Artificial Intelligence for Engineering Design, Analysis and Manufacturing (1997), *11*, 219–235. Printed in the USA.

Optimized process planning by generative simulated annealing

K.N. BROWN[1] AND J. CAGAN[2]

[1]Department of Computing Science, University of Aberdeen, Aberdeen AB24 3UE, U.K.
[2]Department of Mechanical Engineering, Carnegie Mellon University, Pittsburgh, PA 15213, U.S.A.

(RECEIVED October 28, 1996; ACCEPTED April 3, 1997)

Abstract

Manufacturing process planning is a difficult problem with a prohibitively large search space. It is normally tackled by decomposing goal objects into features, and then sequencing features to obtain a plan. This paper investigates an alternative approach. The capabilities of a manufacturing process are represented by a formal language of shape, in which sentences correspond to manufacturable objects. The language is interpreted to describe process plans corresponding to the shape generation, complete with cost estimates. A macro layer that describes single operations of the machine is implemented on top of the formal language. The space it describes is searched by the generative simulated annealing algorithm, a stochastic search technique based on simulated annealing. Plans that are close to the optimum are generated in reasonable time.

Keywords: Process Planning; Simulated Annealing; Shape Grammar; Semantics; Optimization

1. INTRODUCTION

Manufacturing process planning is the problem of taking a part description, a description of an initial workpiece and knowledge of process capabilities, and generating a sequence of automatic and human-assisted actions that construct the part from the initial workpiece. Different plans can be ranked according to the cost of the initial workpiece, the time taken to execute the plan, and the operating costs of the different machines used in the plan. Process planning is a difficult problem, for a number of reasons: a suitable representation is required for the possibly complex geometry of the target part; planning knowledge must encompass reasoning with shapes and with the relationships between manufacturing processes, materials, shape changes, and geometric tolerances; and, finally, the space of possible plans is prohibitively large. An attempt to generate optimal plans obviously increases the complexity of the problem. In an attempt to manage this complexity, most approaches start by discretizing the geometry into features—collections of geometric data that correspond at a high level to manufacturing operations. Manufacturing knowledge is represented in terms of the processes required to remove material corresponding to features. From the feature-based description, a sequence of material removing operations is then constructed to produce the part. Normally, little attempt is made to optimize the plan, relying instead on the planning knowledge to rule out inefficient operations.

The work we present in this paper takes a different approach. We construct plans from the bottom up, using knowledge of the capabilities of the machine to generate the plans. By building from the low-level operations, we are able to produce accurate cost estimates, and we use these cost estimates during the planning process to direct the generation toward optimal plans. Previously, Brown et al. (1996) developed a computable representation of a manufacturing process in terms of a generative grammar of shape, which specifies the language of shapes able to be manufactured. A particular generation of a shape corresponds to a particular plan, and the generation is formally interpreted as the construction of that plan. We now further interpret the plan as the cost of the manufacturing sequence. This gives us a vast space, in which any shape on the fringe is manufacturable, and the path taken to reach a shape gives us a detailed costed process plan. This still, of course, leaves us with two prob-

Reprint requests: Dr. Ken Brown, Department of Computing Science, University of Aberdeen, King's College, Aberdeen AB24 3UE, U.K. Phone: +44 1224 272597; Fax: +44 1224 273422; E-mail: kbrown@csd.abdn.ac.uk

lems: directing our search through the space toward the target part, and obtaining the optimum plan. We could, at this stage, revert to a feature-based approach, and use a feature model of the target part to select and sequence the operations. However, because we have a computable description of the manufacturing process, which specifies all possible plans rather than those obtainable from feature models,[a] we have chosen instead to investigate other ways in which this space might be searched.

To cope with the complexity of the space, we apply a method we call *generative simulated annealing* to search for near-optimal plans. This technique is adapted from simulated annealing, and was originally devised by Cagan and Mitchell (1993) to control generations in shape grammars under the name *shape annealing*. Here we call it generative simulated annealing for two reasons: to distinguish it from the normal application of simulated annealing to solution spaces, and to emphasize that it is applicable to any generative search space and not just those defined by shape grammars. Rather than work with precomputed features, at each stage we compare the current state of the part with the target part and semirandomly generate a candidate operation. The control algorithm then decides whether or not to accept that operation. At its most basic level, our approach is knowledge-free with respect to higher level features, manufacturing heuristics and planning strategies; although to ensure good performance, we need a heuristic estimate of the cost to complete each partial plan. The aim of this work is to investigate whether the approach is feasible for solving problems of the size and nature of process planning. As these are initial investigations, we have restricted ourselves to a single manufacturing process, and made a number of simplifying assumptions. We do not believe that these assumptions are significant for demonstrating the feasibility of the approach.

In the next section, we present the necessary background. We briefly discuss the problem of process planning, concentrating on feature-based approaches. We then describe research into spatial grammars, followed by research into nonsystematic search techniques. In Section 3 we present our formulation of the domain, and discuss how the part shapes and the process are represented as a formal grammar of shape and its semantics. In Section 4, we discuss the generative simulated annealing algorithm in more detail and informally analyze its performance. In Section 5, we discuss how we cast our process representation as a planning problem, present the operators we use, and show how this is combined with the algorithm. Finally, in Section 6 we discuss the results of a number of experiments, and include directions for future research.

[a]The difference being that our plans can be generated at the level of individual tool paths. Feature-based methods are usually hierarchical, specifying removal blocks and then decomposing them into tool paths, which assumes that the optimal tool paths correspond to feature boundaries. In practice, however, this may not be a significant limitation.

2. BACKGROUND

2.1. Feature-based process planning

The general process planning problem consists of a number of interrelated subtasks, including selecting the different machine operations, determining the sequence of those operations, selecting the specific tools to carry out the operations, planning the low-level paths of those tools (for removing material and repositioning the tool), selecting the machines on which the operations will be executed, determining the machine parameters for each operation, ensuring that the workpiece is securely fastened, and frequently designing the fixtures and tools themselves. Each of these subtasks is a complex problem in its own right, and most research in computer-aided process planning tackles only a limited subset. There are two main approaches to producing plans: *variant* and *generative*. Below, we briefly review both of these approaches; a comprehensive treatment is given by Chang and Wysk (1985).

Variant process planning starts with a database of existing process plans for a range of parts. The job of the planner when presented with a target part is to find other similar parts in the database, retrieve their plans, and then modify the plans to correspond to the target part. The common method uses *group technology* or other coding schemes (see, e.g., Opitz, 1970), in which each part is assigned a code depending on the presence of various attributes, and this code is then used to select the closest plans. More recent AI-based work has used case-based reasoning to retrieve and modify plans (Marefat & Britanik, 1996). However, the problem still remains of determining which modifications are required to produce the given part.

Generative process planning, on the other hand, attempts to create a completely new plan for each part, using knowledge of manufacturing operations, process capabilities, and the part description. The main problem is managing the complexity. The most common approach is to access the process knowledge by way of *features*, where a feature is a representation of the ". . . engineering meaning of (part of) the geometry of a part or assembly" (Shah, 1991). A feature is a discretization of continuous geometric data, and in process planning typically represents a volume of material to be removed corresponding to a single manufacturing operation, possibly augmented with other manufacturing information. Before features can be used, however, they have to be recognized. Feature recognition is, unfortunately, a difficult task. Generally, a geometric representation of the target part is searched for specific topological patterns that correspond to the topology of generic features. Once a feature has been recognized, its attributes can be determined from the geometry (see, e.g., Joshi & Chang, 1988). Given a representation of the part in terms of features, the tasks are then to specify the operations that will construct the features, and to find an ordering on those features that corresponds to a feasible sequence of manufacturing operations

(e.g., Hayes & Wright, 1989). Typically, the knowledge required to carry out these tasks is represented using a mix of manufacturing heuristics and detailed process knowledge, and most systems are content to produce a feasible plan, relying on the heuristics to rule out obvious inefficiencies. The work of Nau et al. (1995) is a notable exception, in that their approach is oriented toward producing optimal plans. A more liberal definition of feature is used, and a set of general features recognized. Many different feature models for the target part can then be obtained from the set. The eventual model chosen, together with its associated plan, is then found by a branch and bound search. Husbands and Mill (1991) describe another approach to optimal process planning in which they used genetic algorithms to search in the space of feature-based plans. Features have also been used for variant planning, in which group technology codes are derived from feature models of the part (Shah & Bhatnagar, 1989; Srikantappa & Crawford, 1992). Finally, most of the work described in this section initially ignores the problem of ensuring that the workpiece is securely gripped throughout the manufacturing process, and only attempts to set the fixturing after the basic plan has been generated.

2.2. Spatial grammars and semantics

The shape grammar formalism was developed by Stiny (1980) to specify languages of shape. In his original formalism, a shape was defined to be a limited arrangement of straight lines in two-dimensional space, augmented with symbolic labels to describe nongeometric information, and grammar productions were rewrite rules defined in terms of labelled shapes. The rules could then be recursively applied to (sub)shapes to produce languages of two-dimensional shape designs. Figure 1 shows a simple shape grammar, the generation of a shape, and a number of other shapes in the language. Parametric shape grammars can also be defined, in which rule schemas with variable parameters are specified, and rule application proceeds with an instantiated version of one of the rule schemas. A number of shape grammars have been presented in the literature, including a grammar of Palladian villas (Stiny & Mitchell, 1978) and a grammar of Queen Anne houses (Flemming, 1987). A general definition of the algebra of shapes is given by Stiny (1991), encompassing three-dimensional solids. Shape grammars were originally used as a means of analyzing and understanding existing corpuses of designs, by showing that the designs conform to a regular structure.

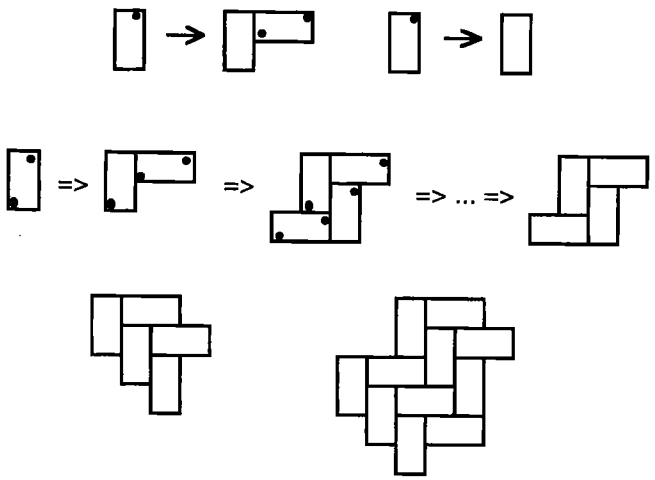


Fig. 1. Simple shape grammar.

Grammatical methods have also been applied in a number of cases to engineering problems. Fitzhorn (1990) describes how graph-grammars (in which production rules specify transformations of graphs rather than shapes or strings) may be used to specify spaces of realizable solids, using constructive solid geometry and boundary representations. Finger and Rinderle (1990) use a graph-grammar to generate gear layouts from functional requirements, while Brown et al. (1994*b*) use a grammar of constraint structures to specify a language of parametric shaft designs. Graph grammars have been used extensively in feature recognition, in which the grammars are used to parse solid models to recognize features (Finger & Safier, 1990; Chuang & Henderson, 1991). In manufacturing, Fu and de Pennington (1991) describe a system in which a grammar is used to translate backwards and forwards between representations in terms of design features and manufacturing features. Finally, Brown et al. (1994*a*) develop a grammar of shape that specifies the language of shapes manufacturable on a simplified lathe. In that work, the parametric shape grammar formalism is extended to include attributed labels and constraints on the values of the shape parameters and the label attributes occurring in rule applications.

The use of grammars for generation requires the ability to interpret the generated structures, to guide progress toward meeting a goal. The conventional view of the semantics of a representation provides a mapping from the elements of the representation to real-world objects. However, we can also regard the performance of the generated object as the semantics, either in terms of rigorous mathematical notions of performance, or the more fluid and subjective assessments of human users. We can use the constructive nature of grammatical tools to construct these semantics in tandem with the syntactic generation. Stiny (1981) proposed the generation of design descriptions by associating rules operating on descriptions with the grammar rules that operate on the shapes. Following on from the work mentioned above, Brown et al. (1995, 1996) applied a variant of this formalism to the interpretation of manufactured artifacts in terms of feature descriptions, and then developed description functions to interpret generations of shapes in the language as process plans. That latter work is discussed in Section 3, and we build directly on that work in this paper. For string and graph based formalisms, attribute grammars (Knuth, 1968) have been used, in which attributes representing additional information augment symbols, and attribute rules

define the computation of the attribute values. Variations on this theme are proposed by Penjam (1990), representing the resistance of electrical circuits, Rinderle (1991), representing forces and weights of boom designs, and Brown et al. (1992), representing stress concentrations on loaded shafts. These three applications are all oriented toward providing a final interpretation or assessment of objects in the language. When used in generation, the assumption is that the partial interpretation (constructed as a generation is in progress) can be used to direct the generation toward the desired end result. This is not necessarily the case, and depends on the particular semantic functions being used. The problem remains for each particular space and task of determining objective functions for which variations in the intermediate assessments after individual rule applications accurately reflect the variations in the assessment of the final objects resulting from those rules.

2.3. Nonsystematic search methods

Many general AI search techniques run into space or time problems when faced with vast search spaces. Recently attention has turned to nonsystematic search strategies, which do not guarantee completeness, but can improve average search performance. Harvey (1995) proposes nonsystematic backtracking, which occasionally ignores its own heuristics. Selman et al. (1992) have investigated GSAT, a stochastic search strategy for solving satisfiability problems, with promising results. GSAT can be considered to be essentially a greedy hill-climbing algorithm with random restarts. Simulated annealing (Kirkpatrick et al., 1983) is a stochastic optimizing technique for searching parameter spaces by iterative improvement, which has been used successfully in a number of domains. For search spaces in which solutions are constructed rather than improved, Cagan and Mitchell (1993) proposed shape annealing (defined to control the generation of shapes in a shape grammar, and hence its name). Shape annealing applies moves more or less at random, and they are accepted according to a steadily more discriminating acceptance criterion. Thus, a search in a design space consists of successive rule applications and retractions, which are initially randomly selected and accepted almost regardless of their effect on the objective function. As time progresses, moves that degrade the objective function are accepted with decreasing probability. Eventually, the algorithm converges to a good, possibly global but possibly local, optimum. Because shape annealing is stochastic, it is able to recover from early, poor local optima through backtracking. Thus, intermediate objective functions do not need to be strictly accurate, as moves that initially appear to take us closest to the goal but later turn out to have been nonoptimal are not fatal to the algorithm. Shape annealing has been applied to component layout (Szykman & Cagan, 1993) and the generation of truss topologies (Reddy & Cagan, 1995), and a variant, called recursive annealing, has been applied to machine design (Schmidt & Cagan, 1995).

The full definition of the shape annealing algorithm is as follows. Let $G = (S, L, R, I)$ be a shape grammar, consisting of a set of shapes, S, a set of labels, L, a set of shape rules, R, and an initial labeled shape, I. Let $f:(S,L)^+ \rightarrow \mathbf{R}$ be the objective function, mapping labeled shapes to real numbers. Let T be a real-valued variable, called the temperature. A state s_i is a labeled shape obtainable by recursively applying a sequence of n rules from R to I ($n \geq 0$). Let $C_i = f(s_i)$. A shape annealing move is then the application of a grammar rule r_j to s_i or the retraction of the last rule applied to obtain s_i, to obtain the labeled shape s_{i+1}. Let $C_{i+1} = f(s_{i+1})$. If $C_{i+1} < C_i$, then s_{i+1} is accepted as the next state. If $C_{i+1} \geq C_i$, then s_{i+1} is either accepted or rejected as the next state, with the probability of acceptance defined by

$$Pr(\text{accepting } s_{i+1}) = e^{-[(C_{i+1} - C_i)/T * Z(T)]},$$

where $Z(T)$ is a normalization factor. T decreases with time, according to a function called the *annealing schedule*.

Although it can be proven that simulated annealing under certain conditions relating to the parameters of the algorithm will always find the global optimum, the conditions are infeasibly restrictive for real applications. At least the same restrictions apply to shape annealing. However, although shape annealing has been shown to work well on some classes of problems, it appears that when tested on certain other problem classes, shape annealing can produce inferior solutions to standard simulated annealing (Szykman & Cagan, 1995). There are two main reasons for this. First, although the algorithm is designed to be tolerant to inaccuracies in the objective function, the choice of function for evaluating intermediate states is still significant. It appears to be harder to relate the early stages of a generation to the final object and make decisions on that basis than it does to represent violations of constraints in fully instantiated objects and minimize those violations. Second, there is a subtle difference in the implications of accepting a move in the two algorithms. This point will be explored in Section 4.

3. APPLICATION AND FORMAL LANGUAGE

3.1. Machining domain

Our domain of application is axi-symmetric machining on a simple lathe. The lathe is capable of drilling, external turning, boring, and external and internal gripping, and is shown in Figure 2. All workpieces must be axi-symmetric, and must be securely gripped during all cutting operations. We restrict gripping to be on cylindrical sections, and we assume the whole length of the gripping face must be in contact with the workpiece. External gripping may take place anywhere along the length of the workpiece, providing the workpiece can be inserted into the headstock, while internal gripping is restricted to be at either end. For the purpose of

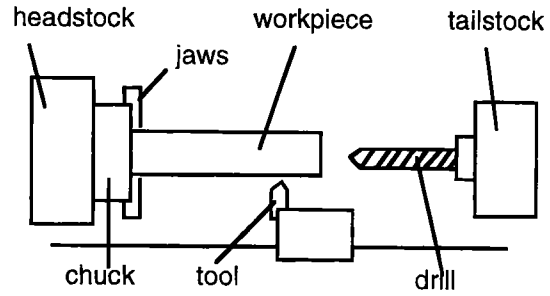


Fig. 2. Simple lathe.

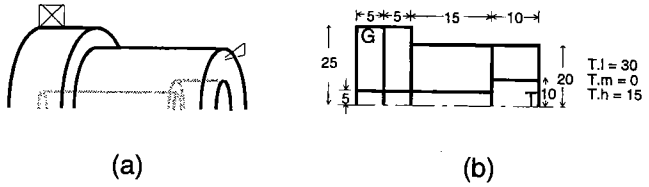


Fig. 3. Workpiece representation.

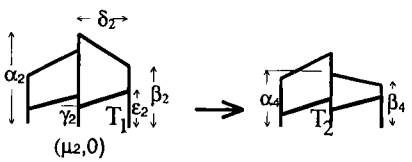
these experiments, we allow one finishing and one roughing tool each for turning and boring, and we allow drills with two different diameters. Roughing tools can only perform horizontal cuts, while finishing tools may cut profiles consisting of straight line segments. The gradient of external finishing cuts is restricted by the shape of the tool, while the gradient of boring cuts must not be oriented away from the center axis. At any time, cutting tools may be withdrawn and grips released, and the workpiece manually rotated.

Our intention is to plan at the level of operations. We define an operation to be either a gripping maneuver, or a cutting operation. Gripping maneuvers will consist of a regrip (including an optional reverse of the workpiece) and a location, while a cutting operation will consist of a tool selection, and a cutting path through the material. Our aim is to find a low-cost sequence of operations that will produce the part, and, as stated earlier, to attempt this planning problem using as little heuristic knowledge as possible, relying on our knowledge of the capabilities of the machine. The first thing to do, therefore, is specify the capabilities of the machine in an explicit, computable manner, and then use this specification as the basis of our planning procedure.

3.2. Grammar

The shapes in the language of the grammar are, ultimately, nominal shapes. We assume that all operations result in the intended shape, and we do not represent surface finishes, tolerances, or uncertainty in the shape. Given these assumptions, the grammar provides a formal computable specification of the capabilities of the lathe. Because our domain is axi-symmetric machining, we are able to represent our workpieces by ordered sets of simple two-dimensional parametric shapes. Labels attached to the shapes, together with their associated attribute values, will indicate the type, position, and status of the machine tools and the grips. For example, the simple workpiece in Figure 3a is represented by the shape in Figure 3b (by revolving it by 360° around the horizontal axis), where the label "G" indicates the shape is gripped from the left, and the label "T" indicates a turning tool adjacent to the right face. In this case, the label "T" has three attributes: "l" (representing the remaining length for the current cutting operation), "m" (the gradient of the cutting line), and "h" (the current cutting height), with values of 30, 0, and 15, respectively.

Each grammar rule represents a transition between a combined state of the workpiece and machine and a new state. Individual rules are represented as rewrite rules on sets of parametric, attributed labelled shapes, using the parametric attributed set grammar formalism (Brown et al. 1994*a*). If a rule can be instantiated so that its left-hand side exactly matches a subset of the existing labelled shapes and its constraints are satisfied, then the rule can be applied, removing the subset from the current shape, and replacing it with the right-hand side of the rule, if necessary extending the instantiation to satisfy the right-hand side constraints. Rules may be reflected in the vertical axis before being applied, corresponding to a workpiece which has been reversed during the process. An example rule is shown in Figure 4, representing a parametric turning operation. If an instantiation of the left-hand side matches a part of the current shape, and the constraints marked "C_L" are satisfied, then the turning tool can cut through that section. Thus, for this particular rule, the left constraints state that the section length (δ_2) must be no greater than the remaining reach of the tool arm ($T_1 \cdot l$), the maximum radius of the section (α_2, β_2) must be less than the maximum clearance height (η, a global constant), the section radius at the point where cutting will begin (β_2) must be greater than the current height of the tool ($T_1 \cdot h$), and the final radius (α_2) must be greater than the projected finishing height of the tool ($T_1 \cdot h + \delta_2 * T_1 \cdot m$). The first two constraints thus ensure that the operation being modelled is feasible, while the latter two constraints ensure that,



C_L: $\delta_2 \le T_1.l,\ \alpha_2, \beta_2 < \eta$
$\gamma_2 < T_1.h + \delta_2 * T_1.m \le \alpha_2$
$\varepsilon_2 < T_1.h \le \beta_2$

C_R: $\alpha_4 = T_1.h + \delta_2 * T_1.m,\ \beta_4 = T_1.h$
$T_2.h = \alpha_4,\ T_2.l = T_1.l - \delta_2$

Fig. 4. Example shape rule.

for the application of this particular rule, the cutting tool will always be in contact with the material. Once the left-hand side of a rule has been matched, it can be applied, modifying the dimensions of the shape according to the constraints marked "C_R" (using the same instantiation, or an extension of it). Thus, for this rule, the new section radii (α_4, β_4) are obtained from the attributes of the label that represent the cutting path ($T_1 \cdot h$, $T_1 \cdot m$), and then the attribute values are updated, computing the new height of the tool ($T_2 \cdot h$) and the reach remaining ($T_2 \cdot m$). An application of this rule to the shape of Figure 3 is shown in Figure 5, together with its intended interpretation.

There are over 120 rules in the grammar, divided into six sets—one each for external gripping, internal gripping, drilling, turning, boring, and retracting the tool. The majority of the rules do not modify the shape, but examine the workpiece and collect data (using attribute values) on the state of the shape and the machine. This data then act as constraints on subsequent rule applications. The grammar enforces the overall operating sequence shown in Figure 6: First, the workpiece must be securely gripped; any number of operations may then be applied; the workpiece must then be released; the process can then finish, or the workpiece regripped (after possibly being reversed) and the process repeated. At any given time, the state of the workpiece is represented by the current shape, and the state of the machine is represented by the attribute values and positions of the shape labels. The legal transitions between states of the workpiece and lathe are defined by the grammar rules. The changing values of the attributes, and the propagation of the labels over the shape as the shape changes, model the physical process that creates the finished object. The shapes in the language of the grammar then represent the parts that can be machined from the initial workpiece using the lathe. There are many possible ways to generate a given shape, corresponding to different manufacturing sequences.

The grammar we have briefly outlined here is an improved version of the one presented by Brown et al. (1994*a*). The most significant difference is that we no longer have the restriction that the tool must be retracted and the grip released after every operation, making it easier to sequence a series of operations for a single position of the workpiece. Note that the representation of the grammar rules has changed.

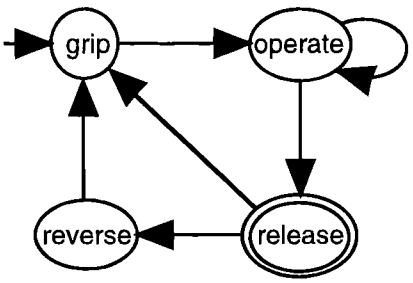


Fig. 6. Operating sequence.

3.3. Semantics

The grammar described above defines a space in which the final states (or fringe nodes) correspond to shapes manufacturable on the lathe. For process planning, however, it is not enough to demonstrate that a part *can* be manufactured—we must also state *how* it is to be manufactured. Above, we described the meaning of the grammar rules informally as manufacturing operations—this interpretation can be formalized, producing process plans corresponding to paths through the space. The plans we want to produce should describe unambiguously the operation of the machine. For each cutting operation, we will specify a safe height for moving a tool from its current position to the starting point of the operation, a tool, the type of cut (roughing or finishing), a feed rate (the rate at which the tool moves along the part), a cutting speed (the speed at which the workpiece is rotated), a cutting depth (representing the maximum depth of material able to be removed by a single pass of the tool), and a block of material to be removed. For gripping maneuvers, we specify whether the grip is external or internal, whether the workpiece is to be reversed, and the location of the grip.

We associate description functions with the individual grammar rules to construct the process plans as generation proceeds. The description functions operate on pairs, consisting of a material and a list of operations (because the

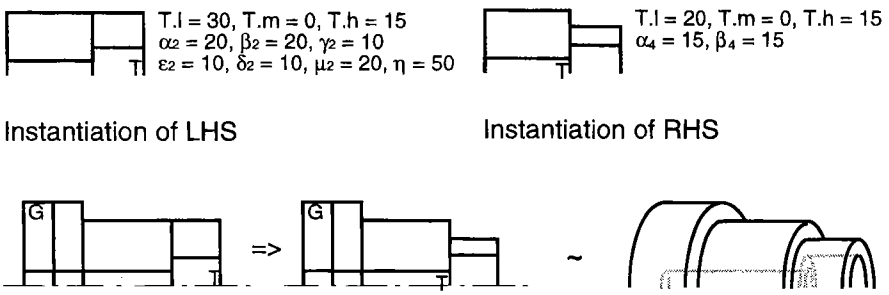


Fig. 5. Application of rule from Figure 3.

material does not change, we will omit it from the following discussion). Each operation is a 13-tuple, representing the different pieces of information above. The last element of the tuple is a list of blocks, where each block is a list of subblocks. Each subblock has single line segments for its upper and lower boundaries in a two-dimensional profile, and is represented by six parameters: the start and finish x-locations, and the y-values of the endpoints of the boundary lines. A block is then a connected sequence of subblocks, such that it has one continuous boundary (representing a cutting line), with a number of other restrictions. The description functions take as input the current labelled shape, the current list of operations, and the values used to instantiate the rule, and produce as output a new list. The description functions are not simple interpretations of the rules, for a number of reasons. In the grammar, the shape parameters are located with respect to the gripped end of the workpiece, while in control languages for automated lathes, the origin for specifying cutting operations is typically at the other end, and thus at least some of the rules must transform coordinates accordingly. To ensure that operations are valid, the grammar effectively enforces tool retraction between each pass of the tool; in certain cases, the semantic functions combine a number of units into a larger block with no retractions in between.

Each function is defined to act on the list of operations. The corresponding function for the rule of Figure 4 is shown below, and described informally in the next paragraph. We use the logic programming list representation—details of the syntax are given by Brown et al. (1996).

$f: L \rightarrow L :: [(z_1, z_2, \ldots, z_{12}, [(x_1, x_2, y_1, y_2, y_3, y_4)|T1])|T2] \rightarrow$
$[(z_1, z_2, \ldots, z_{12}, [(x_1, x_2 + \delta_2, y_1, y_2, \alpha_2, \alpha_4)|T1])|T2]$
if $y_1 = \beta_2$ and $(y_3 - y_1)/(x_2 - x_1) = (\alpha_2 - \beta_2)/\delta_2$
$:: [(z_1, z_2, \ldots, Z_{12}, B)|T2] \rightarrow$
$[(z_1, z_2, \ldots, z_{12}, [(\zeta - \nu_2, \zeta - \mu_2, \beta_2, \beta_4, \alpha_2, \alpha_4)|B])|T2]$
otherwise.

Most of the information regarding the parameters of the operation (z_1 to z_2) will have been set by the preceding rules, and this rule is primarily concerned with updating the description of the removal volume. If the list of subblocks for the current block has already been started, and the upper boundary of the first subblock [the points (x_1, y_1), (x_2, y_3)] is collinear with the upper line in the representation of the section about to be cut, then we extend the subblock by modifying the values of x_2, y_3, and y_4; otherwise, we start a new subblock and add it to the front of the list of subblocks. In that case, x_1 is set to $\zeta - \nu_2$, where ζ is a global variable initialized to the length of the initial workpiece (in our example, $\zeta = 35$), x_2 is set to $\zeta - \mu_2$, and the y-values are set to the coordinates of the shape removed by the rule.

To illustrate this function being applied in our interpretation, we must assume that a partial plan has already been generated. If we suppose that we had started with a cylinder, gripped from the left, drilled an axial hole, (finish) bored out that hole, turned down a section, and prepared to begin a new turning operation, we would have the following pair (omitting details of the earlier operations):

(m, [(rturn, λ, 0, λ, λ, λ, 15, t_1, r, 1.02, 30.5, 5.3, [])
(rturn, ...) (fbore, ...) (drill, ...) (egrip, ...)]).

If we now interpret the application of the rule shown in Figure 5, the pair changes to:

(m, [(rturn, λ, 0, λ, λ, λ, 15, t_1, r, 1.02, 30.5, 5.3,
[(0, 10, 20, 15, 20, 15)]) (rturn, ...) (fbore, ...)
(drill, ...) (egrip, ...)]).

Note that a new subblock had not been started, so we applied the second case in the function, and we used the values obtained from the instantiation of the grammar rule as shown in Figure 5. The informal interpretation of this pair is the following plan (including the details omitted above):

grip externally, location: 0, radius: 25
drill, d_1, location: 0, length: 40, feed: 1.02, speed: 30.5
bore, moving tool b_1 below height 10 until 0,
block [(0,10), (0,5), (10,10), (10,5)], finish: f,
feed: 0.25, speed: 73.0, depth: 1.4
turn, moving tool t_1 above height 20 until 0,
block [(0,25), (0,20), (25,25), (25,20)], finish: r,
feed: 1.02, speed: 30.5, depth: 5.3
turn, moving tool t_1 above height 15 until 0,
block [(0,20), (0,15), (10,20), (10,15)], finish: r,
feed: 1.02, speed: 30.5, depth: 5.3.

Note that in Figure 5 the shape has changed by removing a chunk of material, and in tandem with it, our representation of the partial process plan has changed by adding in the details of the second turning operation. This use of description functions to describe process plans was presented by Brown et al. (1996). The description tuples and the semantic functions have been improved, taking advantage of the changes to the underlying grammar, and the functions are now simpler and more intuitive. However, the basic formalism remains the same.

For each intermediate stage in the generation, we now have a detailed description of the operation of the lathe, including material to be removed, the different tools used, the speed of rotation of the workpiece, and the rate at which the tool cuts material. We can use this information to generate estimates of the cost of the partial plan. We will use the same formalism as above, and generate description tuples representing the volume of material, the time occupied in cutting material, the time taken by rapid movement and selection of the tools, an estimate of the time taken for manual operation (starting and stopping the lathe, and releasing, reversing, and gripping workpieces), and an estimate of the wear on each individual tool. Each semantic rule takes as

input a shape to which the grammar rule was applied, the instantiation of the rule, the original and resulting partial plans, and the existing cost description, and produces a new cost description as output. Although many of the grammar rules have associated functions that represent the cost contribution of those rules, most of the computation is done by the functions associated with the rules that finish an individual operation, that is, once a full description of an operation is available. The functions examine the block and maximum depth of cut and compute the required number of individual passes of the tool as part of the computation. The functions are further complicated by the way in which the process plan semantics may collapse two successive cutting blocks into a single block; in this case, the cost functions must erase the estimate of the cost of the first block, and recompute for the new block. As a result, we again use a list-based representation in building up the cost estimates, allowing us to unravel the representation when required. Although we have enough information to compute the tool wear based on cutting parameters, depths of cut, and material properties, because we do not use the tool wear in subsequent experiments, we have chosen instead to represent only the amount of time each tool is cutting material. As an example, using the same generation sequence as above, the cost estimate before the rule is applied is (omitting the volume of material):

([(cut, 6.059), (rapid, 1.25) (change, fb1, rt1) (rapid, 1)
(cut, 5.2882) (rapid, 0.25) (change, d1, fb1)
(rapid, 1.89451) (cut, 2.29577)],
[(release, ex) stop start (grip, ex)],
[(rt1, 6.059) (fb1, 5.2882) (d1, 2.29577)]),

while the cost estimate after the rule has been applied (and further rules applied to finish the operation) is:

([(cut, 1.8177) (rapid, 1.25) (cut, 6.059) (rapid, 1.25)
(change, fb1, rt1) (rapid, 1) (cut, 5.2882) (rapid, 0.25)
(change, d1, fb1) (rapid, 1.89451) (cut, 2.29577)],
[(release, ex) stop start (grip, ex)],
[(rt1, 7.8767) (fb1, 5.2882) (d1, 2.29577)]).

Note that extra elements have been added to the first list for repositioning the tool and performing the cut, and that the tool wear for tool rt1 in the third list has been increased. Once rules have been selected to finish the whole generation, an extra rapid movement time will be added to the first list. Note also that the second list already contains the operations to stop the lathe and release the workpiece. Because every grip or start must eventually be paired with a release or stop, this does not affect the final cost estimates, and makes it easier to compare different intermediate plans. Computing the total time for a given plan is then a simple matter of summing the times in each of the "cut" or "rapid" elements of the first list, and adding estimates for the "change" operations and the operations in the second list. Finally, although we do not do so, our list representation would make it easy to compare the costs of individual operations when comparing different completed plans.

In summary, the formal semantics of a rule is a function which, when the rule is applied in a particular situation, takes that situation and computes an interpretation. The end result of a generation in our system is a shape representation of a part that can be manufactured, a detailed plan for the manufacture of that part, and a measure of the cost of the plan. Intermediate stages in a generation give us an intermediate part, a plan to get that far, and an estimate of the cost of getting that far. We can now regard the grammatical space as a large search space in which the rules are parametric operators, and the semantic functions provide a cost function, giving the cost of reaching any point in the space. What remains for the process planning problem is a method of finding the cheapest path from the initial state to our target part.

4. GENERATIVE SIMULATED ANNEALING

The algorithm used in shape annealing (defined in Section 2) is more generally applicable than solely as a control mechanism for shape grammars. If we consider the initial shape to be an initial state and the grammar rules to be search operators, each of which expands certain types of states and returns new states, and we define particular shapes in the language to be goal states, we have a standard search space. The objective function can be considered to be either a simple distance function, or a combined distance function and estimated distance to a goal. The shape annealing algorithm is then a search technique: At any stage in the search, we either select and apply an operator to the current state, or we backtrack one step, to get a candidate state; we then compare the distance functions of the two states, and either move to the new state or remain where we are, according to the (changing) simulated annealing acceptance criterion. Note that we are not searching through complete configurations. To make this usage clear, we will refer to the algorithm as generative simulated annealing.

Note that each step in a path through the space constrains the subspace that is subsequently reachable, and thus indirectly contributes to the final cost, while in many problem spaces the goal is to find the shortest or cheapest path; and thus each step directly contributes to the cost. Also, the length of the partial paths, on average, increases with time. However, the acceptance criterion becomes more discriminating with time (as the temperature drops). Thus, decisions on whether operators and backtracks are accepted are not uniform over the length of the path, but are more discriminatory toward the end. If complete knowledge of the space and the evaluation function were available, it would be seen that the optimality of the steps in the path increases with depth. This phenomenon was referred to implicitly in the original paper by Cagan and Mitchell (1993), in which they discussed the way in which their results were crucially de-

pendent on the early choice of moves. If we assume that our aim is to find the shortest path, then each step contributes some distance to the total, and thus early steps are just as important as later ones. Therefore, if the algorithm is fortunate in the choices made in the early stages, it should produce close to an optimal path; if it is unfortunate, then no amount of optimization at the end will produce the optimum. (This is not to say that the early moves in the sequence are randomly chosen, since the algorithm frequently backtracks over those moves and tries alternatives. It is simply that when the algorithm is concentrating on the early stages, the acceptance criterion is loose, and as the criterion becomes more stringent and the sequence lengthens, the algorithm is less able to backtrack long distances, if we assume that we have an evaluation function that favors goal states.) In addition, the algorithm has no memory of the states it has visited, and once it has backtracked out of a superior path, it has no knowledge of the previous evaluations it had discovered and may subsequently settle on poorer states. Finally, it should be noted that generative simulated annealing in itself is not the cause of the above behavior, but its application to generative search in which each move restricts the space of reachable states, and each move contributes directly to the final evaluation. These arguments do not apply to systems in which only the final state evaluation is significant, and in which any state can be reached from almost any other state by forward rule applications—for example, Reddy and Cagan's (1995) truss topology grammar.

These comments suggest a number of possibilities for improving the algorithm. One is to ensure that the objective functions at the early stages better reflect the utility of the subspaces that the moves create, either by finding a better objective function or by changing the operators so that the initial moves create better partitions of the whole space. A second is to include some form of memory, so that previous objective function evaluations are incorporated in some way into the algorithm. A third is to change the way the algorithm moves through the space to try to compensate for the nonuniform optimization. Although not explicitly motivated by these goals, some research has been carried out into these areas. Cagan and Kotovsky (1997) investigated the propagation of objective function evaluations from states to neighboring states, effectively combining the memory approach with the improved objective function approach. Schmidt and Cagan's (1995) recursive annealing, in which design progresses through different levels of abstraction with annealing at each stage, provides better estimates of subsequent costs when deciding upon moves at higher levels. In our work, we consider the third option, and change the way the algorithm moves through the space. As our intention is to increase the likelihood of the algorithm backtracking out of poor local optima, we have modified the backtracking strategy, allowing the algorithm to jump back to arbitrary stages in the plan rather than backtrack one step at a time. Although the change in energy levels between the states may be greater than for individual steps (and thus making it less likely that we accept a jump back than a step back), the probability of accepting the jump back is greater than the cumulative probability of stepping all the way back to that point. Thus, we increase the ability of the algorithm to visit regions throughout the space, decreasing the chances of becoming trapped in a bad region. The new move set is shown in Figure 7. Note that this modification is similar in spirit to Gaschnig's (1979) backjumping, but relies on the annealing process to find a good backtracking height. An investigation of a number of different backtracking strategies on a simple problem, providing evidence supporting our modified backtracking, can be found in Brown and Cagan (1996).

5. PLANNING PROBLEM

Our manufacturing planning problem is to find the cheapest manufacturing sequence for a given part. Considering our cost estimate as a distance function, we want to find the shortest path through the search space to a goal node, and thus we have an optimizing search problem. Given the system described in Section 3, we face two main problems: We have a prohibitively large branching factor, because our operators are low-level and parametric, and, to evaluate individual nodes in the space, we must find an estimate of the cost to complete a plan.

Note that the grammar itself entails no notion of a goal state—every end state of a generation is a valid part manufacturable on the lathe, but the grammar rules do not refer to particular target parts, and do not prevent the simulated process from removing more material than is required. It is the responsibility of the controller of the grammar (whether human or automatic) to ensure that no more than the required material is removed. Note also that it is the controller's responsibility to instantiate the parameters in the rules (thus deciding on the parameters of the cutting operations) and in certain cases to decide on the details of the interpretation of the rules (selecting a roughing or finishing cut, for example). We have found it easiest to implement this by initially computing the total volume of material to be re-

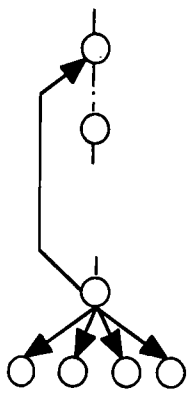


Fig. 7. "Backjump" move.

moved (by subtracting the target part from the initial workpiece) and then maintaining and updating this removal volume as the underlying grammar updates the shape. To ensure that we do not waste time attempting operations that would cut into the target part, we first postulate a cutting operation by examining the remaining removal volume and selecting a cutting line through that material. This gives us what we call a macro operation, which is then deterministically translated down to a sequence of instantiated grammar rules. If the sequence of grammar rules can be applied to the current workpiece, we know we have generated a legal operation. The semantic functions then extend the current process plan and estimate the cost, and we are then ready to suggest a new operation. To generate the candidate cutting lines, we must first select a tool (and hence a target surface finish). We then select an entry face on the removal volume, select an initial height, and, from the geometry of the neighborhood of the entry face, select a valid initial cutting gradient. We then use a simple shape grammar to generate a cutting line through the removal volume. Once the cutting line stops passing through the material of the removal volume, we have finished its generation and can pass the parameters to the main machining grammar.

This section where we generate the cutting line is one of two areas where we incorporate heuristic knowledge. First, we make a random selection of the entry face. Then we must select an initial height on that face. From our process plan semantics, we obtain the maximum cutting depth for a single pass of the selected tool, and compute the set of cutting heights corresponding to multiples of that maximum depth from the top of the entry face. We union this set with a precomputed set of the target diameters. We then bias the random selection of a cutting height to be from that set [using the heuristic obtained from Green (1992) stating that once the cutting tool has been obtained, select the deepest cut that will achieve the desired finish]. Similarly, when generating the gradient of the line, we bias the random selection toward selecting the gradient of the desired surface. Finally, we impose a bias toward generating the longest cutting line given those initial parameters. Figure 8 illustrates the set of likely cutting heights for an external (roughing) cut.

We have now described the mechanism for postulating individual operations, and simulating them in the grammar. As discussed in Section 3, the semantics of the grammar produce process plans and detailed estimates of the cost of partial plans. This, however, is not enough information to control the search effectively. Selecting the cheapest operation at each stage will not normally lead us toward the optimum plan. We also require an estimate of the cost to completion of any given plan. However, it is not possible to obtain an accurate prediction of the cost to completion without solving the planning problem itself, and so we rely on simple heuristics. Once we have the two different cost estimates (the cost of partial plans and the cost of completing those plans), we can combine them together to get a heuristic estimate of the worth of any node in the space. The

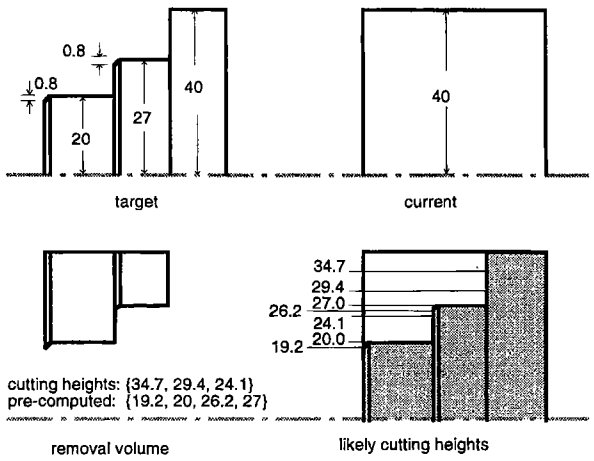


Fig. 8. Likely cutting heights for an intermediate workpiece.

heuristic functions we use estimate the time required to process the removal volumes (discussed above). We examine the removal volumes, considering each surface to be machined in turn. We compute the total volume to be removed above each surface, and then use the characteristics of the machining parameters to estimate the time for a single pass of the tool through this volume. We then compute the total height of that section divided by a cost factor, take the maximum of that with 1.0, and take the product of that with the time estimate for a single pass. The cost factor is related to the maximum depth of cut for the roughing tool, which is 5.3, and the reason we multiply by at least 1.0 is to ensure that we are estimating the time for at least one cut. The standard cost factor we use is 4.5, giving us some slack in our estimate to allow for rapid movement of the tool between cuts, and cases where some cuts are less than the maximum depth. For surfaces requiring a finishing tool, we also add on a multiple of the length of the section. Once we have computed the cutting time estimate, we then examine the access faces and count the different surface types, and then add an estimate for the number of tool changes and workpiece reversals. The combined sum is our heuristic estimate of the cost to complete the plan.

We now have a search space in which the planning operators are parametric, and are at the higher level, rather than at the level of the grammar rules. Note that we still require the underlying grammar as well as the operations on the removal volume. The macro operators simply postulate possible cuts, while the underlying grammar checks that the postulated cuts are valid, and provides the process plan and a detailed estimate of the incurred costs. Technically, because our operators are parametric over continuous domains, the branching factor is infinite. In our implementation, we impose a minimum granularity on the parameter values, and so the branching factor is finite, but the space is still too vast to search by conventional means. However, we now have all the capabilities required for the generative simu-