Using Goal Interactions to Guide Planning: The Program Model

Caroline Hayes

CMU-RI-TR-87-10

The Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

April 1987

This research was funded by Cincinnati Milacron and Chrysler Corporation.

Table of Contents

List of Figures

Abstract

The ***Machinist*** program uses a new planning method that is an extension to domain dependent planning technology. It is modeled after the behavior of human machinists, and makes plans for fabricating metal parts using machine tools. Many existing planning programs rely on a problem solving strategy that involves fixing problems in plans only after they occur. The result is that planning time may be wasted when a bad plan is unnecessarily generated and must be thrown out or modified. The Machinist program improves on these methods by looking for cues in the problem specification that may indicate potential difficulties or conflicting goal interactions, before generating any plans. It plans around those difficulties, greatly increasing the probability of producing a good plan on the first try. Planning efficiency is greatly increased when false starts can be eliminated. The Machinist program contains about 180 OPS5 rules, and has been judged by experienced machinists to make plans that are, on the average, better than those of a 5 year journeyman. The knowledge that makes the technique effective is domain dependent, but the technique itself can be used in other domains.

1 Introduction

Machinist is a planning program that works on machining problems, and produces feasible plans for manufacturing individual metal parts. ***Machining*** is the art of producing metal parts using a variety of power tools to shape the metal. It is a highly skilled task requiring 10 to 15 years to become fairly accomplished.

The program works by first scanning the problem specification (a set of shapes to be cut in a metal block, and some information on raw material, dimensions, etc.) for cues or patterns that indicate potential problems. It also looks for other types of patterns that provide salient information: what set of tools can be used for specific cuts, and what processes, as well as information on the details and restrictions on those processes. Using this information as the building blocks, the program constructs a plan for producing the part.

This approach is more efficient than traditional planning methods for domains that have many interactions between the goals. Traditional planners typically work by first generating a plan, then using "critics" to scan the resulting plan for problems and correct them [15, 13, 6]. Alternatively, they may generate a plan for each subgoal, and then try to merge the subplans together [17, 11]. This is the inverse of the order in which **Machinist** does these steps. When a planner can anticipate problems, much less time is spent fixing bad plans, or trying to merge incompatible subplans.

The ideas for **Machinist's** planning technique are taken from observations of the behavior of human machinists. Protocol analysis was used to collect this information. The resulting program consists of about 180 OPS5 rules, and it runs on a DEC-20, a UNIX VAX, and a SUN workstation.

The main emphasis of this paper is to explain the program's planning methods and to examine how these methods can be used in other domains. The way in which this planning technique is implemented is domain dependent: the ability to identify a goal interaction efficiently just by looking at a specification requires intimate knowledge about that type of problem. This knowledge, in the form of patterns which identify interactions, together with operators that tell how to avoid the interactions, takes many years for the expert to build up and years for the knowledge engineer to extract. As used here, a pattern together with an associated composite operator will be referred to a ***macro-operator***. Unfortunately, the planner *must* have these macro-operators to find these interactions in complex domains, otherwise the search would be tremendous. This does not lend hope for domain independent planners to be successful in large domains, but perhaps we must reconcile ourselves to the fact that efficiency may require expertise [15].

2 The Problem, and Motivation

This research is part of the overall effort to automate the "job shop." Job shops are small machine shops that do custom work. The machinists who run these shops tend to be designers, machinists, and machine tool operators all rolled into one. They will design and make just one or a few copies of a part for a customer. Like any custom work, this is an expensive process. Automation can make custom work more affordable by eliminating the need for large amounts of experienced labor.

This research is aimed at small batch production rather than large, which makes up the bulk of manufacturing in the United States, because automated planning has more economic impact on small batch production. Planning costs are a small part of the cost in large batch production: it is a one time

cost spread out over many pieces. Manufacturers can afford to hire a human to do the planning and let him take as long as he wants. On the other hand, in small batch production (the job shop), planning is a much larger part of the cost. If only one piece is to be made the plan for manufacturing it will be used once and thrown away. The cost of hiring a highly skilled person to do the planning is often prohibitively expensive.

Job shops are essential to industries like robotics and aerospace that make one of a kind products like clean-up vehicles for Three Mile Island, and space shuttles. Additionally, job shops are needed during the design and testing of new products for producing prototypes.

Machinists have a huge body of knowledge that allows them to make plans for many different types of parts which are produced on many types of machines. However, for the purposes of this study:

- manufacturing plans will be designed for only 1 part at a time (This is not a scheduling task for a whole factory like ISIS [7])
- parts will all be produced completely on one machine tool: the Cincinnati-Milacron 5-VC Machining Center which is a three axis vertical machine tool,
- parts will be made from prismatic blocks of material,
- one type of clamp will be used: vices,
- tools will be limited to a set typically found in a job shop,
- material will be limited to cold rolled aluminum, and steel.

3 Machining Background

Since machining is not familiar to everyone, some background may be helpful in understanding the task. ***Machining*** refers to the art of creating parts, usually metal, by carving with power tools such as bandsaws, lathes, and drill presses; and using processes such as drilling, milling, turning, etc. Computer Numeric Controlled (***CNC***) machining is machining using computer controlled tools. This particular problem solving task involves making a plan for a part that can be produced completely on one particular machine, the 5-VC Machining Center, using only a few processes: milling, drilling, boring, reaming, and tapping.

3.1 The Equipment

The 5-VC Machining Center looks much like a large, fancy drill press that is computer controlled, with a movable table, a mechanical arm (for changing tools), and a rotating magazine of tools. All the moving parts can be controlled with a program. The work piece can be clamped to the table, and then the table can be moved from side to side, and frontwards and backwards, so that any part of the piece can be positioned under the cutting tool. The machine tool has a rotating ***spindle*** which can hold different tools in much the same way a drill can hold different drill bits. The spindle can move up and down to lower the tool into the work piece, much as a drill press does.

There are many types of tools that can be put in the spindle: a variety of drills, countersinks, mills, reamers, boring bits, and taps. Mills are used for shaving the sides of things or for making flat-bottomed holes. Boring bits are used for making large holes, reamers for enlarging holes, and taps for putting helical threads inside holes for screws.

In this study, one type of ***fixture*** or ***clamp*** will be used to fix work to the table: the vise. The vise is much like one that you would find in a home workshop, except that it is larger and more accurate. A ***set-up*** is a whole clamping system, comprised of a clamp, a work piece positioned in a clamp, a method for locating the piece, and sometimes some metal plates or small bars known as ***parallels*** that help position the piece. *Set-up* actually has two meanings: it can mean either a particular clamping system, or the sequence of cuts made while using the clamping system.

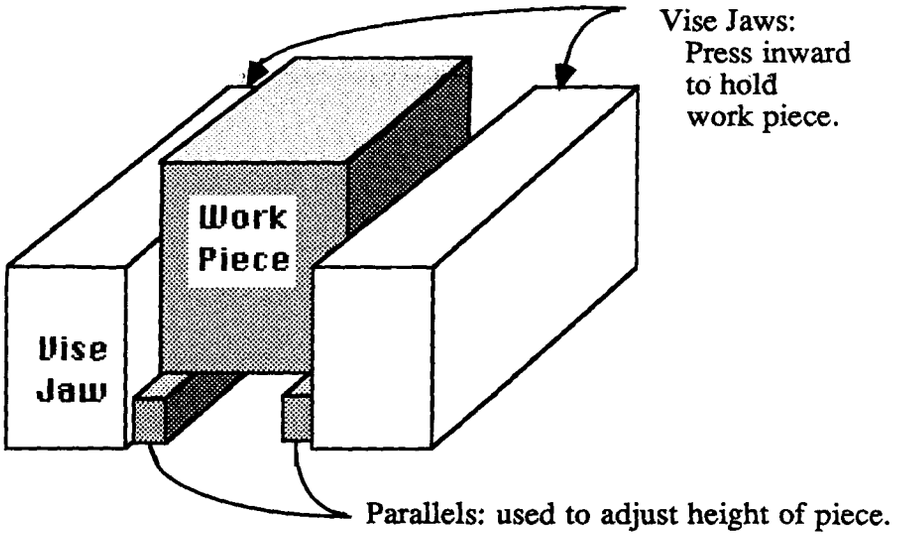


Figure 1: A ***Set-up*** using the vise and two parallels

The design specification that a machinist receives consists of a set of ***features*** which must be cut into a block in order to produce a part. The block that they start with is known as the ***stock*** and it can be made out of a variety of materials such as: aluminum, steel, copper, or plastic. For this application, the stock will always be rectangular. ***Features*** are the individual geometric shapes that are cut into a block of metal to form the part.

The ***Surface finish*** on a particular side of the stock can be either smooth or rough. The rough sides are known as ***saw-cut*** sides, and the smooth sides are known as ***rolled*** sides. Rolled sides are created as a result of the manufacturing process by which stock is made: cold steel is rolled out between rollers. The sides created by the roller are the rolled sides.

4 Interactions Make the Planning Problem Difficult

A major problem that the machinist confronts in planning is interactions between the different features that are cut into the part. Cutting one feature first may make it difficult or impossible to cut subsequent ones. One can view the collection of features as subgoals to be achieved in the machining plan. The difficulty in making a plan is to find an order in which none of the subgoals interferes too seriously with achieving the others.

This type of problem is not isolated to the machining domain; interactions between subgoals have been observed in many planning domains by many researchers: Stefik [14], Hammond [8], Sussman [15], Tate [16], and Carbonell [3] to name a few. Sussman noted it as early as 1973 in HACKER: "interactions

between steps, (are) a common cause of bugs."[1] Stefik perhaps, expressed it best: "In planning problems, there are typically many goals to be achieved in some order. The goals interact with each other in many ways which depend both on the order in which they are achieved and on the particular operators which are used to achieve them."[2] A ***feature interaction*** happens when cutting one collection of features affects the way in which others can be made. Subsequent features may become totally impossible to produce, or severely restricted in the methods by which they can be produced,

Feature interactions have several different causes. Most commonly they result from clamping problems; producing one feature destroys the clamping surfaces needed to grip the piece while cutting another feature. The clamps used in all the examples in this paper will be vises.

A feature interaction is shown below in figure 2. This part has two features: an angle, and a hole. The angle has been cut, and the hole is about to be drilled, but when the drill touches the angled surface, it will slip sideways and cause the hole to be placed inaccurately. The angle can be said to interact with the hole. The solution is to drill the hole first while the end of the part is still flat. Since the hole does not affect how the angle is made, a simple reordering again prevents the features from interacting.

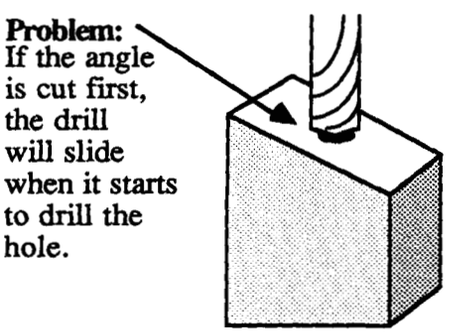


Figure 2: ***Feature Interaction***: the hole must be cut before the angle

Feature interactions can result in restrictions on either the process used to execute a step, or the order of the steps. In this paper, we have chosen examples which only put restrictions on the order of the steps.

5 How the Program Works

The **Machinist** program is modeled after the human's planning process but it only implements a part of that process. The human's planning process is described by Hayes [10]. The most important omission is that there is no verification phase at the end of the program's planning.

To demonstrate how the program works, let us suppose one wanted to make the following part from the block of metal shown below. There are five features that need to be cut into this part: three holes, an angle, and a shoulder (a shoulder is any ledge-like shape cut out of a side). The part is represented in the program as a rectangular block from which features are subtracted. The block of metal from which it

---

[1]Gerald J. Sussman, *A Computer Model of Skill Acquisition*, American Elsevier Publishing Company, New York, 1975, MIT AI Technical Report TR-297, August 1973, p. 119

[2]Mark Stefik, "Planning and Meta-Planning (MOLGEN: Part 2)," *Artificial Intelligence*, vol 16, no. 2., 1981, p. 141.

will be made, the ***stock***, is saw-cut and irregular on all sides (this fact will be important to planning decisions made later).

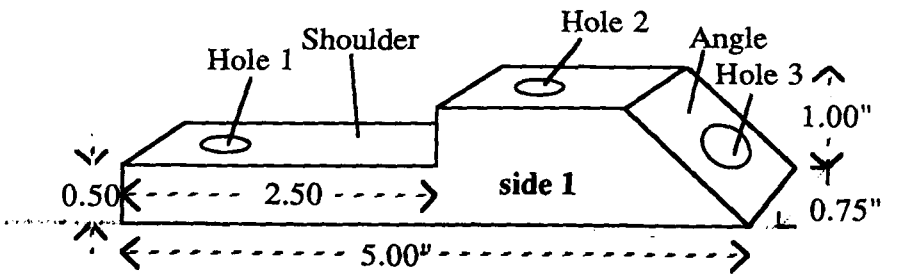


Figure 3: A ***part*** with 5 features: three holes, a shoulder, and an angle

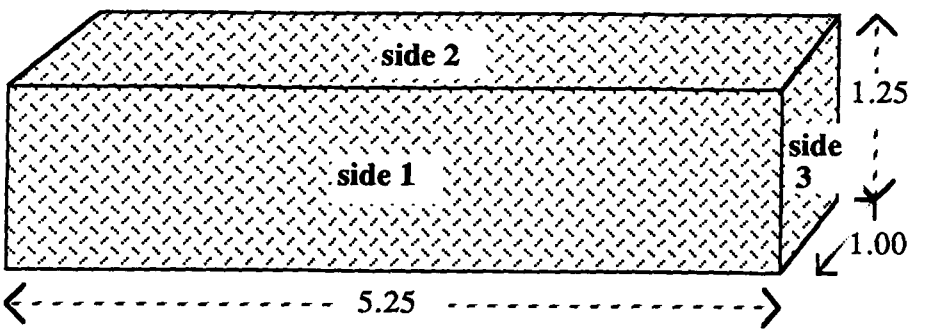


Figure 4: The ***stock*** that the part will be made from: saw-cut on all sides

5.1 Representing the Part Specification

Before describing how the program works, I will briefly describe how the part specification is represented. The outer envelope of the part is represented as a prismatic solid from which the feature volumes are subtracted. There are 8 feature types used in this system: blind-hole, thru-hole, pocket, blind-slot, thru-slot, angle, shoulder, and channel. All sides on the finished part are assumed to be machined.

The stock is represented as a prismatic solid slightly larger than the final part. Each side of the stock is assigned a surface finish of either: rolled (smooth) or saw-cut (rough). Volumes are subtracted from the stock until it looks the same as the part specification.

Both the part and the stock make use of a three dimensional coordinate system that is set up by assigning numbers to the sides of the part. Since the sides rotate as the part is rotated, the coordinate system moves with the part and there is no need for matrix transformations. The two largest area sides are given the numbers 1 and 4, the second largest sides: 2 and 5, and the smallest area sides: 3 and 6. If the part is a cube, then any pair of sides can be chosen to be the "largest." If side 1 faces up, and side 2 faces right, then the front is assigned to be side 3, thus establishing an left and right frame of reference to the part.

All features are described and referenced in terms of the side labeling system. For instance, in the part shown in figure 3 on page 5, Hole 2 is 3 inches from side 3, .375 inches from side 1, with a diameter of .31, and it goes all the way through the piece.

This feature, Hole 2, would be described in the following way:

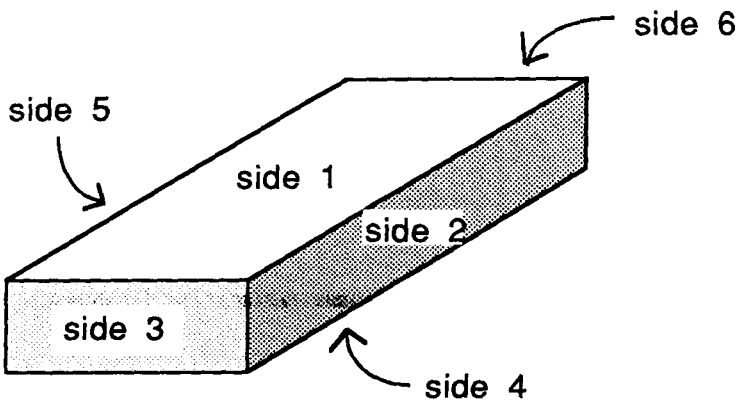


Figure 5: The side labeling conventions for a prismatic shape

```
(feature ^type thru_hole
         ^name Hole_2)
```
Hole 2 goes all the way through the piece.

```
(Open_on ^feature Hole_2
         ^side    1)
(Open_on ^feature Hole_2
         ^side    4)
```
Hole 2 opens on to side 2 on the top and side 5 on the bottom.

```
(Diameter ^of Hole_2
          ^is .31)
```
The diameter is .31 inches

```
(Center_dist ^of Hole_2
             ^from side_3
             ^is 3.000)
(Center_dist ^of Hole_2
             ^from side_1
             ^is .375)
```
The center of Hole 2 is 3 inches from side 3

The center distance from side 1 is .375 inches

In the current implementation, no tolerances (the amount of play allowed in a dimension) are specified. Tolerances are simply assumed to be high (little play allowed), and consequently the machining methods recommended by the program are very conservative. These methods insure that tolerances on the part will be as high as it is possible to get in a single roughing and finishing pass on the 5-VC. If a method works for high tolerance parts, it will also work for low tolerance parts, so the planner is capable of producing working plans with a wide variety of tolerances. However, if tolerances could be specified, then the planner could take advantage of this knowledge to make the plans for low tolerance parts more efficient. Plans for low tolerance parts are more flexible; the machinist can sacrifice some of the plan's accuracy in exchange for fewer steps.

5.2 Getting Oriented

In this phase, the program reads the problem specification, and checks for any major problems: "Does the part fit between the jaws of vise?" "Is the part so long and thin in certain directions that it will bend when clamps are applied to it?" and so on.

This step is modeled directly on the initial problem solving behavior of the expert. Both the human and the program use this phase to orient themselves into the basic structure and difficulties of the problem.

5.3 Exploring and Modeling Feature Interactions

The program's activities in this phase again mirror those in the human's behavior. The program, like the human, has a pre-planning phase in which it looks at the schematic for cues to potential problems and feature interactions.

In this part there are three interactions to be noticed: one interaction occurs between Hole 3 and the angle. If the angle is made first, it will interact with the hole by causing the drill bit to slip on the slanted surface. This will make the hole placement inaccurate, as shown in figure 2 on page 4. The restriction that this interaction puts on the plan is that Hole 3 must be made before the angle. The second interaction is between Hole 3 and the shoulder: the hole must be made before the shoulder. If the shoulder is made first, the part will be too thin and floppy when it is clamped, to cut the hole. The result of the third interaction is that the angle must be made before the shoulder, for similar reasons.

The program uses built-in generalized patterns to spot these interactions quickly. For instance, to spot the hole-angle interaction the program has a pattern that matches any drilled hole or depression that enters a non-flat surface. If the pattern is matched, then it puts a restriction on the plan, that the hole must be cut before the depression. Similarly, for each additional pattern, there is an associated operator that tells the program how to avoid the interaction. In this case, the way to avoid it was to cut the hole first. The pattern, together with the resulting plan restriction, is loosely referred to as a macro-operator. These operators are a large part of the knowledge that makes the expert an expert: they are the compiled result of his many years of experience, and they are immensely important to efficiently generating good plans. This strategy is similar to the use of chess board patterns to index playing plans in chess programs like PARADISE, and CHUNKER [19, 2]).

The three interactions, Hole 3 before Angle, Hole 3 before Shoulder, and Angle before Shoulder, all restrict the order in which the features can be cut. They can be put together into one ***interaction graph*** (figure 6). Each arrow represents one interaction. Note that Holes 1 and 2 are included in the same step as the shoulder. Since there are no interactions between Hole 1 or Hole 2 and any of the other features, it might seem that the holes could be cut anywhere in the sequence. However, it is most efficient to drill them at the same time that the shoulder is cut, so that is where the program puts them. It treats the group, Shoulder, Hole1 and Hole2, as one unit for planning purposes.

5.4 Choosing a Squaring Graph

The next task is to retrieve a ***squaring graph*** from memory. A squaring graph outlines all methods for getting the raw material into a square and accurate shape with the minimum waste of material. It represents the constraints on the order in which each of the sides may be "squared off." It serves as a framework from which the feature constraints can be hung.

Since there are only a limited number of ways in which a block can be ***squared-up*** and still maintain high accuracy, the majority of stock shapes and sizes can be ***squared-up*** using only nine squaring graphs. (This is not true for low accuracy parts, since there is more flexibility in the way they can be squared-up, and hence more graphs; but we will not concern ourselves with that since our problem only involves high accuracy parts.)

For any given piece of stock, the program chooses one of the nine squaring graphs, based on the stock's size, shape, and surface finishes. The squaring plan chosen by the program for this example is

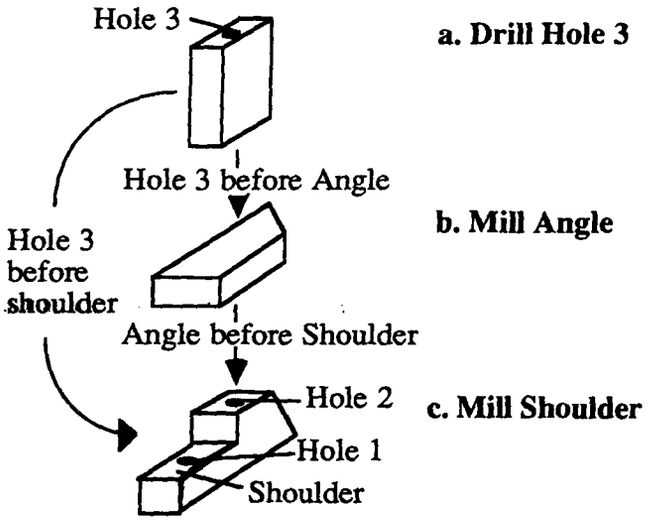


Figure 6: ***Interaction Graph***: the order in which the features may be cut

shown in figure 7. This particular plan was chosen because of the stock's length, and because all sides of the stock were saw cut. (This stock was shown on page 5 in figure 4). This plan takes advantage of the length, and makes extra allowances for the saw cut sides. Details on how squaring graphs are chosen, and what the nine graphs look like, are discussed by Hayes [10].

In each step of the plan shown in the figure, the shaded surface will be machined smooth. Steps shown side by side as branches in the graph can be done in either order. It does not matter which side of a branch is done first.

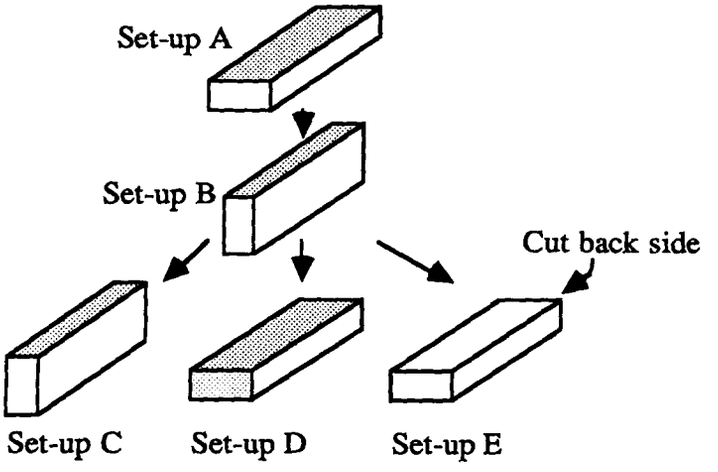


Figure 7: The ***Squaring Graph*** for squaring up a block that is sawn on all sides

A squaring graph could also be represented as a set of rules. However, for this problem it is more convenient to use the graph representation since all choices in the step sequence are laid out at one time, and alternative plans can be considered more easily. The squaring rules represented by this graph are:

- machine a largest area side first,
- machine a medium area side second,

- the remaining sides can be machined in any order.
- all sides must be machined eventually.

This graph method has a number of advantages, one is that it is easy to merge the interaction graph with the squaring graph, when the two are both represented as graphs rather then one as a graph and the other as rules. Second, this approach made it easy to plan efficiently because the squaring graph lays out all step choices at the same time for the planner making it easier to consider alternative plans. Third, there are only nine different squaring graphs that cover most stock types, so it is quite easy to store all of them. It is much more convenient to retrieve one of these stored graphs from memory then it is to generate each from scratch every time it is needed.

5.5 Integrating the Feature Interaction Graph with the Squaring Graph

We now have a graph showing the orders in which the features can be produced, and a graph showing the orders in which the sides may be cut. Each graph represents a separate set of constraints on the plan. The two must be merged with as much overlap between the steps as possible, so that we get a compact sequence. The more overlap the better, because the plan will be more concise.

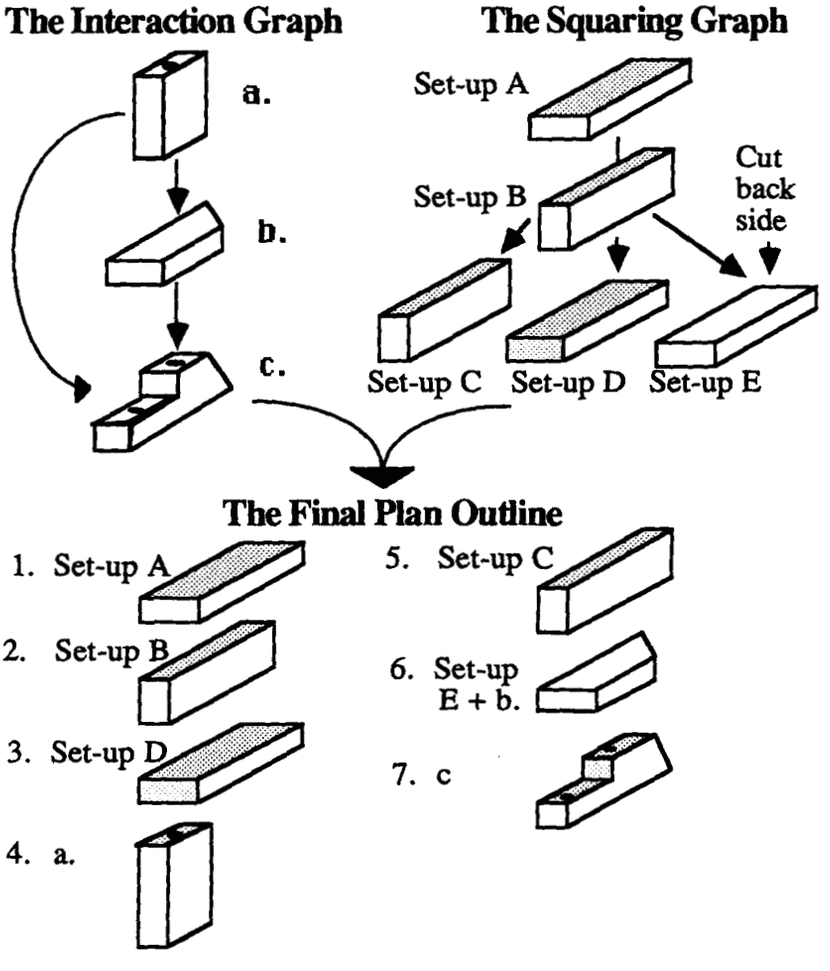


Figure 8: Merging the ***Interaction Graph*** with the ***Squaring Graph*** to produce a ***Plan OutLine***

The merging is shown in figure 8. Observe that between the Interaction Graph and the Squaring Graph there are 8 steps, but in the final plan there are only 7. This is because we were able to combine step b from the Interaction Graph with Set-up E from the Squaring Graph. The details on the process by which the two graphs are merged is described in Hayes and Wright [9].

5.6 Elaborating the Plan

In the final phase, after the program has decided on an ordering for the steps, it prints each of them out in some detail, listing the operation type (mill, face mill, drill, etc.) and the orientation of the block in the vice. The program's result is shown in figure 9. The steps (set-ups) are exactly the same as those listed in the final plan outline in figure 8 on page 9.

After producing the plan, the program does not go through the final verification phase that the human goes through. If all problems and goal interactions have been properly identified, the plan *will* be correct and the verification step unnecessary.

However, the program would obviously be more robust if it used a verification step as the human does. It is not always possible to identify all problems beforehand: neither the machinist nor the program can have a complete set of patterns to identify absolutely all possible problems and goal conflicts. Therefore, the plans produced will not always be good the first time: There needs to be some sort of a safety net to catch problems that initially escape notice. Human machinists also use a "critic" approach, to check the final plan for errors. They may reorder steps, or back up and replan to fix them. Future versions of the ***Machinist*** program will also be able to do this.

6 Discussion

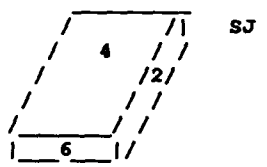
Out of the 180 productions that comprise this system:

- 10 productions identify features interactions and construct the feature interaction graph,
- 39 identify other problems and generate constraints not caused by interactions,
- 13 choose the squaring graph,
- 44 merge the interaction graph with the squaring graph,
- 11 generate the final plan from the merged constraint graphs.
- 63 other: enter and check data, infer missing data, group features, push and pop goals, etc.

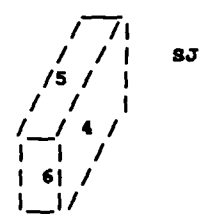
The first two categories which involve generating constraints, are the ones that have the most room to grow. Productions can be added to these two categories, greatly increasing the range of parts that the system can handle, while the rest of the system remains the same.

How much do the heuristics implemented by these rules cut down the search space? There are several categories of heuristics used by the program: feature interactions, squaring graphs, and graph merging. If the total effect of all the heuristics is taken together, we find that they speed up search by a minimum factor of 1,663,200, for the example part used in this paper. There are 5 features and 6 sides to be machined, so there are 11! (or 39,916,800) different ways of ordering these 11 things to be cut. (This is not even taking into account the different ways of grouping the cuts within the plan.) Out of all those plans, there are only 12 that are both correct and efficient. That is one good plan out of every 3,326,400, so on the average on would have to try about half that many plans, 1,663,200 to find a good one.

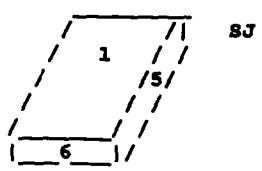
```
Set-Up  1
  - Put side  1 DOWN
  - Put side  2 ON_SOLID_JAW
  -  FACE_MILL 4
```



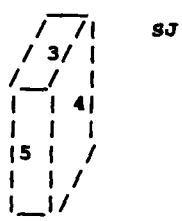

```
------------------------

Set-Up  2
  - Put side  4 ON_SOLID_JAW
  - Put side  2 DOWN
  -  FACE_MILL 5
```



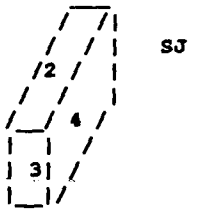

```
------------------------

Set-Up  3
  - Put side  4 DOWN
  - Put side  5 ON_SOLID_JAW
  -  END_MILL 6
  -  FACE_MILL 1
```



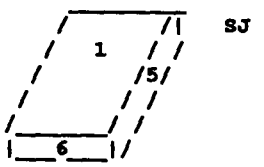

```
------------------------
Set-Up  4
  - Put side  4 ON_SOLID_JAW
  - Put side  6 DOWN
  -  FACE_MILL 3
  -  DRILL H3
```



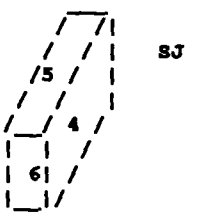

```
Set-Up  5
  - Put side  4 ON_SOLID_JAW
  - Put side  5 DOWN
  -  FACE_MILL 2
```




```
------------------------

Set-Up  6
  - Put side  4 DOWN
  - Put side  5 ON_SOLID_JAW
  -  MILL A1
```




```
------------------------

Set-Up  7
  - Put side  4 ON_SOLID_JAW
  - Put side  2 DOWN
  -  MILL S1
  -  DRILL H1
  -  DRILL H2
```




```
------------------------
```

Figure 9: A Plan Produced by the Program

Let us now consider only the feature interaction heuristic. For the example part used in this paper there are 5 features but only 3 interactions. If one uses no heuristic to order the 5 features there are 5! = 120 ways of ordering them. However there are only 6 ways to order the features so that the plan is both feasible and efficient: first drill the hole, second cut the angle, and last cut the shoulder and the two holes in any order. If there are 120 orders and only 6 good orders, then there is one good order for every 20 tried. On the average one can expect to try about 10 orders before finding one good one. The program will produce a good one on the first try, so for this example, the feature interaction heuristic cut down the search by a factor of 10.

If we look at a more complicated example taken from Hayes [10] that has 14 features and 5 interactions, there are 14! = 87,178,281,200 possible ways of ordering the features. Out of these 87 million orders there are only 69 thousand good ones. That's one good order for every 1,260,000. On the average one can expect to try 630,000 feature orderings before finding a good one. Again, the machinist's feature interaction heuristics lead him to a good ordering in one try. (Provided that he does not miss any interactions.) For this second example the heuristic cut down the search by a factor of 630,000.

Essentially, the more features and the more interactions there are the more difficult it is to find a good plan. The problem is not that the search space gets larger as more interactions are added, it is that the density of good solutions in that space goes down. The machinist's knowledge of feature interactions helps him to zero in on only those good solutions.

The squaring graph heuristic always improves the search about the same amount, regardless of the part; but the feature interaction heuristic helps the most when there are many features and several interactions.

7 Comparison of the Program Against Human Performance.

The program was tested against four machinists at various experience levels: two second year apprentices, one third year apprentice, and one journeyman with 5 years experience including the apprenticeship. Each of these subjects was asked to create a machining plan for the same series of three parts. Each part was apparently simple, but contained difficulties when examined more closely.

Their resulting plans were judged by two very experienced machinists, each having more than 15 years experience. The average ratings given to each of the four subjects and the program are shown in figure 10. The program's average performance was better than that of the apprentices or the journeyman. In fact, Machinist 1 declared the program's plan for Part III to be "Almost the perfect plan. Who ever did this is a man after my own heart."

The program solved problems in times comparable to the machinists'. The program took about 12 to 15 minutes per problem on a moderately loaded DEC 20, or 3.5 minutes on a SUN work station, while the expert machinists took about 10 to 12 minutes, and the apprentices about 20 minutes per problem.

The judging was done in the following way: for each of the three parts there were five plans generated, one from each of the four young machinists, and one from the program. All information indicating who (or what) created the plan was removed, and the the plans were presented to the two experienced machinists. Independently, they ordered each set of five plans, rating them from best to worst. The best

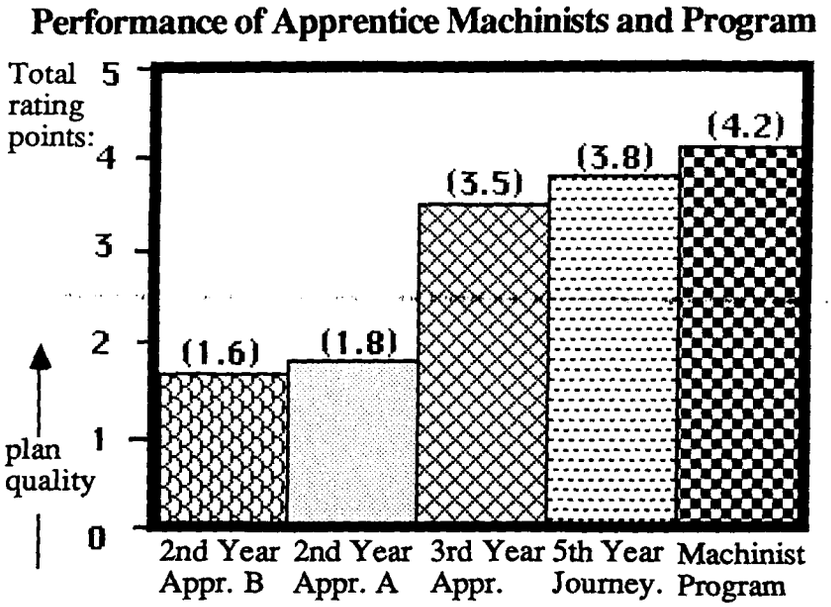


Figure 10: Average Plan Rating for Each Subject

plans were given a score of 5 and the worst 1. The sums of all scores earned by each apprentice machinist (or machine) are shown in the histogram in figure 10. The numbers written above the bars are the sums of all scores earned for all plans made by one subject.

The machinists commented on a variety of criteria they used for judging the plans. Was the plan efficient (i.e., how many set-ups), were there any bad practices used that might lessen the accuracy of the final product, and were there any mistakes that would make the plan unworkable? Furthermore, different mistakes had different degrees of seriousness. A plan with three small errors might still be rated higher than a plan with one big error. Plans that would not work were always rated lower than plans that did work.

Neither machinist felt that the other was wrong in his ratings (except for the one error that Machinist 2 missed). Both felt that the plans which they rated differently were actually very close in quality and that it was difficult to decide which was better.

In judging what this comparison means, it is important to keep in mind that the program only solves problems in a very narrow domain, but it can solve them very well. In contrast, the apprentices do not solve problems as well, but they have a much broader scope of problems they can solve. The breadth of the program's knowledge can, however, be increased by adding more knowledge to its existing framework until its breadth approaches that of the apprentices.

8 Past Programs Dealing with Interacting Plans

Many pieces of this planning process have been described before but not as one cohesive method. Virtually all of the planners referenced in this paper recognize the importance of goal interactions in planning, but their method of dealing with this problem is different than **Machinist's.** Typically they do not foresee problems in the problem specification and avoid them. Instead they make plans with mistakes in them and use critics to recognize and correct them after the fact. Time is wasted fixing and replanning.

Hacker [15] and Noah [13] are both examples of planners using critics.

TWEAK [4], GARI [6], and HI-MAPP [1], all work by successively adding constraints to the description of the solution. The interaction and squaring graphs used by ***MachInist*** are also constraints, but ***MachInist's*** advance over this approach is to obtain the constraints as the result of feature interactions.

A number of chess strategy planners use macro-operators. They use patterns associated with plans to make search more efficient. Interestingly, many of them have been modeled, at least indirectly, from human behavior. Wilkins [19], Pitrat [12], Berliner and Campbell [2], and De. Groot [5] all use some variant of this method but none of them seem to consider the effect of goal interactions on planning.

There are only a few programs that take goal interactions into account *before* attempting a plan. One of the earliest, Tate's [16] planner for house construction, does take interactions into account before it makes a plan. However, these interactions must be entered by a human since the planner itself cannot determine what tasks interact. A new set of interactions must be entered for each new task. This type of solution is not practical for machining problems since each new problem contains a different set of interactions. One cannot reuse the same set of interactions over and over again for a large class of problems.

Wilenski's planner, PANDORA [18], and to some extent Wilkins planner, SIPE [20] specifically look for goal interactions before planning (which is a great advance in domain independent planning). However, since it is domain independent, it can not make use of domain knowledge (in the form of patterns) to help identify goal interactions quickly and to find a way around them. Consequently, its performance on complex tasks such machining problems would be impractically slow.

Chef [8] is a planner that generates recipes for Chinese cooking. It is one of the few planners that looks at the problem description for cues to potential problems and interactions, and plans so as to avoid them. However, it does not use the interaction information to generate the plan as **MachInist** does but only to retrieve and modify plans. This is a good approach for many problems but it will not do for machining. Small differences in the shape or size of a part may make big differences in the plan so it is not good enough to index a past plan for a part that looks similar, and modify it. The plans may have so little similarity that it is easier to construct a new plan from scratch.

9 Conclusion

The difference between **MachInist** and other planners is that it has *all* of the following properties *together*:

1. a pre-planning step in which it scans the problem specification for signs of possible goal interactions,
2. macro-operators to identify goal interactions, and to suggest ways to restrict the plan so as to avoid them,
3. a plan constructed from collected information rather then a plan that is indexed from memory and modified.

In particular, the pre-planning identification of problem areas can greatly increase planning efficiency within a particular domain. The macro-operators that identify problem areas and suggest solutions are the key to planning efficiency. The set of operators used must be domain dependent, but the general

strategy can be applied to other domains.

10 Acknowledgements

I would like to extend a special thanks to Jim Dillinger, Dan McKeel, Ken Pander, Steve Klim, Dave Belotti, and Phyllis Huckestein for contributing their machining expertise to this project; to Paul Wright, Jaime Carbonell, Herb Simon, Irene Skupneiwicz, Paul Englert, Brack Hazen, Gregg Lebovitz, Barbara Wright, Mark Perlin for their advice, comments, and contributions; to Ken Mohnkern for the art work; and to Nancy Serviou for the technical editing. This research was funded by Cincinnati Milacron and Chrysler.

References

[1] Berenji, H. R.; B. Khoshnevis.
Use of Artificial Intelligence in Automated Process Planning.
Computers in Mechanical Engineering :47-55, September, 1986.

[2] Berliner, Hans; Murray Campbell.
Using Chunking to Solve Chess Pawn Endgames.
Technical Report CMU-CS-83-122, Carnegie Mellon University, April, 1983.

[3] Carbonell, Jaime G.
Subjective Understanding, Computer Models of Belief Systems.
UMI Research Press, Ann Arbor, Michigan, 1981.
PHD Thesis, Yale University, 1979.

[4] Chapman, David.
Planning for Conjunctive Goals.
Technical Report 802, Massachusetts Institute of Technology, May, 1985.

[5] De Groot, A. D.
Thought and Choice in Chess.
Mouton & Co., The Hauge, Netherlands, 1965.

[6] Descotte, Y., J. Latombe.
GARI: A Problem Solver that Plans how to Machine Mechanical Parts.
Proceedings of IJCAI :766-772, 1981.

[7] Fox, M. S., B. Allen, S. Smith, G. Strohm.
ISIS: A constraint-Directed Search Approach to Job Shop Scheduling.
Proceedings of the IEEE Computer Society Conference on Trends and Applications :19-30, 1983.

[8] Hammond, Kristian J.
CHEF: A Model fo Case-Based Planning.
AAAI-86 :267-271, 1986.

[9] Hayes, C. C.; P. K. Wright.
Automated Planning in the Machining Domain.
The Winter Annual Meeting of the American Society of Mechanical Engineers 24:221-232, 1986.

[10] Hayes, Caroline C.
Planning in the Machining Domain: Using Goal Interactions to Guide Search.
Master's thesis, Mellon College of Science, Carnegie Mellon University, April, 1987.

[11] Hayes-Roth, B.; F. Hayes-Roth.
A Cognitive Model of Planning.
Cognitive Science 3 :275-310, 1979.

[12] Pitrat, Jacques.
A Chess Combination Program Which Uses Plans.
Artificial Intelligence 8 :275-321, 1977.

[13] Sacerdoti, Earl D.
The Nonlinear Nature of Plans.
IJCAI4 :206-214, 1975.

[14] Stefik, Mark.
Planning and Meta-Planning (MOLGEN: Part 2).
Artificial Intelligence 16(2):141-170, 1981.

[15] Sussman, Gerald J.
A Computer Model of Skill Acquisition.
American Elsevier Publishing Company, New York, 1975.
MIT AI Technical Report TR-297, August 1973.

[16] Tate, Austin.
Project Planning Using a Hierarchic Non-Linear Planner.
Technical Report D.A.I. Research Report No. 25, University of Edinburgh, University of Edinburgh, August, 1976.

[17] Vere, Steven A.
Splicing Plans to Achieve Misordered Goals.
IJCAI :1016-1021, 85.

[18] Wilensky, Robert.
Meta-planning.
AAAI :334-336, 1980.

[19] Wilkins, David.
Using Plans in Chess.
IJCAI :960-967, 1979.

[20] Wilkins, David E.
Domain-independent Planning: Representation and Plan Generation.
Artificial Intelligence :269-301, 1984.