3 are ring-shaped.

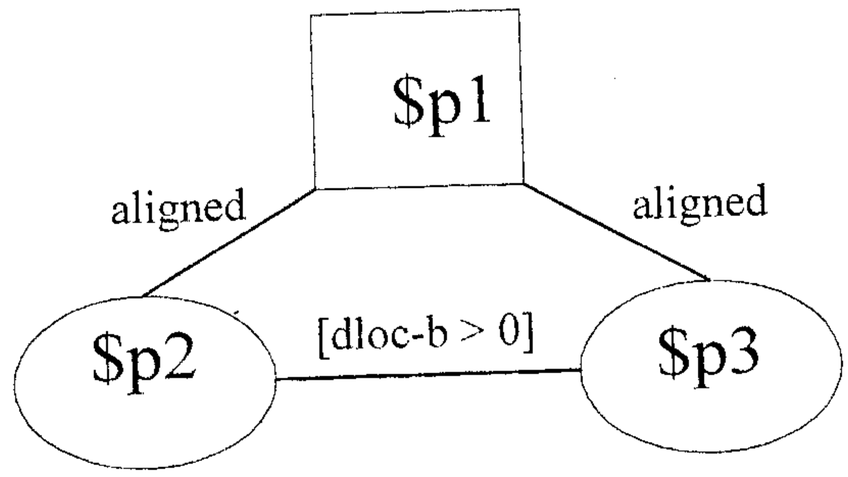


Figure 7.5 Candidate precedence schema P2.

With the two candidate precedence schemas, P1 and P2, and their grouping schema in memory, the system is given a bracket mounting assembly as a new problem to solve, as shown in Figure 7.6. Now, NOMAD applies these candidate schemas in solving the new problem. This problem is very similar to the Bell Head assembly in Figure 7.3: the Bolt is pin-shaped and the Nut and the Bracket are ring-shaped. From the grouping schema of P1 and P2, NOMAD identifies the Bolt as the base component and the Bracket and the Nut as satellite components.

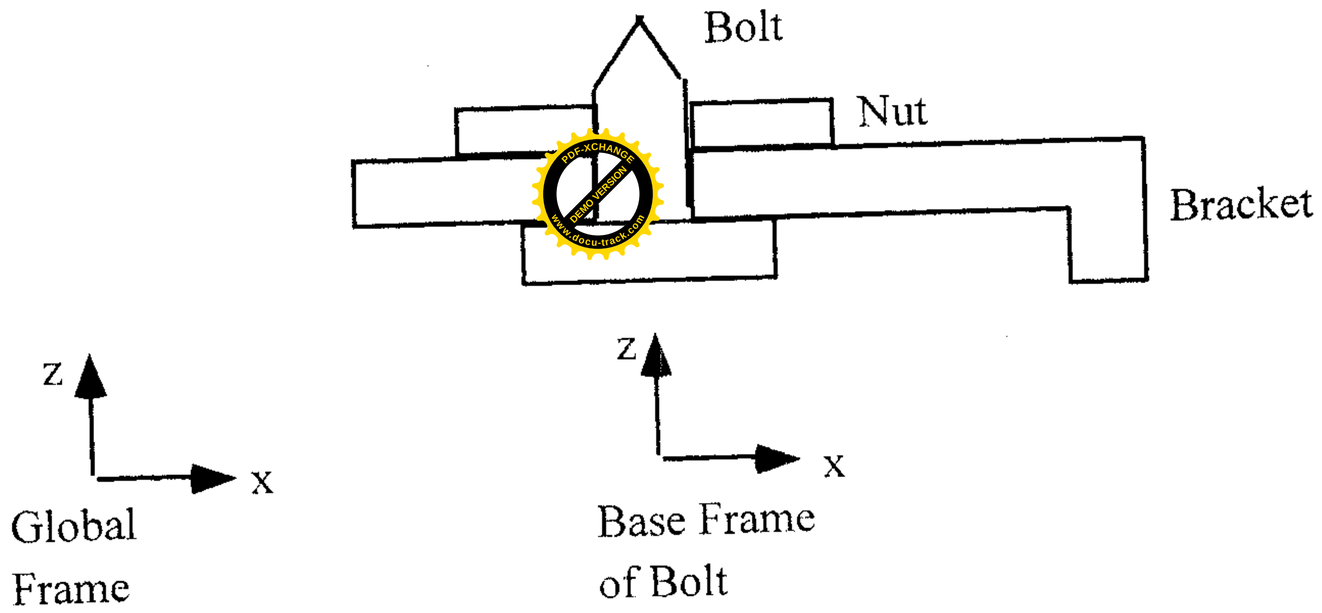


Figure 7.6 Bracket mounting assembly.

Using P1, the system plans that the Nut is attached to the Bolt first because the Nut is higher than the Bracket, as shown in Figure 7.7. However, the plan (or a hypothesis) fails for a similar reason as the case shown in Figure 7.3: the Bracket cannot be assembled if the Nut is attached first because of the spatial interference between the bracket with both the Bolt and the Nut. This failed planning episode weakens the credibility of P1.

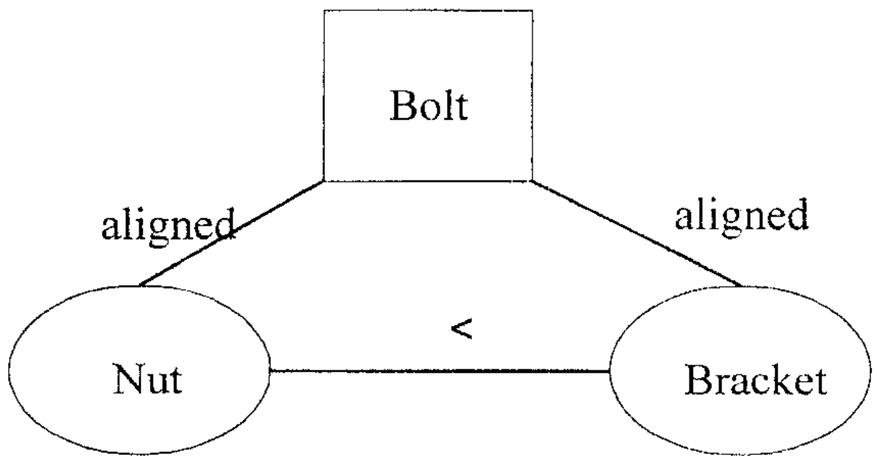


Figure 7.7 Failed application of the candidate schema P1.

An alternative plan using P2 says that the Bracket is attached to the Bolt first, as shown in Figure 7.8, because the Bracket is lower than the Nut with respect to the Bolt. This planning episode is successful and enforces the belief of P2. As an end result, P2 is going to be applied in future planning situations over invoking P1 with higher confidence.

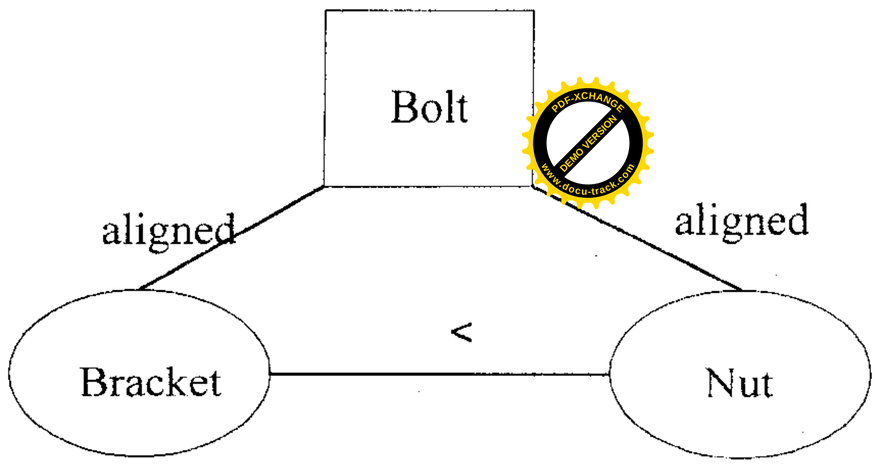


Figure 7.8 Successful application of the candidate schema P2.

Consider a screw assembly, shown in Figure 7.9. Using the grouping schema, the system recognizes the screw as a base component because it is pin-shaped. The Washers 1 and 2 are satellite components of the grouping because they are ring-shaped. Among the precedence schemas of the grouping schema, NOMAD invokes P2 over P1 from the previous planning experience. Using P2, the Washer 1 is assembled before the Washer 2 because the Washer 2 is higher than the Washer 1 with respect to the Screw. This planning episode is shown in Figure 7.10.

This learning scenario shows how the base and satellite components of an assembly are recognized, and the candidate descriptions of precedence relations are maintained or pruned in memory by actively experimenting with them in a new planning situation. The learning scenario has shown the closed-loop and constructive induction aspects of MCL by NOMAD. Now, the next example will demonstrate the use of the deductive restructuring in MCL by NOMAD. With the empirically validated schema P2, the system is presented with a new and more complicated screw assembly with three washers, as shown in Figure 7.11.

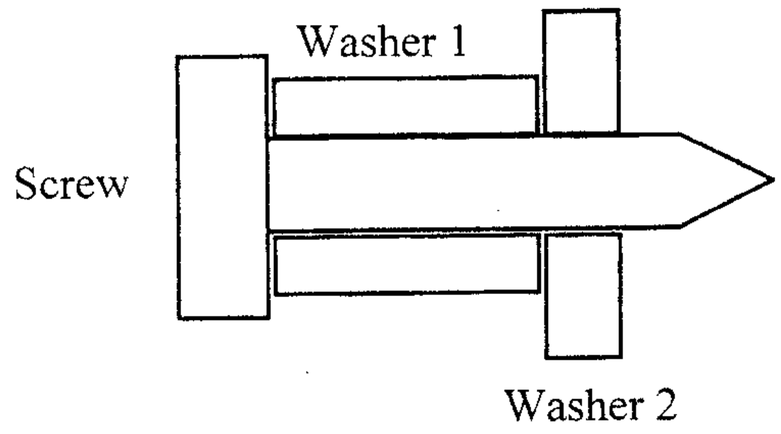


Figure 7.9. Screw assembly.

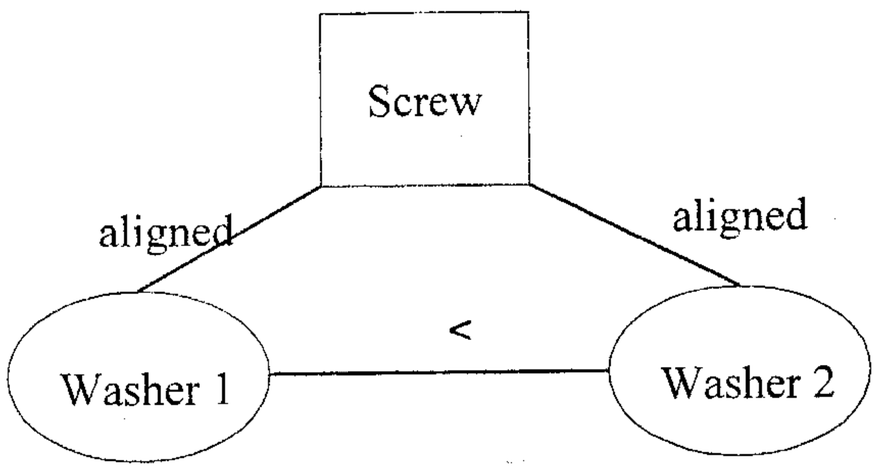


Figure 7.10 Screw assembly episode with P2.

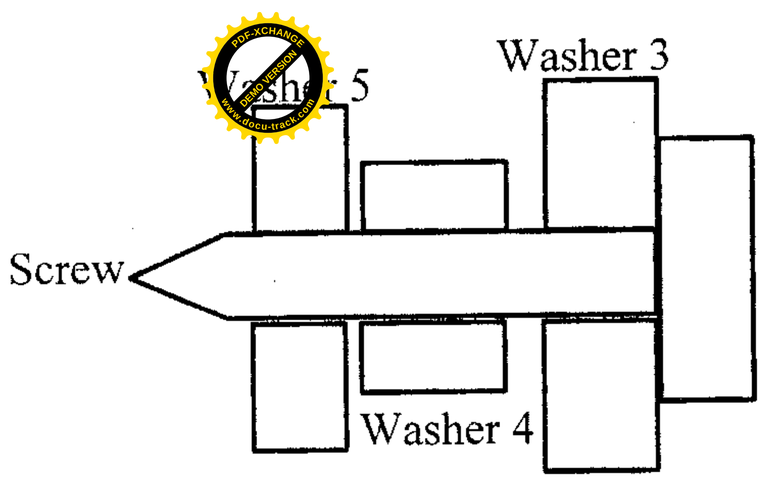


Figure 7.11 Screw assembly with three washers.

Using the grouping schema of P2, the Screw is the base component because it is pin-shaped. All the Washers 3, 4, and 5 are ring-shaped. Then, there are six possible bindings between the washers and the satellite schemas, $p2 and $p3. However, using the precedence of P2, only three of them are possible and the rest are pruned. They are shown in Figure 7.12. Episode 1 instructs assembly of the Washer 3 before Washer 4, because Washer 4 is higher than Washer 3 with respect to the Screw; Episode 2 states that Washer 3 should be assembled before Washer 5 because Washer 5 is higher than Washer 3 with respect to the Screw; likewise for the Episode 3, Washer 4 before Washer 5. These precedence constraints are all equally correct and non-conflicting with

each other, but none of them by themselves are *complete* in planning for the assembly structure of Figure 7.11.

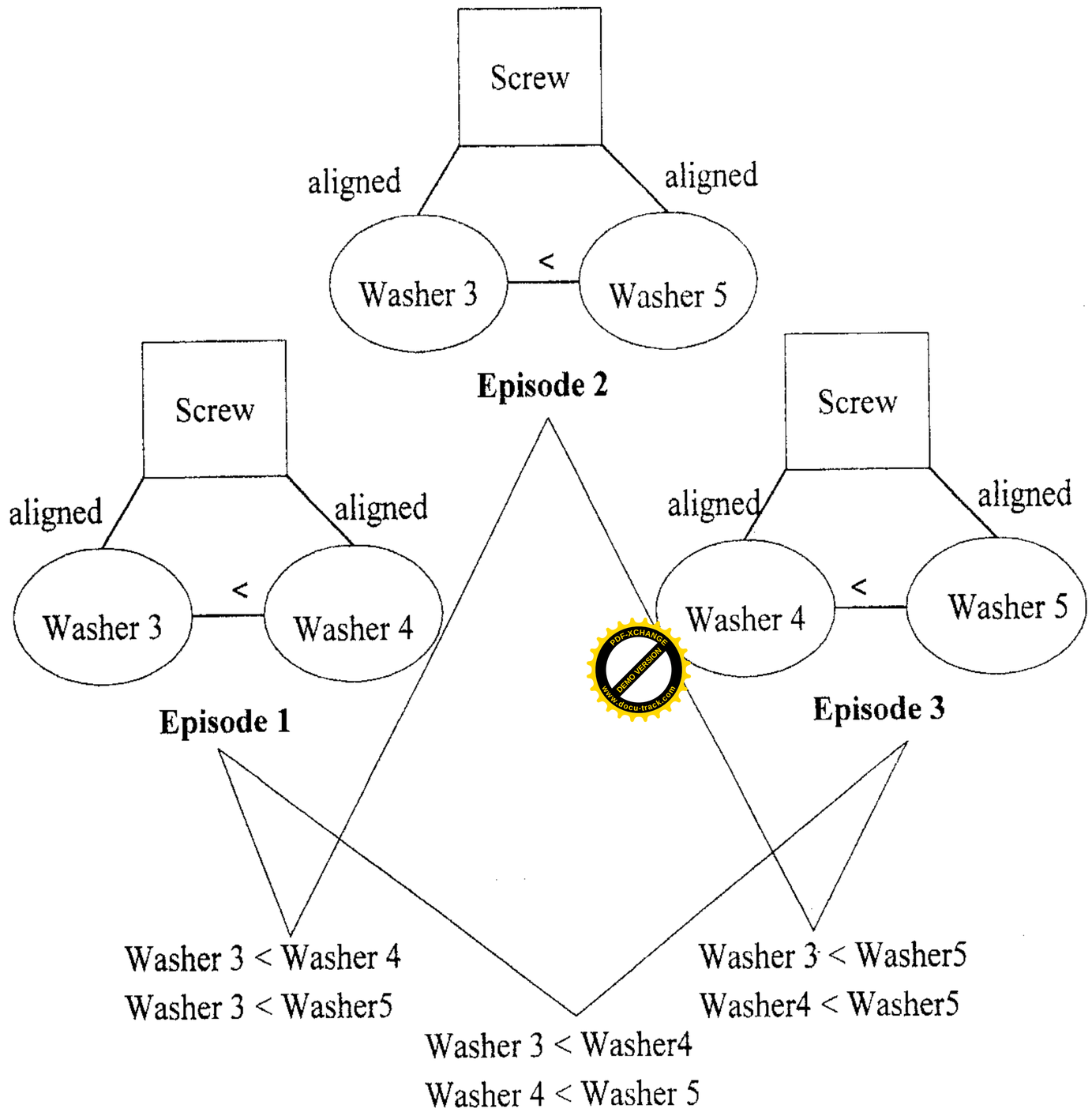


Figure 7.12 Three assembly episodes with the candidate schema P2.

When creating a complete assembly sequence plan, it is not necessary to combine all three episodes of Figure 7.12; Episodes 1 and 3 are sufficient to predict the precedence constraint of Episode 2, and therefore it is redundant. The remaining candidate assembly episodes are merged by a collapsing operator, an instance of a deductive restructuring step of MCL. Here, the collapsing operator is given two episodes at a time. From the collapse of Episodes 1 and 2, the system can predict that Washer 3 is the very first washer to assemble, but there is no precedence between Washers 4 and 5. From the collapse of Episodes 2 and 3, the system can predict that Washer 5 is the last washer to assemble, but there is no precedence relation between Washers 3 and 4. From the collapse of Episodes 1 and 3, the system predicts Washers 3, 4, and 5 are assembled in sequence. Figure 7.13 shows the collapsed structure as a new training instance. The

collapsed structure is used by NOMAD to apply a part-to-whole generalization method of the individual episodes.

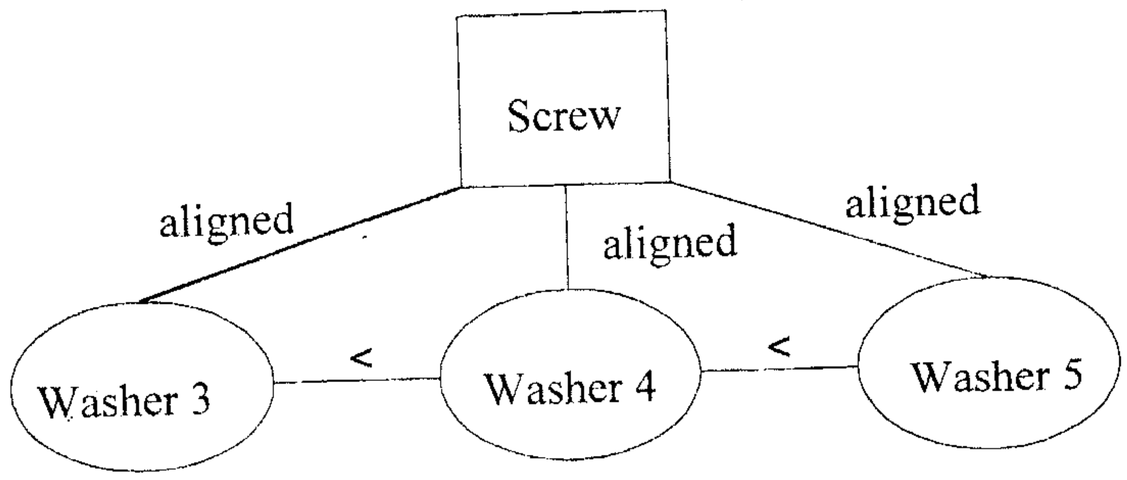


Figure 7.13 The collapsed assembly episode.

7.5 COMPARISON WITH PREVIOUS RESEARCH

Most previous planning research assumed a heuristic state space search method as a basic problem solving mechanism: plan actions as state transition operators and the plan structure as a sequence of operators to make the transitions from the initial to the goal state. This transition process turned out to be an exponential search process, and a learning component was introduced as an after-thought to reduce the search time in reaching the goal state from the initial state. In the assembly task domain, the states represent partially assembled components, the goal state represents fully assembled components, and the operators are assembly steps involving grasping, moving, stacking and others. Then, the learning component can reduce the search process in two ways:

1. by inventing a powerful heuristic that can avoid the assembly steps that are not part of the final assembly procedure, or
2. by memorizing the planning episodes as macro operators so that the search process can reduce the number of individual operator invocations.

In the former case, little is known about formulating a powerful strategic knowledge for the assembly task domain in general. In the second approach, a macro-operator schema was compiled away by a deductive generalization of a planning episode, Explanation-Based Learning (EBL) (Mitchell, Keller and Kedar-Cabelli, 1986; De Jong and Mooney, 1986). The constants of a plan episode were generalized minimally so that the operator schemas, together with the background knowledge, logically implies the macro-operator schema after the generalization. The generalization was deductive; hence, the learning process is called *analytic macro-operator schema learning*. Unfortunately, these compiled macro-operators are considered during the searching process in addition to the primitive operators, and they may create an additional burden on the search procedure. Hence, EBL macro-operator learning is too conservative to yield a new form of knowledge that can make the assembly sequencing problem tractable.

Empirical assembly planning gains fundamentally new knowledge from experience by inductively generalizing the planning episodes. The main learning task here is to

find a plausible generalization covering all the positive instances of a concept but none of the negative instances. Here, a basic learning mechanism is to compare objects as instances or non-instances of a concept to uncover common similarities and dissimilarities among the instances. Hence, the learning mechanism requires an efficient means of comparing one instance with the next.

When each instance is described by attributes, the inner structure of an instance object is abstracted out with no component objects: it is represented by a single object, namely by itself. Then, comparing one instance to the next is trivial because there is always one component object to compare. There have been numerous inductive learning methods from attribute-based training instances (Michalski, 1973; Quinlan, 1983). Unfortunately, an assembly plan structure is a highly structured object consisting of spatial and temporal relations. Therefore, an attribute-based learning technique is not directly applicable.

The comparison procedure is complicated for structured objects where a structured object consists of component objects and their relations. When comparing two structured objects, a component object from one structured object corresponds with one from the other structured object. This problem is known as the *object-correspondence* problem. When there are M distinct objects in each of two instances; there are $M!$ possible object correspondences between them, assuming one-to-one mapping (the complexity is worse otherwise). Therefore, comparing two structured objects is an inherently explosive process. To compound the difficulties, many instances should be compared to uncover regularities.

Winston's seminal work (Winston, 1970) developed a domain-independent method for learning structural concepts from the visual scene. It divided the objects of the scene into groups based on observed similarities and dissimilarities, a grouping procedure. Each grouping as a structured object is compared with an other structured object as instances or non-instances of a structural concept, a comparison procedure. The grouping procedure was later improved by CLUSTER (Stepp, 1984). It developed a number of useful similarity metrics for the clustering criteria, and an heuristic control algorithm for the hierarchical cluster formation. The comparison procedure was improved by INDUCE (Larson, 1977). It introduced an explicit two-phase approach for comparing between objects where the structural relations (links) were matched before the object attributes were compared to find the object correspondences. Each structured object is represented by a labeled graph, and used a network matching algorithm of the subgraph isomorphism problem. Although assembly structures and plans are structured objects, the domain-independent method for learning structural concepts from them is not directly applicable to learning from assembly planning experience either.

The domain independent structural concept learning systems above handle temporally structured objects like sequences no differently from any other structures. Learning a sequential pattern was studied in SPARC (Sequential PAttern ReCognition) (Michalski, Ko and Chen, 1987) as a distinct learning problem requiring a part-to-whole generalization, as opposed to an instance-to-class generalization in the previous structural concept learning systems. Here, each element of the sequence is a manifestation of some underlying process as an instance, and the learning task is to induce a sequence generating rule to explain the sequence and to predict the future continuation of the sequence as a symbolic process prediction. Hence, the learning mechanism is focused on the sequential relationships among the instances. Depending on the types of sequential relationships, SPARC has three rule models (decomposition, periodic, and disjunctive), where each rule model is looking for its distinct type of

sequential pattern. The rule model is similar to the grouping and the precedence schemas in NOMAD.

The structured concept learning systems were applied to the recognition of spatial structures like assemblies, Winston's Arch problem; the sequential pattern learning systems were applied to the symbolic process prediction. NOMAD is learning from a sequence of spatial structures (assembly structures) because the assembly sequence planning domain involves both spatially and temporally structured domain objects.

7.6 CONCLUSION

Empirical assembly planning introduces an inductive learning mechanism in an assembly planning context. Learning from planning experience must determine which concept to learn as well as the corresponding training instances from the planning experience. NOMAD decomposes the planning experience into grouping and precedence relations as assembly episodes from the grouping and precedence schemas in memory. The credibility of these schemas is incrementally updated and modified from the planning experience. Such an incremental learning in the multi-concept learning situation is a closed-loop learning. Furthermore, the learning element introduced a deductive restructuring of the episodes when generating the final assembly sequence. The deductive restructuring generates more "interesting" training instances that are not covered by the current schemas in memory so that the learner can shift the focus of its working hypothesis to a new one guided by the new instance. With the addition of a constructive induction step, these three learning steps are combined by the multistrategy constructive learning element in NOMAD.

To extend the current implementation to practical industrial use, NOMAD must have a large store of effective grouping and precedence schemas summarizing experienced planning cases in memory. An efficient means of presenting a large body of planning cases to the learning system, a virtual assembly simulation system is being developed. Here, the teacher provides an assembly description as well as the process of assembly procedure graphically using a Virtual Reality (VR) interface, a visual robot programming environment. Then, the teacher may provide a large store of assembling episodes which are the basis of applying the multi-concept learning scenario introduced in this chapter in the more practical industrial setting. When the virtual assembly system is complete, it will be a part of an intelligent CAD-based assembly modeling and simulation system for a concurrent product development and deployment environment.

ACKNOWLEDGEMENTS

I would like to express the deepest regard for the guidance given by my thesis advisor, Professor Ryszard S. Michalski. In addition, I would like to express my deepest gratitude to the members of the Machine Learning Group at the University of Illinois at Urbana-Champaign, who provided intellectual challenges and helpful feedback while I was a Ph.D. student there.

REFERENCES

De Fazio, T.L. and Whitney, D.E. (1988). Simplified Generation of All Mechanical Assembly Sequences. *IEEE Journal of Robotics and Automation.*

DeJong, G. and Mooney, R. (1986). Explanation-Based Learning: An Alternative View. *Machine Learning Journal*, vol. 2.

de Kleer, J. (1986). An Assumption-Based Truth Maintenance System. *Artificial Intelligence*, vol. 28, no. 1.

Donald, B.R. (1984). Motion Planning with Six Degrees of Freedom. *TR-91*, Massachusetts Institute of Science and Technology, Artificial Intelligence Laboratory.

Ko, H. and Lee, K. (1987). Automatic Assembly Sequence Generation. *Computer Aided Design* vol. 19, no. 1, pp. 3–10.

Ko, H. (1989). Empirical Assembly Planning: A Learning Approach. *PhD Thesis*, Department of Computer Science, Univerisity of Illinois, Urbana.

Larson, J.B. (1977). Inductive Inference in Variable-valued Predicate Logic System VL21: A Methodology and Computer Implementation. *Ph.D. Thesis, Report No. 869*, Department of Computer Science, Univerisity of Illinois, Urbana.

Michalski, R.S. (1973). Using Classifcation Rules using Variable-valued Logic System VL1. *Proceedings of the Third International Joint Conference on Artificial Intelligence*, Stanford.

Michalski, R. S. (1983). Theory and Methodology of Inductive Learning. Chapter in R.S. Michalski, J.G. Carbonell, T. M. Mitchell (Eds.), *Machine Learning: An Artificial Intelligence Approach,* Tioga Publishing.

Michalski, R. S., Ko, H. and Chen, K. (1987). Qualitative Prediction: The SPARC/G Methodology for Describing and Predicting Discrete Processes. Chapter in P. Dufour and A. Van Lamsweede (eds.), *Expert Systems*, Academic Press, pp. 125–158.

Michalski, R. S. (1993). Toward a unified theory of learning: Multistrategy task-adaptive learning. *Readings in Knowledge Acquisition and Learning* edited by B.G. Buchanan & D. Wilkins, Morgan Kaufmann.

Mitchell, T. M., Keller, T. and Kedar-Cabelli, S. (1986). Explanation-Based Generalization: A Unifying View. *Machine Learning Journal*, vol. 1.

Mostow, J. (1983). Machine Transformation of Advice into a Heuristic Search Procedure. Chapter in R. S. Michalski, J. G. Carbonell and T. M. Mitchell (eds.), *Machine Learning: An Artificial Intelligence Approach, Vol. I*, pp. 367–403, Morgan Kaufmann, Los Altos, CA.

Quinlan, J. R. (1979). Discovering Rules from Large Collections of Examples: A Case Study. Chapter in D. Michie (Ed.), *Expert Systems in the Microelectronic Age*, Edinburgh University Press, Edinburgh.

Stepp, R. E. (1984). Conjunctive Conceptual Clustering: A Methodology and Experimentation. *Ph.D. Thesis*, *UIUCDCS-R-84-1189*, Department of Computer Science, University of Illinois, Urbana.

Ulrich, K.T. and Seering W.P. (1988). Function Sharing in Mechanical Design. *Draft*, Massachusettes Institute of Technology, Artificial Intelligence Laboratory.

Winston, P. H. (1970). Learning Structural Descriptions from Examples. *Ph.D. Thesis*, *MAC TR-76*, Massachusetts Institute of Technology.