HIERARCHICAL ABSTRACTION FOR PROCESS PLANNING

Dana S. Nau[1]

Computer Science Dept., and
Institute for Advanced Computer Studies
University of Maryland

ABSTRACT

In most frame-based reasoning systems, the data manipulated by the system is represented using frames, and the problem-solving knowledge used to manipulate this data consists of rules. However, rules are not always the best way to represent problem-solving knowledge.

This paper describes an alternative way to represent problem-solving knowledge called *hierarchical knowledge clustering*. Hierarchical knowledge clustering has been implemented in a system called SIPS (Semi-Intelligent Process Selector), which plans what machining processes to use in manufacturing metal parts. The paper describes the approach to knowledge representation and problem solving used in SIPS, and compares and contrasts this approach to other work.

1. INTRODUCTION

In most frame-based reasoning systems, the information being manipulated is represented using frames, and the problem-solving knowledge that manipulates the frames consists of rules. But for some problem domains, rules may not be the most natural way to represent knowledge—and in addition, rule-based systems can require large amounts of computation during problem solving if the rule base is large.

[1]This work has been supported in part by the following sources: an NSF Presidential Young Investigator Award to Dana Nau, NSF Grant NSFD CDR-85-00108 to the University of Maryland Systems Research Center, IBM Research, General Motors Research Laboratories, Martin Marietta Laboratories, and the National Bureau of Standards.

This paper describes a way to address these problems using *hierarchical knowledge clustering*, a technique for hierarchical abstraction of problem-solving information. For some problem domains, this approach can be more natural and more efficient than rule-based problem solving.

Hierarchical knowledge clustering has been implemented in a system called SIPS (Semi-Intelligent Process Selector) [18]. SIPS was developed to produce plans of action for the creation of metal parts using metal removal operations such as milling, drilling, reaming, etc. Each of these operations or *machining processes* creates a *feature* on the metal part, such as a hole, slot, pocket, etc. Given the specification for the final part, the task of deciding what sequence or sequences of machining processes to use in creating the part is known as *process selection*. To do process selection, SIPS starts with the specification of the part to be produced, and reasons about the intrinsic capabilities of each machining process.

SIPS has recently been interfaced to a solid modeling system at General Motors Research Laboratories. This interface allows the user to create part descriptions graphically, and have SIPS select suitable machining processes to create these parts. Also, SIPS has recently been extended to do not just process selection, but also tool selection and the determination of process parameters. The latest version of SIPS is being integrated into the Automated Manufacturing Research Facility (AMRF) project [2] at the National Bureau of Standards, where it will be used to do part of the process planning for an automated machine shop.

This paper gives an overview of SIPS. Section 2 explains the motivation for the hierarchical knowledge clustering technique, and Section 3 explains how this technique has been implemented in SIPS. Section 4 discusses the relationships between SIPS and work by others, and Section 5 contains concluding remarks.

2. MOTIVATION

In most knowledge-based problem-solving systems, problem-solving knowledge consists of rules of the form "IF *conditions* THEN *action*". Even in frame systems, where the data (and possibly the knowledge base) are represented using frames, the knowledge base still usually consists of rules. However, there are several problems with using this approach for process selection.

Consider the problem of creating a hole h. There are many machining processes capable of creating holes, but to keep the example simple, suppose we consider only three processes: twist drilling, rough boring, and finish boring. Each of these processes has different restrictions how good a hole it can produce. If the restrictions for twist drilling are satisfied, twist drilling can produce h without requiring that anything else be done. However, rough boring (if its restrictions are satisfied) produces h by modifying a hole g which must already be present. Finish boring is similar to rough boring, except that it can satisfy stricter machining tolerances for h. One way to describe these processes would be rules similar to those shown in Figure 1.

R_1: IF `goal(`h`)` & $A(h)$ & $B(h)$
THEN assert `twist-drilling(`h`)`

R_2: IF `goal(`h`)` & $A(h)$ & $C(h)$ & $D(h)$
THEN remove `goal(`h`)`; assert `rough-boring(`h`)`; $g = f_1(h)$; assert `goal(`g`)`

R_3: IF `goal(`h`)` & $A(h)$ & $C(h)$ & $E(h)$
THEN remove `goal(`h`)`; assert `finish-boring(`h`)`; $g = f_2(h)$; assert `goal(`g`)`

Figure 1: A simple set of rules. A, B, C, D, and E are different sets of restrictions.

R_4: IF `goal(`h`)` & $A(h)$
THEN remove `goal(`h`)`; assert `hole-process(`h`)`

R_5: IF `hole-process(`h`)` & $B(h)$
THEN remove `goal(`h`)`; assert `twist-drilling(`h`)`

R_6: IF `hole-process(`h`)` & $C(h)$
THEN remove `goal(`h`)`; assert `hole-improve-process(`h`)`

R_7: IF `hole-improve-process(`h`)` & $D(h)$
THEN remove `goal(`h`)`; assert `rough-boring(`h`)`; $g = f_1(h)$; assert `goal(`g`)`

R_8: IF `hole-improve-process(`h`)` & $E(h)$
THEN remove `goal(`h`)`; assert `finish-boring(`h`)`; $g = f_2(h)$; assert `goal(`g`)`

Figure 2: A better set of rules.

One problem with these rules is the repititiousness of their preconditions: each rule tells what distinguishes some machining process from every other machining process in the entire knowledge base. It would be more natural and (depending on the control strategy) probably more efficient to set up context in which hole processes are the only processes being considered, and then describe each hole process only in terms of what distinguishes it from the other hole processes. This approach would lead to rules such as those shown in Figure 2.

Another problem is how to select the appropriate rule when more than one rule is applicable. For example, suppose both `twist-drilling` and `hole-improve-process` are capable of creating h. Since `twist-drilling` is less costly, one would want to use R_5 instead of R_6, but the rules include no way to assure that this will happen.

This problem could be handled if one could attach priorities to the rules corresponding to the costs of the machining processes—and rule-based systems

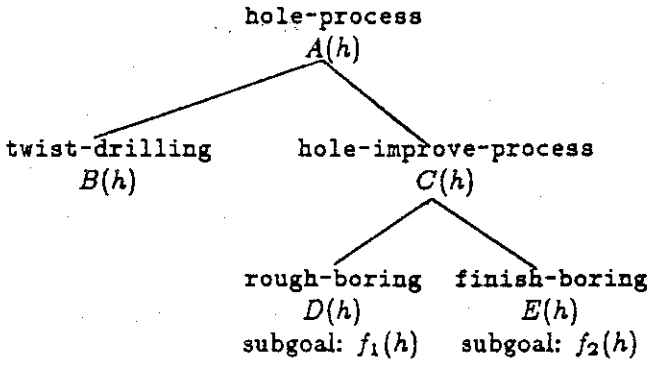

Figure 3: A tree corresponding to the rules in Figure 2.

sometimes include ways to do this. But in this case, it is not so easy: the priorities are not available beforehand to put into the rules, but instead are functions of the various machining processes. For example, the cost of a hole improvement process should be computed as the minimum of the cost of rough boring and finish boring.

One way to handle this is to notice that the rules in Figure 2 correspond to the tree shown in Figure 3. By representing each node in the tree as a frame, one could represent the process costs as slots whose values could be computed as functions of other frames. Additional slots could represent various other relevant properties of the processes—feed rates, cutting speeds, location of the machine in the factory, etc.

If we represent the machining processes in this fashion, the next question is how to represent and invoke the IF and THEN parts of the rules. Although message passing is often used in frame systems, it would not work well here, because it would still send messages to more costly processes even if a less costly process were applicable. In order to make sure that only the least-cost frames get activated, a global control strategy is needed to supervise the activation of the frames. The combination of the hierarchical representation with such a control strategy is called *hierarchical knowledge clustering*.

3. IMPLEMENTATION

Hierarchical knowledge clustering has been implemented in a system called SIPS. SIPS includes a frame system which can be used to represent both static knowledge (e.g., representations of three-dimensional objects) and problem-solving knowledge (as discussed in Section 2).

Figure 4 shows a frame structure corresponding to the tree shown in Figure 3. This frame structure is much simpler than the knowledge base actually used in SIPS (which contains about seventy frames), but it illustrates how SIPS represents problem-solving knowledge.

The `relevant` slot in the `hole-process` frame specifies that a hole process is relevant for making a hole. This information is used to start SIPS's search

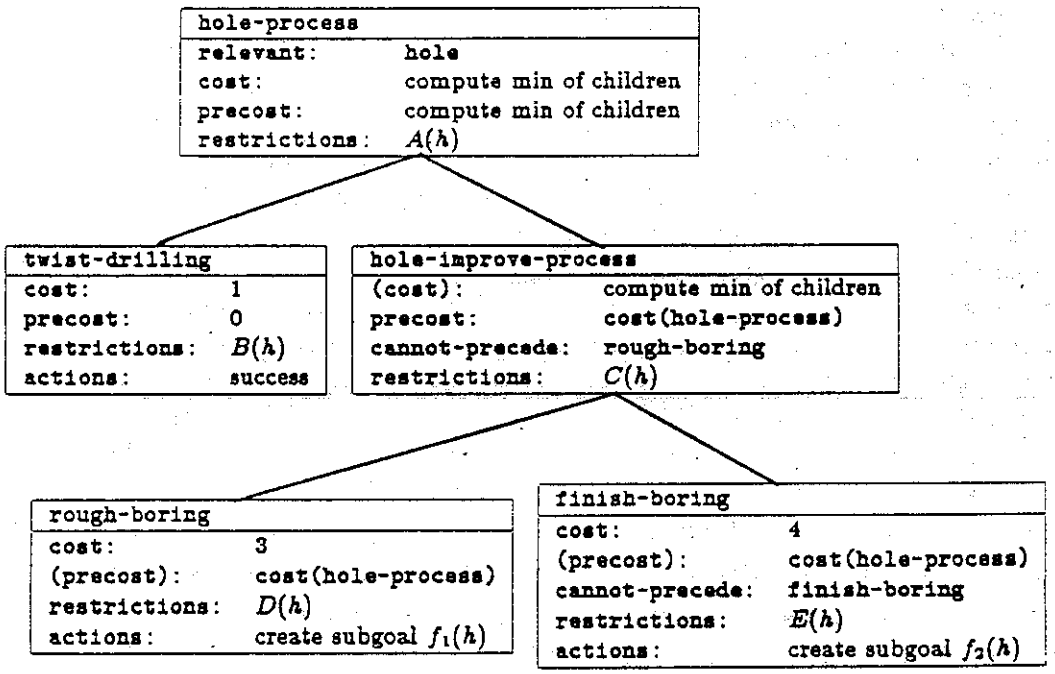

Figure 4: A frame structure corresponding to the tree shown in Figure 3. Parentheses around a slot name indicate that the slot is inherited from the parent frame.

when SIPS is told to find plan the creation of a hole.

The `cost` slot is intended to be a lower bound on the cost of performing a process. In the case of `hole-process`, this lower bound is computed by an attached procedure which takes the minimum of the `cost` slots of the child frames. `hole-improve-process` inherits this procedure from `hole-process`, so its cost will also be computed as the minimum of the costs of its children. Since the `twist-drilling`, `rough-boring`, and `finish-boring` frames represent single kinds of machining processes rather than classes of machining processes, the relative costs of these processes are put into their `cost` slots.

Similarly, `precost` is intended to be a lower bound on the cost of any other processes which might be required before doing the hole process. For `hole-process`, this bound is computed by an attached procedure which computes the minimum of the `precost` slots of the children. Since `twist-drilling` does not need to have any other processes occur before it, its `precost` slot contains the value 0. But a hole improvement process takes an existing hole g and transforms it into the desired hole—and since g must be created by some kind of hole process, the cost of creating g will be at least the minimum cost for a hole process. Thus, the `precost` slot for `hole-improve-process` is the value of `hole-process`'s `cost` slot. Both `rough-boring` and `finish-boring` inherit this value from `hole-improve-process`.

A process's `restrictions` slot tells what restrictions must be satisfied in order for that process to be a feasible way to achieve the desired goal. For `hole-process`, the restrictions are mainly geometric ones—for example, restrictions on the angle between the hole and the surface in which it is to be created. For the other processes in Figure 4, the restrictions are mainly restrictions on the hole dimensions and on the best machining tolerances achievable by the process (parallelism, roundness, true position, etc.).

The `cannot-precede` slots for `hole-improve-process` and `finish-boring` state that in no sensible process plan will these processes be followed by certain other machining processes. This slot is not really necessary for correct operation of SIPS, but it makes SIPS more efficient by decreasing the size of the search space.

SIPS does problem solving by searching backwards from the ultimate goal to be achieved. Therefore, the `actions` slot for a machining process must specify what SIPS needs to do *before* it can perform the machining process. For `twist-drilling`, nothing need be done beforehand—so `twist-drilling`'s `actions` slot states that twist drilling succeeds immediately. However, rough boring and finish boring produce a better hole from an existing hole—and SIPS needs to figure out how to make this hole. The `actions` statements for `rough-boring` and `finish-boring` set up the creation of this hole as a subgoal for SIPS.

Figure 5 shows part of the state space which can be generated from the set of frames shown in Figure 4. Each state in the state space is a (partial) plan for creating a hole `h1`. Whether or not this plan is feasible will depend on the nature of `h1`—except that the plans marked "infeasible" in Figure 5 can never be feasible, because of the `cannot-precede` slots in the knowledge

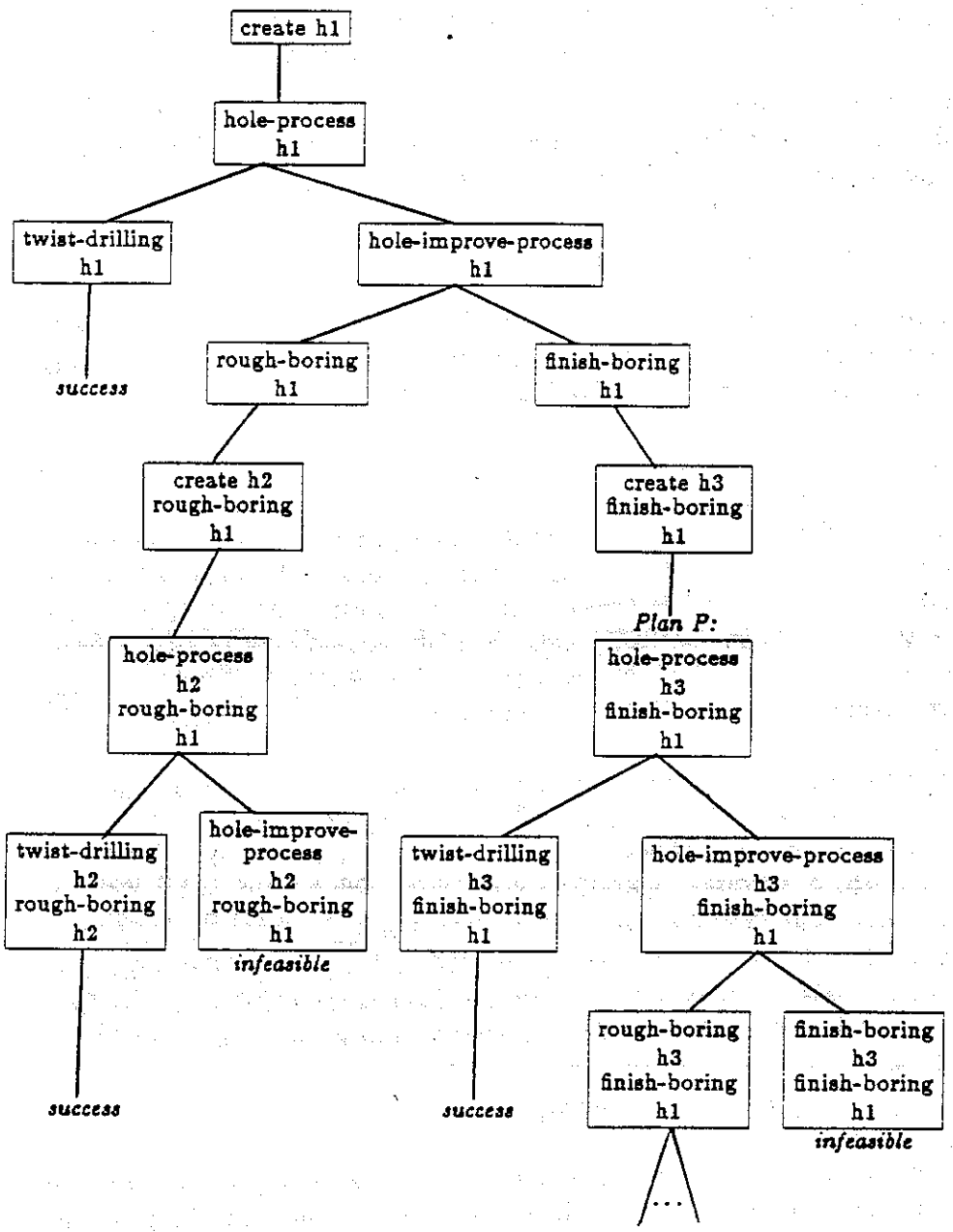

Figure 5: Part of a search space for creating a hole h1. Plan P is labeled for reference in the text.

base. When a plan is infeasible, its children will never be generated.

SIPS searches the state space using an adaptation of Branch and Bound. The lower bound function LB which guides this search is computed from the `cost` and `precost` slots of the machining processes. For example, for the plan labeled P in Figure 4,

$$LB(P) = \texttt{precost(hole-process)} + \texttt{cost(hole-process)} + \texttt{cost(finish-boring)}.$$

So that SIPS will avoid generating expensive plans when cheaper ones can be used, SIPS's search strategy is best-first.[2] Thus, the first solution found by SIPS is guaranteed to be the least costly one.

4. RELATION TO OTHER WORK

This section discusses the relationships between SIPS and other work in three areas: automated process planning, planning with abstraction, and computational approaches for knowledge-based systems.

4.1. Process Planning

A number of computer systems exist which provide partial automation of process planning. In most existing systems, process planning is done by retrieving from a data base a process plan for another part similar to the desired part, and modifying this plan by hand to produce a process plan for the desired part. Examples of such systems are CAPP [12] and MIPLAN [24]. For more detailed descriptions of such systems, the reader is referred to [4] and [19].

Devising a complete process plan automatically using a part's specifications (e.g., a full technical drawing) is a very difficult problem. There are several systems which attempt to produce a process plan for the exact part desired—but most such systems are experimental and have limited capabilities. A few of the better-known systems include CPPP [8], APPAS [26], CADCAM [3,6], TIPPS [5], GARI [7] and TOM [13], and SIPP [16,17] (a predecessor to SIPS, implemented in Prolog). Except for SIPP, these systems use problem-solving approaches rather different from what is used in SIPS.

4.2. Planning with Abstraction

Hierarchical knowledge clustering can be viewed as a way to do planning based on abstraction. For example, the `hole-process` frame in Figure 4 represents an abstract machining process which has two possible instantiations: `twist-drilling` and `hole-improve-process`.

Several types of abstraction have been explored in the literature on planning. One type of abstraction is that used in NOAH [23], in which an action

[2] Thus, SIPS's search procedure may also be thought of as an adaptation of A* [20], with LB as the heuristic function.

A is an abstraction of actions A_1 and A_2 if A_1 and A_2 are each steps in the performance of A. This is rather different from the abstraction used in SIPS: in SIPS, A is an abstraction of actions A_1 and A_2 if A_1 and A_2 are alternate instantiations of A.

Another type of abstraction is that used in ABSTRIPS [20], in which a complete plan is constructed ignoring some of the preconditions of each action and the plan is then modified to meet the preconditions which were ignored. This type of abstraction is related to that used in SIPS in the following sense: an instantiation of an action A is an action A_1 which must satisfy the preconditions of A and also some additional preconditions, and both SIPS and ABSTRIPS refine a plan containing A by checking those preconditions of A_1 which differ from the preconditions of A. However, there are several important differences:

1. SIPS completely instantiates the last action in a plan before considering what actions should precede this action, whereas ABSTRIPS generates a complete (but possibly incorrect) plan and then tries to fix it up.

2. In SIPS, an abstract action has several possible alternate instantiations, but in ABSTRIPS, only one instantiation is possible. Thus in ABSTRIPS, the notion of considering alternate instantiations of an action and choosing the one of least estimated cost does not make sense.

Another type of abstraction which is quite close to that used in SIPS is proposed by Tenenberg [22]. This approach is similar to SIPS in the sense that each abstract action may have more than one possible instantiation. It is potentially more general than that used in SIPS, in the sense that the effects of actions are represented hierarchically, as well as their preconditions—but so far, Tenenberg's approach has not yet been implemented.

Several systems for diagnostic problem-solving make use of certain kinds of taxonomic hierarchies. Both MDX [14] and Centaur [10] use taxonomies of various diagnostic problems, in which knowledge about each class of problems is located at the node in the hierarchy which represents that class. These approaches yield some of the same benefits as SIPS in terms of representational clarity and efficiency of problem-solving. However, the details of how they represent and manipulate their knowledge are rather different from what SIPS does.

4.3. Computational Approaches

It is well known that rule-based systems having large rule bases can require substantial computational overhead. Suppose a rule-based system is trying to solve a problem in some problem domain D. Each time the system applies a rule, this changes the system's current state S—and in order to decide what rule to apply next, the system must determine which rules match S. If the system searched through its entire set of rules to find the ones matching S, the computational overhead would be tremendous.

Several approaches have been tried for alleviating this problem. One approach, which is used in KEE [9], is to provide facilities whereby the user can divide a set of rules R into smaller subsets $R_1, R_2, \ldots, R_n$, such that each subset is relevant for a different problem domain. Given a problem to solve, the system starts out by determining which problem domain the problem is in. It then selects the rule set R_i for that domain, and then uses R_i exclusively from that point on, ignoring all the other rules. Since R_i is smaller than R, the problems with efficiency are lessened.

Hierarchical knowledge clustering can be thought of as as an extension of the above approach. It provides a way to tell, directly from the current state S, that only some subset R_S of the rules in R is relevant to S.[3] Thus, all rules not in R_S can temporarily be ignored. Since R_S is normally quite small, this provides improved efficiency.

Another approach to reducing the computational overhead of computing rule matches is the rete match algorithm used in OPS5 [11] and YAPS [1]. This algorithm provides a way to store partial rule matches in a network so that the system can determine whether a rule matches the current state without having to re-evaluate all of the rule's preconditions each time the current state changes. This makes the complexity of computing rule matches depend not on the size of R, but instead on the size of the set P_S of rules whose preconditions partially match S. If P_S is small, then the rete match procedure is efficient, but if P_S is large, the elaboration of partial matches may incur significant overhead.

Hierarchical knowledge clustering can be thought of as a way to control the elaboration of partial matches, by distributing the preconditions of a rule throughout the levels of a hierarchical structure and elaborating a partial match only if it looks promising. Thus, the approach used in SIPS may have potential for increasing the efficiency of the rete match procedure.

5. CONCLUDING REMARKS

SIPS currently runs in Franz Lisp on a Sun, and in Zeta Lisp on a Symbolics Lisp Machine and a TI Explorer. The current knowledge base consists of about seventy frames describing various machining processes and machinable features. SIPS can either read prepared data from a file, or (if some of this data is omitted) run interactively, asking the user for any needed information. Various user features have been implemented in SIPS. For example, if SIPS produces a plan for producing some feature, the user can later tell SIPS to go back and find other alternative plans for producing this feature.

For the process planning problem domain, hierarchical knowledge clustering appears to be more natural to use than a "flat" set of production rules. In the experience of a manufacturing engineer who has worked on SIPS's knowledge base, SIPS's style of knowledge representation has been easy to under-

[3] In particular, finding R_S corresponds either to retrieving the children of some frame or (when SIPS creates a subgoal) retrieving all frames relevant to the creation of a feature. In each case, only a few of SIPS's process frames are relevant—and which frames are relevant is determined easily from the frame system.

stand and use. Trying to represent SIPS's knowledge base as a rule-based system would make the rules very cumbersome.

A more sophisticated interface for SIPS is currently being developed. SIPS has been interfaced to a solid modeling system at General Motors Research Laboratories, so that the user can build up an object to be created by giving graphical specifications of its machinable features, and have SIPS select sequences of machining processes capable of creating those features. Further work on solid modeling for SIPS is currently underway [25].

SIPS is being extended to do not just process selection, but also tool selection and the determination of process parameters. This is being done by giving SIPS a knowledge base for tooling in addition to its knowledge base for process selection. Thus, the current knowledge base for SIPS consists of three hierarchies: a taxonomy of machinable features, a taxonomy of machining processes, and a taxonomy of cutting tools. Once SIPS finds a successful sequence of machining processes for a given machinable surface, it uses its knowledge about the characteristics of each cutting tool to decide, for each machining process, what cutting tool to use and what process parameters to use. The latest version of SIPS is being integrated into the Automated Manufacturing Research Facility (AMRF) project [2] at the National Bureau of Standards, where it will be used as part of the process planning system in an automated machine shop.

References

[1] E. M. Allen, "Yaps: Yet Another Production System," Tech. Report TR-1146, Computer Science Dept., Univ. of Maryland, College Park, MD, Feb. 1982.

[2] P. F. Brown and C. R. McLean, "Interactive Process Planning in the AMRF," *Symposium on Knowledge-Based Expert Systems for Manufacturing at ASME Winter Annual Meeting*, Anaheim, CA, Dec. 1986, pp. 245-262.

[3] T. C. Chang, "Interfacing CAD and CAM - A Study of Hole Design," M.S. Thesis, Virginia Polytechnic Institute, 1980.

[4] T. C. Chang and R. A. Wysk, *An Introduction to Automated Process Planning Systems*, Prentice-Hall, Englewood Cliffs, NJ, 1985.

[5] T.C. Chang and R. A. Wysk, "Integrating CAD and CAM through Automated Process Planning," *International Journal of Production Research* 22:5, 1985.

[6] T. C. Chang and R. A. Wysk, "An Integrated CAD/Automated Process Planning System," *AIIE Transactions* 13:3, Sept. 1981.

[7] Y. Descotte and J. C. Latombe, "GARI: A Problem Solver that Plans How to Machine Mechanical Parts," *Proc. Seventh International Joint Conf. Artif. Intel.*, Aug. 1981, pp. 766-772.

[8] M. S. Dunn, Jr., J. D. Bertelsen, C. H. Rothauser, W. S. Strickland, and A. C. Milsop, "Implementation of Computerized Production Process Planning," Report $R_8$1-945220-14, United Technologies Research Center, East Hartford, CT, June 1981.

[9] R. Fikes and T. Kehler, "The Role of Frame-Based Representation in Reasoning," Communications of the ACM **28:9**, Sept. 1985, pp. 904-920.

[10] P. Jackson, "Introduction to Expert Systems," Addison-Wesley, Wokingham, England, 1986, pp. 142-157.

[11] C. L. Forgy, "The OPS5 User's Manual," Tech. Report CMU-CS-81-135, Computer Sci. Dept., Carnegie-Mellon University, 1980.

[12] C. H. Link, "CAPP—CAM-I Automated Process Planning System," *Proc. 13th Numerical Control Society Annual Meeting and Technical Conference*, Cincinnati, March 1976.

[13] K. Matsushima, N. Okada, and T. Sata, "The Integration of CAD and CAM by Application of Artificial-Intelligence," *CIRP*, 1982, pp. 329-332.

[14] S. Mittal, B. Chandrasekaran, and J. Smith, "Overview of MDX–A System for Medical Diagnosis," *Proc. Third Annual Symposium on Computer Applications in Medical Care*, Washington, DC, Oct. 1979.

[15] D. S. Nau and T. C. Chang, "Prospects for Process Selection Using Artificial Intelligence," *Computers in Industry* 4, 1983, pp. 253-263.

[16] D. S. Nau and T. C. Chang, "A Knowledge-Based Approach to Generative Process Planning," *Production Engineering Conference at ASME Winter Annual Meeting*, Miami Beach, Nov. 1985, pp. 65-71.

[17] D. S. Nau and T. C. Chang, "Hierarchical Representation of Problem-Solving Knowledge in a Frame-Based Process Planning System," *Jour. Intelligent Systems* 1:1, 1986, pp. 29-44.

[18] D. S. Nau and M. Gray, "SIPS: An Application of Hierarchical Knowledge Clustering to Process Planning," *Symposium on Integrated and Intelligent Manufacturing at ASME Winter Annual Meeting*, Anaheim, CA, Dec. 1986, pp. 219-225.

[19] D. S. Nau, J. A. Reggia, M. W. Blanks, Y. Peng, and D. Sutton, "Prospects for Knowledge-Based Computing in Automated Process Planning and Shop Control," Tech. Report, Computer Science Dept., University of Maryland, College Park, Feb. 1984.

[20] N. J. Nilsson, *Principles of Artificial Intelligence*, Tioga Press, Palo Alto, 1980, pp. 350-354.

[21] C. Ramsey, J. A. Reggia, D. S. Nau, and A. Ferrentino, "A Comparative Analysis of Methods for Expert Systems," *Internat. Jour. Man-Machine Studies*, 1986.

[22] J. Tenenberg, "Planning with Abstraction," *Proc. AAAI-86*, Philadelphia, 1986, pp. 76-80.

[23] E. Sacerdoti, *A Structure for Plans and Behavior*, American Elsevier, New York, 1977.

[24] TNO, "Introduction to MIPLAN," Organization for Industrial Research, Inc., Waltham, MA, 1981.

[25] G. Vanecek and D. Nau, "Computing Geometric Boolean Operations by Input Directed Decomposition," in preparation, 1987.

[26] R. A. Wysk, "An Automated Process Planning and Selection Program: APPAS," Ph.D. Thesis, Purdue University, 1977.