[17] Sevenler, K., Raghupathi, P.S. and Altan, T. "Forming Sequence Design For Multistage Cold Forging", *Internal Report of Battele Labs.*, courtesy of Dr. Altan, March, 1986.

[18] Newell, A. and Simon, H.A. *Human Problem Solving,* Prentice-Hall, Inc., Englewood Cliffs, NJ 07632, 1972.

[19] Eshel, G., *Automatic Generation of Process Outlines of Forming and Machining Processes,* - Ph.D. Dissertation, Purdue University, School of I.E., W. Lafayette, IN. August, 1986.

[20] Eshel, G., Barash M. and Johnson, W., "Rule Based Modeling for Planning Axisymmetrical Deep-Drawing", forthcoming, *Journal of Metalworking Technology,* Elsevier.

[21] Eshel, G., Barash M. and Chang, T.C., "A Rule-Based System for Automatic Generation of Deep-Drawing Process Outlines", Bound volume, *Computer-Aided / Intelligent Process Planning, ASME Winter Annual Meeting,* Miami-Beach, FL., Nov. 1985.

[22] Eshel, G., Barash M. and Fu, K.S., "Generating The Inclusive Test-Rule in a Rule Based System For Process Planning", Bound volume, *Computer-Aided / Intelligent Process Planning, ASME Winter Annual Meeting,* Miami-Beach, FL., Nov. 1985.

[23] Jones, F.D., *Die Design and Diemaking Practice,* Industrial Press, 1930.

Symposium on Integrated and Intelligent Manufacturing at ASME Winter Annual Meeting

SIPS: AN APPLICATION OF HIERARCHICAL KNOWLEDGE CLUSTERING TO PROCESS PLANNING

D. A. Nau* and M. Gray
Computer Science Department
University of Maryland
College Park, Maryland

This paper describes SIPS, an AI system which selects machining operations for the creation of metal parts. SIPS is a successor to SIPP, which was described at a previous ASME conference. Whereas SIPP was intended purely as a prototype, SIPS is intended to be at the core of a useable process planning system.

SIPS uses knowledge-based reasoning techniques to make process plans completely from scratch, using only the specification of the part to be produced and knowledge about the intrinsic capabilities of each manufacturing operation. SIPS incorporates a knowledge representation language written in Lisp, a best-first Branch and Bound strategy for finding process plans of least possible cost, and a new knowledge representation technique called ***hierarchical knowledge clustering.***

1 Introduction

This paper describes a generative process planning system called SIPS (Semi-Intelligent Process Selector), which uses frame-based reasoning techniques to select machining processes for the creation of metal parts using metal removal operations. SIPS uses knowledge-based reasoning techniques to make process plans completely from scratch, using only the specification of the part to be produced and knowledge about the capabilities of each manufacturing operation.

In most frame-based reasoning systems, the data manipulated by the system is represented using frames, but the problem-solving knowledge used to manipulate this data is represented as production rules. Such rules are not always a natural way to represent knowledge—and in addition, large rule bases may cause severe problems with inefficiency as the system repeatedly searches the rule base to find applicable rules. SIPS addresses these problems using a new knowledge representation technique called ***hierarchical knowledge clustering.***

SIPS is a successor to a previous system called SIPP [17,18]. Whereas SIPP was intended purely as a prototype, SIPS is intended to be at the core of a useable process planning system. It is being integrated into the process planning portion of the AMRF project at the National Bureau of Standards, and plans are underway to integrate it with systems being built at General Motors Research Laboratories. This paper 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 to Dana Nau, with matching funds from IBM Research, General Motors Research Laboratories, and Martin Marietta Laboratories; and through sabbatical arrangements with the National Bureau of Standards and General Motors Research Laboratories.

2 Background

Most commercially available computer-aided process planning systems are based on the use of Group Technology (GT) codes [4], which are used to classify a part as being in a family of similar parts. In such systems, when a process plan for a part is desired, a human user enters the GT code for the part into the system, and the system retrieves a process plan which was previously used for some other similar part. The process plan is then modified by the user to produce a plan for the desired part. Examples of such systems are CAPP [11] and MIPLAN [12].

Generative process planning systems have been developed experimentally which use as input a represention of each of the machinable features of the part.[1] Examples include CPPP [13,8], APPAS [22], CADCAM [3,6], and TIPPS [5]. Due to the difficulty of truly generative process planning, each of these systems has various limitations on the kinds of parts for which it can produce process plans.

More sophisticated approaches to generative process planning will require sophisticated representational and reasoning techniques [15,16]. The use of AI rule-based reasoning is being tried experimentally in Gari [7] and TOM [14]. Another system [21] is said to use some elementary expert system techniques, but the paper describing it does not give any further details. Each of these systems has various limitations on the kinds of process plans it can produce.

3 Knowledge Representation

In order to represent knowledge, SIPS includes a frame system which was written in Lisp. Frames are somewhat similar to Pascal record structures: Pascal records have fields which have both names and values, and frames have slots which have both names and values. However, frame systems also have various other features. For example, slot values can be inherited from "parent" frames if the values are not explicitly given, and frame systems usually allow the user to define "daemon" procedures which cause various computations to occur spontaneously whenever appropriate conditions are met.

SIPS's frame system incorporates several worthwhile advances over the frame system used in SIPP. These include more flexible and general ways of handling inheritance, and a full implementation of the hierarchical knowledge clustering idea (which was explored only experimentally in SIPP).

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) consists of items. Both items and archetypes have slots into which values can be stored and retrieved. The frame system has general inheritance mechanisms which allow slots and slot values either to be inherited in the usual way from parent to child, or to be computed as arbitrary functions of values stored in other frames (for example, see the **`$ifneeded`** function described below).

3.1 Static Knowledge

The way SIPS represents static knowledge (e.g., representations of three-dimensional objects) is similar to most other frame systems. The following example is considerably simpler that what actually appears in SIPS's knowledge base, but it will give the reader an idea how SIPS represent static knowledge. More significant is how SIPS represents problem-solving knowledge (e.g., knowledge about the capabilities of machining processes); this will be discussed in Section 3.2.

```
(defarchetype surface () (
   (surface-finish $type posnumberp
                   $init 100
                   $comment "permissible deviation from perfect smoothness")
```

[1]By a *machinable feature* we mean any geometric surface or combination of geometric surfaces which can be produced by a single machine tool operation. For example, a hole is a single machined surface, although it consists of both a cylinder and a cone. For more information on machining operations, see [2, Chapter 1].

```
   (wall-thickness $type posnumberp
                   $comment "how much deeper than the stock or casting")
   (contains $type framelistp
             $init ()
             $comment "all surfaces contained by this one (e.g. holes, slots)")
   (adjacent $init ()
             $comment "all surfaces adjacent to this one")))
```

This defines an archetype called **`surface`** which has four slots. For most (but not all) of these slots, data types have been specified using the **`$type`** keyword. For example, any value put into the **`surface-finish`** slot must be a positive number. Using the **`$init`** keyword, an initial value of 100 has been specified for the **`surface-finish`** slot. For each of the slots, the **`$comment`** keyword has been used to specify a comment which can be printed out by SIPS when the user wants to know what the slot is for.

```
(defarchetype flat-surface (surface) (
   (norm $type coord3
         $comment "unit vector perpendicular to the surface")
   (flatness $type posnumberp
             $comment "permissible deviation from perfect flatness")
   (angularity $type posnumberp
               $comment "maximum permissible angularity")
   (parallelism $type posnumberp
                $comment "permissible deviation from perfect parallelism")))
```

This defines an archetype called **`flat-surface`** which is a child of **`surface`**. This archetype inherits all four of the slot definitions given for **`surface`**, and also has four new slots.

```
(defitem f1 flat-surface (
   (norm (1 0 0))
   (flatness 0.1)
   (adjacent (f2 f3 f4 f5))))
```

This defines the item **`f1`** to be an instance of **`flat-surface`**. Values for the **`norm`**, **`flatness`**, and **`adjacent`** slots have been given explicitly, and the values for several other slots are inherited from **`flat-surface`**. For example, the value of the **`contains`** slot is the empty list, and the value of the **`surface-finish`** slot is 100.

3.2 Problem-Solving Knowledge

Most frame-based systems use frames to represent the data being manipulated, but represent problem-solving information using if-then rules. Recent work [18,20] suggests that in some problem domains (such as process planning), rules may not be the best way to represent problem-solving knowledge. For this reason, SIPS uses a frame-based representation of problem-solving knowledge called *hierarchical knowledge clustering*, in which knowledge about the capabilities of machining processes is organized into a taxonomic hierarchy. Archetype frames are used to represent classes of machining processes (such as **`twist-drill`** or **`mill`**), and item frames are used to represent specific machining processes (such as **`spade-drill-2`** or **`rough-end-mill-14`**).

The following example is much simpler than the information appearing in SIPS's knowledge base, but it illustrates how hierarchical knowledge clustering works.

```
(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 defines an archetype called `process` which has two slots, `cost` and `projected-cost`. The initial value for the `cost` slot is the minimum of the `cost` slots of the children of `process`. Similarly, the initial value for `projected-cost` is the minimum of the `projected-cost` slots of the children. (The intent of this is that the `cost` slot will contain a lower bound on the cost of performing a machining process, and the `projected-cost` slot will contain a lower bound on the cost of any other processes which might be required beforehand.)

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

This defines an archetype called `hole-process` which is a child of `process`. No slots are specified explicitly for `hole-process`, but since it is a child of `process`, it has `cost` and `projected-cost` slots whose initial values are the minimum values of these slots in the children of `hole-process`. Since the children of `hole-process` are `twist-drill` and `rough-bore` (see below), this means that `cost` gets the value 1 and `projected-cost` gets the value 0. The `relevant` statement says that `hole-process` is relevant for creating any item which is a `hole`.

The `defrestriction` statement is used to specify one or more restrictions, all of which must be satisfied before `hole-process` can be used to create a hole `h`. `h` is a parameter, `surface` is a local variable, the `setq` function assigns a value to a variable, the `get-archetype` function tells what kind of frame its argument is, the `getval` function returns the value of a slot in a frame, and the `parallel` function tests for parallelism of two vectors. Thus, the `defrestriction` statement states 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)))
(defrestriction twist-drill (h) (diam)
    (setq diam (getval h 'diameter))
    (greaterp diam 0.0625)
    (lessp diam 2)
    (lessp (getval h 'depth) 6)
    (greaterp (getval h 'roundness) 0.004))
(defaction twist-drill (p h) ()
    (success p))
```

This defines an archetype called `twist-drill` which is a child of `hole-process`. Its `cost` and `projected-cost` slots have initial values of 1 and 0, respectively. Since `twist-drill` is a child of `hole-process`, SIPS will not consider using `twist-drill` to create a hole `h` unless `h` satisfies the restrictions for `hole-process`. However, if SIPS does decide to consider `twist-drill`, the `defrestriction` 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 the roundness of `h` exceeds 0.004. If these restrictions are satisfied, then the `defaction` statement says that `twist-drill` will succeed in creating `h` without requiring that anything else be done.

```
(defarchetype rough-bore (hole-process) (
    (cost $init 3)
    (projected-cost $init (getval 'hole-process 'cost))))
(cannot-precede rough-bore twist-drill)
(defrestriction rough-bore (h) ()
    (greaterp (getval h 'roundness) 0.0003))
(defaction rough-bore (p h) (g diam)
    (setq g (copy-item h))
```

```
    (setq diam (getval h 'diameter))
    (putval g 'diameter (difference diam (times 0.01 (sqrt diam))))
    (putval g 'roundness 0.01)
    (subgoal p g))
```

This defines an archetype called `rough-bore` which is a child of `hole-process`. The `cost` and `projected-cost` slots for `rough-bore` have values of 3 and 1, respectively. `rough-bore` cannot precede `twist-drill` in any plan produced by SIPS. Since `rough-bore` is a child of `hole-process`, SIPS will not consider using `rough-bore` to create a hole `h` unless `h` satisfies the restrictions for `hole-process`. However, if SIPS does decide to consider `rough-bore`, `rough-bore` cannot be used to create `h` unless the roundness of `h` is greater than 0.0003. If SIPS decides to use `rough-bore`, `rough-bore` can create `h` provided that another hole called `g` is created first. The diameter of `g` is less than the diameter of `h`, and `g` need not be as round as `h`.

4 Problem Solving

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

When told to create a plan for some feature f, SIPS invokes a least-cost-first Branch and Bound search to produce a least-cost plan. The Branch and Bound procedure uses an *active list* containing all alternative plans being actively considered. Each plan consists of some sequence of processes, along with the features these processes create, culminating in the creation of f. For example, the sequence $t = (p_1, f_1, p_2, f_2, p_3, f)$ 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 plan for creating f is found. Since this expansion is done backwards, the processes p_2 and p_3 in t will have already been completely determined, but 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`) which represents a set of several different kinds of processes. If the restrictions for p_1 (given in the `defrestriction` statement) are not satisfied, then the expansion of t is empty; otherwise, the expansion of t consists of the set of plans

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

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

1. that p_1 can be performed directly (e.g., the `success` statement given earlier for `twist-drill`). In this case SIPS has found a successful plan.

2. that some other feature f_0 must first be created in order for p_1 to be performed. In this case, the expansion of t is

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

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. The lower bound on the plan $t = (p_1, f_1, p_2, f_2, p_3, f)$ is computed by adding the values of the `cost` slots for p_1, p_2 and p_3 to the value of p_1's `projected-cost` slot.

Two main features of SIPS's Branch and Bound procedure 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 least 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.

5 Discussion

A predecessor to SIPS was implemented using Prolog [17,18]. 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 to produce process plans for an automated machine shop; and plans are underway for integrating it with software being developed at General Motors Research Laboratories.

For the process planning problem domain, hierarchical knowledge clustering appears to be more natural to use than a "flat" set of production rules. For example, in a frame representing rough face milling, one can concentrate on describing the restrictions and capabilities that distinguish rough face milling from other kinds of face milling processes, rather than having to distinguish it from every other kind of machining operation.

Problems with inefficiency in production rule systems containing large rule bases are well-known, and several approaches have been proposed for alleviating these problems. Systems such as OPS-5 [10] and YAPS [1] provide ways to determine whether a rule is applicable without having to re-evaluate all of its preconditions each time around, and KEE [9] provides facilities whereby the user can partition a set of rules into subsets and tell the system to use just one of these subsets in solving a problem. However, if the set of rules to be used for a particular problem is large, problem-solving using production rules can still be inefficient. These issues are discussed in more detail in [19].

Suppose a rule-based system has a set of rules S. Each time the system applies a rule, this will change the system's current state C. Determining which rule is applicable would be much more efficient if the system could tell—without having to search the rule base—that some small subset S_C of the rules in S were relevant to C.

SIPS's hierarchical knowledge clustering approach is not necessarily appropriate for all problem domains, but in problem domains (such as process planning) where it is appropriate, this approach makes it very easy to determine S_C. In SIPS, finding S_C corresponds either to expanding some archetype, or (in the case of the `subgoal` statement) retrieving all archetypes relevant to the creation of a feature. This leads to more efficient problem-solving than would be achievable using an ordinary rule-based approach.

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

[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 R81-945220-14, United Technologies Research Center, East Hartford, CT, June 1981.

[9] R. E. Fikes and N. J. Nilsson, "STRIPS: a New Approach to the Application of Theorem Proving to Problem Solving," *Artificial Intelligence* 2:3/4, 1971, pp. 189-208.

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

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

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

[13] W. S. Mann, Jr., M. S. Dunn, and S. J. Pflederer, "Computerized Production Process Planning," Report R77-942625-14, United Technologies Research Center, Nov. 1977.

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

[15] D. S. Nau, "Issues in Spatial Reasoning and Representation for Automated Process Planning," *Proc. Workshop on Spatial Knowledge Representation and Processing*, May 1983.

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

[17] 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.

[18] 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.

[19] D. S. Nau and M. Gray, "Hierarchical Knowledge Clustering: A New Representation for Problem-Solving Knowledge," *in* J. Hendler, *Expert Systems: The User Interface*, Ablex, 1987, to appear.

[20] 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.

[21] P. M. Wolfe and H.K. Kung, "Automating Process Planning Using Artificial Intelligence," *Proc. 1984 Annual International Industrial Engineering Conference*, Chicago, 1984.

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