ORIGINAL

Expert Systems: The User Interface

J. Hendler, editor

FIVE

Hierarchical Knowledge Clustering: A Way to Represent and Use Problem-solving Knowledge*

Dana Nau
Michael Gray

University of Maryland

Abstract

In most frame-based reasoning systems, the data manipulated by the system are represented using frames, but the problem-solving knowledge used to manipulate this data is represented using rules. However, this is not always the best approach. Rules are not always a natural way to represent knowledge—and in addition, rule-based systems containing large knowledge bases may require large amounts of computation in order to determine which rules match the current state in a problem to be solved. This chapter describes a way to address these problems using a new technique called hierarchical knowledge clustering.

A prototypical version of hierarchical knowledge clustering was implemented in Prolog, in a system called SIPP. An improved version has been implemented in Lisp, in a system called SIPS (Semi-Intelligent Process Selector), which plans what machining processes to use in manufacturing metal parts. This chapter gives an overview of SIPS, and describes its knowledge representation and problem solving methods.

* This work was supported in part by an NSF Presidential Young Investigator Award, IBM Research, General Motors Research Laboratories, Martin Marietta Laboratories, the National Bureau of Standards, and NSF grant NSFD CPR-85-00108 to the University of Maryland Systems Research Center.

1. Introduction

In many frame-based reasoning systems, the information being manipulated is stored in the form of frames, but the problem-solving knowledge that manipulates the frames is stored separately in the form of rules. One problem with this approach is that rules are not always a natural way to represent knowledge; another is that systems containing lots of rules may require excessive amounts of computation to determine which rules match the current state during problem solving.

This chapter describes a way to address these problems using a new approach called *hierarchical knowledge clustering*. In this approach, problem-solving knowledge is organized into a taxonomic hierarchy in which each node represents a set of possible actions, and its children represent different subsets of that set. Restrictions are associated with each node which determine whether or not the set of actions represented by that node are feasible actions to perform, and problem solving is done using an adaptation of Branch and Bound. For some problem domains, this approach can be more natural than rule-based representation, and can alleviate the computational inefficiencies that can arise with rule-based systems.

Hierarchical knowledge clustering was first implemented prototypically in Prolog, in a system called SIPP (Nau & Chang, 1986). Experience with SIPP led to refinements of the idea, resulting in a second implementation—this time in Lisp. The Lisp implementation, called SIPS (Semi-Intelligent Process Selector), is the topic of this chapter.

This chapter gives an overview of SIPS, and describes its knowledge representation and problem-solving methods. The chapter also discusses the implications of this work for AI knowledge representation and problem solving.

2. Problem Domain

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 is called a *machining process*, and each machining process is used to create a *feature* on the metal part, such as a hole, a slot, a pocket, etc. Given a specification for what the final part is supposed to look like, the task of deciding which sequence of machining processes to use in creating the part is known as *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. (For more detailed descriptions of such systems, cf. Nau & Chang, 1985; Nau et al., 1984.)

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. Systems which use AI techniques include GARI (Descotte & Latombe, 1981), TOM (Matushima, Okada, & Sata, 1982), and SIPP (a predecessor to SIPS, implemented in Prolog; cf. Nau & Chang, 1986).

The approach used in both SIPP and SIPS is to reason about the intrinsic capabilities of each machining operation in order to produce least-cost process plans. Further extensions of SIPS are expected to address fundamental research issues in reasoning about three-dimensional objects. SIPS is currently being integrated into the AMRF (Automated Manufacturing Research Facility) project at the National Bureau of Standards, where it will be used in producing process plans for an automated machine shop; and plans are under way for integrating it with software being developed at General Motors Research Laboratories.

3. Motivation

In most knowledge-based problem-solving systems that use frames, problem solving is done by manipulating the frames using rules of the form "**if** *conditions* **then** *action*." This approach has proved quite powerful in a number of problem domains, but in some circumstances problems can arise.

One problem with rule-based systems is the problem of efficiency; this problem will be discussed in more detail in Section 6. Another problem is that the way in which the knowledge is represented can sometimes be unnatural. Several ways in which this can occur have been pointed out in the literature (Davis, Buchanan, & Shortliffe, 1977; Reggia, Nau, & Wang, 1983), but the particular one we will consider is this: Since human beings often approach problems hierarchically, a hierarchical representation of problem-solving knowledge can sometimes be easier to understand.

For example, suppose someone is writing a knowledge base about the following milling processes:

rough face milling
finish face milling
rough end milling
finish end milling
rough peripheral milling
finish peripheral milling

The knowledge base presumably consists of a set of rules to determine which of these processes to use in creating some goal feature *f*. Each process has various restrictions on its capabilities, but since they are all milling processes, some of these restrictions will be common to all of the processes. And since rough face milling and finish face milling are both face-milling operations, they will have even more restrictions in common. If one were to write a single rule for each operation, the result might be the set of rules shown in Figure 1, where *A*, *B*, . . . , *J* are different sets of restrictions and *S* is the current state.

One problem with this set of rules is that it does not include any way to decide which rule to invoke when more than one rule is applicable. Suppose, for example, that *f* may be made either by rough face milling or by rough end milling. If face milling is a less costly process than end milling, then one would want R1 to fire instead of R3, and the rules include no way to assure this. What is needed is to attach priorities to the rules corresponding to the costs of the machining processes. Although there is no conceptual difficulty with doing this (it is in some ways analogous to the certainty factors used in Mycin; cf. Davis, Buchanan, & Shortliffe, 1977), the problem is that these priorities are not really available beforehand, but need to be computed in the knowledge base as functions of other machining processes. (How this needs to be done is illustrated in Section 4 in the discussion of SIPS's and `projected cost` slots

Another problem is that the approach illustrated in Figure 1 is not particularly natural—it requires that for each machining operation, one must describe the characteristics that distinguish this operation from

R1: if *S* = '*goal(f)*' & *A(f)* & *B(f)* & *C(f)* then *S* := '*rough-face-mill(f)*'
R2: if *S* = '*goal(f)*' & *A(f)* & *B(f)* & *D(f)* then *S* := '*finish-face-mill(f)*'
R3: if *S* = '*goal(f)*' & *A(f)* & *E(f)* & *F(f)* then *S* := '*rough-end-mill(f)*'
R4: if *S* = '*goal(f)*' & *A(f)* & *E(f)* & *G(f)* then *S* := '*finish-end-mill(f)*'
R5: if *S* = '*goal(f)*' & *A(f)* & *H(f)* & *I(f)* then *S* := '*rough-peripheral-mill (f)*'
R6: if *S* = '*goal(f)*' & *A(f)* & *H(f)* & *J(f)* then *S* := '*finish-peripheral-mill(f)*'

Figure 1. Rules Telling which Milling Operations to Use to Create a Feature *f*.
Each of *A*, *B*, . . . , *J* is a Different Set of Restrictions.

if *S* = '*goal(f)*' & *A(f)* then *S* := '*mill(f)*'
if *S* = '*mill(f)*' & *B(f)* then *S* := '*face-mill(f)*'
if *S* = '*mill(f)*' & *E(f)* then *S* := '*end-mill(f)*'
if *S* = '*mill(f)*' & *H(f)* then *S* := '*peripheral-mill(f)*'
if *S* = '*face-mill(f)*' & *C(f)* then *S* := '*rough-face-mill(f)*'
if *S* = '*face-mill(f)* & *D(f)* then *S* := '*finish-face-mill(f)*'
if *S* = '*end-mill(f)*' & *F(f)* then *S* := '*rough-end-mill(f)*'
if *S* = '*end-mill(f)*' & *G(f)* then *S* := '*finish-end-mill(f)*'
if *S* = '*peripheral-mill(f)*' & *H(f)* then *S* := '*rough-peripheral-mill(f)*'
if *S* = '*peripheral-mill(f)*' & *I(f)* then *S* := '*finish-peripheral-mill(f)*'

Figure 2. A Modified Set of Rules for Milling Operations.

every other machining operation in the entire knowledge base. It would be more natural—and probably more efficient—to write rules that describe each of the milling processes only in terms of what distinguishes it from other milling processes, and use these rules only after it has been decided that milling can be used to create *f*. This approach would yield the set of rules shown in Figure 2.

The rules shown in Figure 2 still do not provide a way to handle the computation of the process costs and the use of these costs in determining rule priorities, but it is interesting to note that each of the rules shown in Figure 2 corresponds to a node of the tree shown in Figure 3. If one were to represent each node in the tree as a frame, one could represent the process costs as values of slots in these frames, and compute the process costs as functions of slots in other frames. One could also represent various other relevant properties of the processes—feed rates, cutting speeds, location of the machine in the factory, etc.

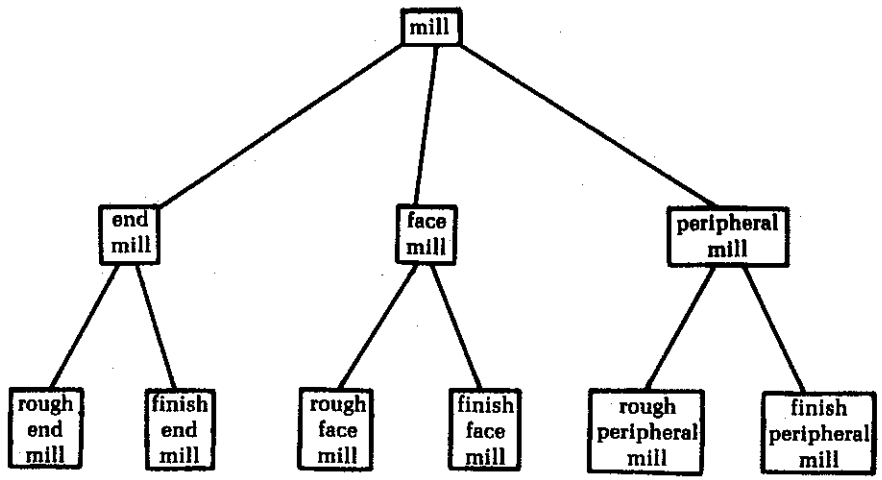

Figure 3. A Tree Corresponding to the Rules in Figure 2

The biggest problem with representing the processes as frames would be how to represent and invoke the **if** and **then** parts of the rules. One approach would be to use procedural attachment and message passing, using the following protocol: If a "consider-yourself-for-creating(*f*)" message were sent to a frame (say, the *mill* frame) and if this frame's restrictions (in this case, *A*(*f*)) were satisfied, then this frame would send the same message to each of its children (the *face-mill*, *end-mill*, and *peripheral-mill* frames). However, this approach would make it very difficult to keep the message from being sent to the *end-mill* frame if face milling were less costly than end milling.

A solution to this problem would be not to use message passing, but instead to write a global control strategy to supervise the activation of the frames. Such a strategy would maintain a list of all frames eligible to be activated, activating first those of least cost. The combination of this control strategy with the frame representation shown in Figure 3 we refer to as *hierarchical knowledge clustering*.

In addition to being a more natural way to organize information about some problem domains (such as process planning), hierarchical knowledge clustering can also be more efficient computationally than a rule-based approach. This is discussed in more detail in the concluding section of this chapter.

4. Knowledge Representation

As described in the previous section, hierarchical knowledge clustering is applicable to situations where the problem-solving knowledge corresponds to a set of actions which can be organized into a taxonomic hierarchy. The hierarchy is set up so that each node represents a set of possible actions that can be performed, and the node's children represent different subsets of that set. Each node is represented by a frame which has restrictions associated with it. A node is eligible for consideration only if the restrictions associated with its parent are satisfied. If a node is eligible for consideration, it may or may not actually be considered, depending on the choices made by a global control strategy. To make this idea clearer, we discuss the way it is implemented in SIPS.

SIPS has two basic types of frames: archetypes and items. Archetypes correspond roughly to sets, and items correspond to members of sets. Normally, the knowledge about a problem domain consists of archetypes, and the data used in solving a specific problem in that domain (both the input data defining the problem and the intermediate data created while the problem is being solved) consist of items. Items

have slots into which values can be stored, and archetypes have slots into which default values can be stored. The frame system has general inheritance mechanisms which allow default values either to be inherited in the usual way, or to be computed as arbitrary functions of values stored in other frames.

One purpose of SIPS's frame system is to represent static knowledge about features (holes, pockets, flat surfaces, slots, etc.). These features are organized into the obvious kind of taxonomy, with archetypes representing abstract features and items representing specific instantiations of those features. For example, the `hole` archetype is a child of the `cylindrical-surface` archetype, which is a child of the `surface` archetype. A specific hole in a specific metal part might be represented bx an item called, say, `*hole-21`. This is not particularly different from how such knowledge would be represented in any other frame system, so we will not discuss it further.

Another purpose of the frame system is to represent problem-solving knowledge—the kind of knowledge which other systems would represent using rules. Information about the capabilities of machining processes is organized into a taxonomy similar to the taxonomy used for features, with archetypes representing abstract machining processes and items representing specific instantiations of those processes. For example, the `twist-drill` archetype is a child of the `hole-create-process` archetype, which is a child of the `hole-process` archetype; and the specific twist drilling process used to create `*hole-21` might be represented by an item called `*twist-drill-13`.

The rest of this section consists of a simple example (much simpler than the information actually appearing in SIPS's knowledge base) of how SIPS represents problem-solving knowledge. The first statement in the example is this:

```
(defarchetype process () (
    (cost $type posnumberp
          $init (min-child-cost current-frame 'cost)
    (projected-cost $type posnumberp
          $init (min-child-cost current-frame
          'projected-coat)))
```

This statement defines an archetype called `process` which has two slots called `cost` and `projected-cost`. The `$type` specification (which is used to specify data types for slots) says that any values put into the `cost` and `projected-cost` slots must satisfy a LISP predicate called `posnumberp`.

The `cost` and `projected-cost` slots are intended to contain lower bounds on, respectively, the cost of performing a machining process

and the cost of any other processes which might be required beforehand. To achieve this intent, the $init specification (which is used to specify initial values for slots) says that the initial value for the cost slot is the minimum of the cost slots of process's children, and the initial value for projected-cost is the minimum of the projected-cost slots of these children. Since the only child of process is hole-process (see below), this means that the initial values are 1 for cost and 0 for projected-cost.

```
(defarchetype hole-process (process) ())
```

This statement says that hole-process is a child of process. No slots are given explicitly for hole-process, but since hole-process is a child of process, it has cost and projected-cost slots whose initial values are the minimum values of these slots in hole-process's children. Since the children of hole-process are twist-drill and rough-bore (see below), this means that the initial values are 1 for cost and 0 for projected-cost.

```
(relevant hole-process hole)
```

This statement says that hole-process is relevant for creating any item which is a hole. In general, when SIPS starts planning the creation of some feature, it starts by looking at all processes which have been declared to be relevant for creating the feature. n this example, neither this statement nor any other statements declare any other process to be relevant for creating a hole, so to create a hole SIPS would start by considering only hole-process. A more detailed description of what SIPS would do at this point appears in Section 5.

```
(defrestriction hole-process (h) (surface)
    (setq surface (getval h 'contained-in))
    (equal (get-archetype surface) 'flat)
    (parallel (getval h 'axis) (getval surface
        'norm)))
```

Every machining process (or class of machining processes) has restrictions on its capabilities—for example, restrictions on the size of the feature it can create, and restrictions on how tight a set of tolerances it can achieve. The above statement describes those restrictions which apply to every hole-process—and since twist-drill and rough-bore are both children of hole-process (see below), this means that the restrictions given in this statement must be satisfied before SIPS will consider twist-drill or rough-bore.

In this statement, h is a parameter which SIPS will bind to the item

feature describing the frame to be created, surface is a local variable, and the rest of the statement is a conjunct of conditions to be satisfied. These conditions state that a hole-process can be used to create a hole h only if the surface containing h is a flat surface whose normal vector is parallel to the axis of h.

```
(defarchetype twist-drill (hole-process) (
    (cost $init 1)
    (projected-cost $init 0)))
```

If SIPS has decided that a hole-process can be used to create some feature, then any child of the hole-process frame may be considered. According to the above statement, one of these children is twist-drill. The statement says that twist-drill's cost and projected-cost slots have initial values of 1 and 0, respectively.

```
(defrestriction twist-drill (h) (diam)
    (setq diam (getval h 'diameter))
    (() diam 0.0625)
    ((( diam 2)
    ((( (getval h 'depth) 6)
    (() (getval h 'roundness) 0.004))
```

Like hole-process, twist-drill has a set of restrictions which must be satisfied if they are to be successful. In the above statement, h is a parameter which SIPS will bind to the item frame describing the feature to be created, and diam is a local variable. The statement says that twist-drill cannot be used to create h unless the diameter of h is greater than 0.0625 and less than six times the depth of h, and unless h's permissible deviation from perfect roundness exceeds 0.004.

```
(defaction twist-drill (p h) ()
    (success p))
```

If SIPS decides that a particular machining process can be used to create some feature, actions must be taken to accomplish this. When creating a plan for the creation of some feature, SIPS searches backwards from the ultimate goal to be achieved—and so the actions associated with a machining process set up subgoals to be achieved before the machining process can be performed. Since hole-process denotes a class of machining operations rather than any particular machining operation, it does not have any actions associated with it—but twist-drill does have actions, as given in the above statement. In this statement, h and p are parameters which SIPS will bind to item frames describing the hole to be created and the particular twist drilling pro-

cess to be used to create it. The statement says that the twist-drilling process p succeeds directly, without the necessity of achieving any subgoals.

```
(defarchetype rough-bore (hole-process) (
    (cost $init 3)
    (projected-cost $init (getval 'hole-process
        'cost))))
```

The only other child of hole-process is rough-bore. As described in the above statement, the initial value for rough-bore's cost slot is 3, and the initial value for the projected-cost slot is the value of hole-process's cost slot, which is 1.

```
(defrestriction rough-bore (f) ()
    () (getval f 'roundness) 0.0003))
```

This statement sets up the restrictions for rough boring. It says that that rough-bore cannot be used to create a hole f unless f's permissible deviation from perfect roundness exceeds 0.0003.

```
(defaction rough-bore (p h) (g diam)
    (setq g (copy-item h))
    (setq diam (getval h 'diameter))
    (putval g 'diameter (- diam (* 0.01 (sqrt diam))))
    (putval g 'roundness 0.01)
    (subgoal p g))
```

This statement says that if SIPS decides to use a rough boring process p to create the hole h, this can be done provided that another hole called g is created first. The hole g has a smaller diameter than h, and g need not be as round as h. In order to produce the description of g, this statement uses copy-item to make g an exact copy of the item h, and uses putval to assign values into g's diameter and roundness slots. The subgoal function tells SIPS to consider the creation of g as a subgoal to be achieved before rough-bore can be successful.

5. Control Strategy

When using SIPS, one normally represents a metal part as some collection of surface features (holes, flat surfaces, slots, etc.). Each such feature is represented by an item frame. The user invokes SIPS separately on each feature in order to produce a process plan for creating that feature.

In considering how to create some feature such as a hole, SIPS does


```
procedure make(f):
   Active := {(p, f)| p is a process archetype relevant for
                      creating whatever kind of feature is}
   /*for example, hole-process is relevant for creating a hole */
   loop
      if Active is empty, then return failure
      select t = (p1, f1, p2, f2, . . .) as described in the text
      remove t from Active
      if the restrictions for p1 are satisfied then begin
        expand t as described in the text
        if success has been achieved, then return the successful plan
        /* the plan returned is guaranteed to have the least possible cost */
   end
  repeat
end make
```

Figure 4. SIPS's Control Strategy

not consider a process such as twist-drill unless the restrictions have been satisfied for twist-drill's ancestors. This is accomplished by means of the least-cost-first Branch and Bound procedure shown in Figure 4.

SIPS uses an *active list* containing all alternative process plans being actively considered. Each plan consists of some sequence of processes, along with the features these processes create, culminating in the creation of the desired feature f. For example, if we let $t = (p_1, f_1, p_2, f_2, p_3, f)$, then t represents the following plan:

first use process p_1 to create the feature f_1,
then use p_2 to transform f_1 into f_2,
then use p_3 to transform f_2 into f.

Each of p_1, f_1, p_2, f_2, p_3, and f are represented by frames.

SIPS expands plans backwards from the ultimate goal of creating f, until a complete and successful plan for creating f is found. Since this expansion is done backwards, the processes p_2, p_3, . . . in the plan t will have already been completely determined, but the process p_1 may not yet be completely determined.

If p_1 has not been completely determined, it will be represented by an archetype (such as hole-process or mill) which represents a class containing several different kinds of processes. If the restrictions specified in the frame for p_1 are not satisfied, then the expansion of t is empty; otherwise, the expansion of t consists of the set of plans

$\{(q_1, f_1, p_2, f_2, p_3, f) | q_1$ is a child of $p_1\}$.

If p_1 has been completely determined, it will be represented by an archetype (such as `twist-drill` or `rough-face-mill`) which falls at the bottom of the hierarchy and thus only represents one kind of process. In this case, if the restrictions given in the frame for p_1 are satisfied, SIPS will create an item frame describing the particular instance of p_1 that is to be used to create f_1, and will perform the actions specified in the action statement for p_1. Normally, these actions will state one of the following:

1. that this instance of p_1 is a process that can be performed directly (in which case SIPS has found a complete and successful process plan). An example of this was given in the action statement for `twist-drill` in Section 4.
2. that some other feature f_0 must first be created in order for this instance of p_1 to be performed. (There are several ways to specify this, depending on what is known about how to create the feature—but one example is the `subgoal` statement discussed in Section 4.) In this case, the expansion of t is

   $\{(p_0, f_0, p_1, f_1, p_2, f_2, p_3, f) | p_0$ is relevant for whatever kind of feature f_0 is$\}$.

SIPS selects plans for expansion one at a time. Which plan is selected is determined by means of lower bounds on the costs of the plans. For example, the lower bound on the plan $t = (p_1, f_1, p_2, f_2, p_3, f)$ is

$$B(t) = (\text{getval } p_1 \text{ projected-cost}) + \sum_{i=1}^{3} (\text{getval } p_i \text{ cost}).$$

Two significant features of the procedure described above are:

1. Several different possible plans are explored in parallel. At each point, SIPS considers the plan that currently looks the best (i.e., the one that has the lowest lower bound). As a result, the first successful plan found by SIPS is guaranteed to be the cheapest possible successful plan.
2. A process p will never appear as part of a plan on the active list unless its restrictions have been satisfied. Thus, unless the restrictions for p are satisfied, the children of p will never be examined.

6. Related Work

Hierarchical knowledge clustering can be viewed as a way to do planning based on abstraction. For example, the `hole-process` frame discussed earlier is basically a representation of an abstract machining process which has several possible instantiations: `twist-drill` and `rough-bore`.

Several types of abstraction have been explored in the literature on planning. One type of abstraction is that used in NOAH (Sacerdoti, 1977), in which an action 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 (Nilsson, 1980), 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 more closely related to that used in SIPS, in the following senses: 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 (1986). 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 this approach has not yet been implemented on any problem.

Several systems for diagnostic problem solving make use of certain

kinds of taxonomic hierarchies. Both MDX (Mittal, Chandrasekaran, & Smith, 1979) and Centaur (Jackson, 1986) use taxonomies of various diagnostic problems, in which knowledge about each class of problems is located at the node in the hierarchy that 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.

7. Concluding Remarks

7.1 Current Status

A predecessor to SIPS was implemented using Prolog (Nau & Chang, 1985, 1986). SIPS, which is implemented in Lisp, incorporates a number of refinements and improvements—particularly in the operation of the frame system. SIPS is currently being integrated into the AMRF (Automated Manufacturing Research Facility) project at the National Bureau of Standards, where it will be used in producing process plans for an automated machine shop; and plans are under way for integrating it with software being developed at General Motors Research Laboratories.

SIPS is currently up and running in Franz Lisp. It can either read prepared data from a file, or (if some of this data are 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.

7.2 Computational Considerations

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, and suppose R is the set of rules for 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 in R match S. If the system searched through all the rules in R 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 (Fikes & Kehler, 1985), is to provide facilities whereby the user can divide a set of rules into smaller subsets

$R_1, R_2, \ldots, R_n$, such that each subset is relevant for a different problem domain.[1] 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 alleviated.

Although hierarchical knowledge clustering was developed without any knowledge of the approach used in KEE, it can be thought of as an extension of that approach. Hierarchical knowledge clustering provides a way to tell, directly from the current state R, that only some subset R_S of the rules in R is relevant to S.[2] 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 (Forgy, 1980) and YAPS (Allen, 1982). 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 its preconditions each time the current state changes. This makes the complexity of computing matches dependent not on the size of D, 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, significant computational overhead will occur in the elaboration of partial matches.

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 potentially be useful in increasing the efficiency of the rete match procedure.

7.3 Interface Considerations

For the process planning problem domain, hierarchical knowledge clustering appears to be more natural to use than a "flat" set of production rules. It yields many of the same advantages as hierarchical representation of static information. For example, in the frame representing

---

[1] By "problem domain," we simply mean some class of problems.

[2] In particular, finding R_S corresponds either to retrieving the children of some archetype or (in the case of the subgoal function) retrieving all archetypes 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.

rough face milling, one can concentrate on describing the restrictions and capabilities that distinguish it from other kinds of face milling processes, rather than having to distinguish it from every other kind of machining operation. This experience has been borne out in the experience of a manufacturing engineer who is currently implementing a SIPS knowledge base for the task of selecting cutting tools once the machining processes are known (Luce, work in progress).

An even more sophisticated interface for SIPS is currently being developed. Work done by the author and others General Motors Research Laboratories has resulted in an interface between SIPS and a solid modeling system, 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. This work will be described in more detail in a subsequent paper (Nau & Sinha, in preparation).

There are certainly some problem domains for which hierarchical knowledge clustering is not appropriate—but for some problem domains it appears to provide a way to represent knowledge more naturally than with production rules, and a way to reduce some of the computational overhead that can occur with rule-based systems.

References

Allen, E. M. (1982). *Yaps: Yet another production system* (Tech. Rep. TR-1146,). Computer Science Department, University of Maryland, College Park.

Boothroyd, G. (1975). *Fundamentals of Metal Machining and Machine Tools.* Scripta, Washington, DC.

Chang, T. C. (1980). *Interfacing CAD and CAM—A study of hole design.* M.S. thesis, Virginia Polytechnic Institute. Blacksburg, VA.

Chang, T. C., & Wysk, R. A. (1985a). *An introduction to automated process planning systems.* Englewood Cliffs, NJ: Prentice–Hall.

Chang, T. C., & Wysk, R. A. (1985). Integrating CAD and CAM through automated process planning. *International Journal of Production Research,* 22:(5).

Chang, T. C., & Wysk, R. A. (1981). An integrated CAD/automated process planning system," *AIIE Transactions,* 13(3).

Davis, R.. Buchanan, B., & Shortliffe, E. (1977). Production rules as a representation for a knowledge-based consultation program, *Artificial Intelligence,* 8(1), 15–45.

Dunn, M. S., Jr., Bertelsen, J. D., Rothauser, C. H., Strickland, W. S., & Milsop, A. C. (1981). *Implementation of computerized production process planning,* (Report R81–945220–14), United Technologies Research Center, East Hartford, CT.

Fikes, R., & Kehler, T. (1985). The role of frame-based representation in reasoning, *Communications of the ACM* 28(9), 904–920.

Forgy, C. L. (1980). *The OPS5 user's manual* (Tech. Rep. CMU–CS–81–135), Computer Science Department, Carnegie–Mellon University, Pittsburgh, PA.

Jackson, P. (1986). *Introduction to expert systems.* Wokingham, England: Addison–Wesley, pp. 142–157.

Link, C. H. (1976). *CAPP—CAM-I automated process planning system. Proceedings 13th Numerical Control Society Annual Meeting and Technical Conference,* Cincinnati.

Luce, M. work in progress.

Mann, W. S., Jr., Dunn, M. S., & Pflederer, S. J. (1977). Computerized production process planning, Report R77-942625–14, United Technologies Research Center.

Matsushima, K., Okada, N., & Sata, T. (1982). The integration of CAD and CAM by application of artificial-intelligence, *CIRP,* 329–332.

Mittal, S., Chandrasekaran, B., & Smith, J. (1979). Overview of MDX—A system for medical diagnosis, *Proceedings Third Annual Symposium on Computer Applications in Medical Care,* Washington, DC.

Nau, D. S., (1983). Issues in spatial reasoning and representation for automated process planning, *Proceedings Workshop on Spatial Knowledge Representation and Processing.*

Nau, D. S., & Chang, T. C. (1983). Prospects for process selection using artificial intelligence, *Computers in Industry* 4, 253–263.

Nau, D. S., & Chang, T. C. (1985). A knowledge-based approach to generative process planning, *Production Engineering Conference at ASME Winter Annual Meeting,* pp. 65–71, Miami Beach, FL.

Nau, D. S., & Chang, T. C. (1986). Hierarchical representation of problem-solving knowledge in a frame-based process planning system, *Journal of Intelligent Systems* 1(1), 29–44.

Nau, D. S., & Sinha, S. in preparation.

Nau, D. S., Reggia, J. A., Blanks, M. W., Peng, Y., & Sutton, D. (1984). Prospects for knowledge-based computing in automated process planning and shop control (Tech. Report), Computer Science Department, University of Maryland, College Park.

Nilsson, N. J. (1980). *Principles of artificial intelligence,* Palo Alto, CA: Tioga Press, pp. 350–354.

Ramsey, C., Reggia, J. A., Nau, D. S., & Ferrentino, A. (1986). A comparative analysis of methods for expert systems, *International Journal Man-Machine Studies.*

Tenenberg, J. (1986). *Planning with abstraction,* Proceedings *AAAI-86,* Philadelphia, pp. 76–80.

Sacerdoti, E. (1977). *A structure for plans and behavior.* New York: American Elsevier.

TNO, (1981). *Introduction to MIPLAN.* Organization for Industrial Research, Inc., Waltham, MA.

Wolfe, P. M., & Kung, H. K. (1984). Automating process planning using ar-

tificial intelligence. *Proceedings 1984 Annual International Industrial Engineering Conference*, Chicago.

Reggia, J. A., Nau, D. S., & Wang, P. Y. (1983). A new inference method for frame-based expert systems. *Proceedings Annual National Conference on Artificial Intelligence*, Washington, DC, pp. 333–337.

Wysk, R. A., (1977). *An automated process planning and selection program: APPAS*. Ph.D. thesis, Purdue University, West Lafayette, IN.