An Approach to Feature-based Comparison of Solid Models of Machined Parts*

Vincent A. Cicirello† William C. Regli‡

Abstract

Solid Models are the critical data elements in modern Computer-Aided Design (CAD) environments, describing the shape and form of manufactured artifacts. Their growing ubiquity has created new problems in how to effectively manage the many models that are now stored in the digital libraries for large design and manufacturing enterprises. Existing techniques from engineering literature and industrial practice, such as group technology, rely on human-supervised encodings and classification; techniques from the multimedia database and computer graphics/vision communities often ignore the manufacturing attributes most significant in the classification of models.

This paper presents our approach to manufacturing similarly assessment of solid models of mechanical parts based on machining features. Our technical approach is three-fold: (1) perform machining feature extraction; (2) construct a model dependency graph (MDG) from the set of machining features; (3) partition the models in a database using a measure of similarity based on the MDGs. We introduce two heuristic search techniques for comparing MDGs and present empirical experiments to validate our approach using our testbed, the National Design Repository.

Keywords: Computer-Aided Design (CAD), geometric reasoning, engineering design databases, Computer-Aided Manufacturing (CAM), Computer-Aided process planning (CAPP).

1 Introduction

We present an approach to comparing the manufacturing similarity of solid models of machined artifacts based on their machining features. Increasingly, manufacturing enterprises must maintain vast digital libraries and databases of Computer-Aided Design (CAD) and Computer-Aided Process Planning (CAPP) knowledge. Such information includes the parametric solid models of parts

*Cicirello, V.A. and W.C. Regli. An Approach to Feature-based Comparison of Solid Models of Machined Parts. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 16(5): 385–399, November 2002.

†The Robotics Institute, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213, cicirello@cs.cmu.edu

‡Geometric and Intelligent Computing Laboratory, Department of Computer Science, Drexel University, 3141 Chestnut Street, Philadelphia, PA 19104, regli@drexel.edu

and assemblies, as well as Numeric Control (NC) machining programs, production plans and cost data.

Our research goal is to develop methods to interrogate large knowledge-bases of solid models of machined artifacts in order to enable variational process planning and cost estimation. This work is part of the National Design Repository project[1], which is an ongoing effort to collect and archive "open source" CAD data and solid models and develop data management technologies for handling engineering information.

Given that we have a large Repository we wish to search, and a new solid model to use as a query, our technical approach, which is illustrated in Figure 1, is three-fold:

1. **Perform feature extraction to map the solid model to a set of machining features.**

   If operating inside a CAD environment, one could plan to retain the design features as new models are created.[2] However, as has been often noted, design features are not necessarily in one-to-one correspondence with manufacturing features. For legacy data and for solid models that are converted between modeling systems, there may not be any readily available feature information. Our work uses volumetric STEP AP 224 machining features acquired from the models by automatic feature recognition. Note that this paper uses existing feature identification technologies and does not present a new feature recognition technique—rather, we present a whole new domain for in which feature recognition can be employed.

2. **Construct a *model dependency graph* (MDG) from the set of machining features.**

   Given the set of machining features for an artifact, our model dependency graph represents an intermediate data structure to be used to model feature interactions and dependencies. Our belief, which this paper empirically tests, is that machined parts with similar feature sets and similar feature interactions have a high probability of possessing similar manufacturing plans. *Model dependency graphs* capture the feature interactions in a given set of features and are used as basis for comparisons among the CAD models.

3. **Query by Example: Partition the models in the database according to a measure of similarity to the query model.**

   Our database consists of 259 models taken from the National Design Repository. Based on their model dependency graphs, interrogation of the Repository becomes a task of comparing the MDG of a query part to those in the database. The specific comparison that we are interested in is that of the *largest common subgraph* (LCS). The general problem of determining the LCS for a given pair of graphs is NP-complete (Garey and Johnson, 1979). This paper introduces several heuristic techniques to solve LCS in the context of our problem:

   - First, we can check for the *existence* of a subgraph isomorphism much more efficiently than we can find the specific LCS mapping. If one exists, we know the LCS is simply the smaller of the two model dependency graphs. If not, we turn to the LCS computation itself. We introduce an approach to the subgraph isomorphism problem based on a

[1] `http://www.designrepository.org`

[2] In earlier work, we modified a random part generator to output the design features used during the generation (Cicirello and Regli, 1999; Cicirello, 1999).

greedy, but complete, heuristic extension of Ullmann's algorithm (Ullmann, 1976) that we call *Greedy Subgraph Isomorphism Checker (GSIC)*.

- Second, for purposes of CAD model indexing, it is not necessary to find the largest common subgraph—what we really need is a measure of similarity. Our approach is to find a "sufficiently" large common subgraph of two MDGs and use it to estimate the edit distance between the MDGs and build a graph similarity measure. Hence, we can use an iterative improvement search algorithm for the largest common subgraph computation. Specifically, we employ a variant on hill-climbing/gradient descent search (Russell and Norvig, 1995) that exploits the feature information in the extracted machining features.
- Third, there is a great deal of domain knowledge present in the CAD model and in the machining features that can reduce the search space. For example, we will only consider mappings that compare similar feature types (i.e., holes map to holes, not to pockets). Additional constraints about vertex degree and size, location, and orientation can also be considered. Also, domain knowledge (e.g., tolerances, manufacturing attributes, surface finish specifications, etc.) can be used to further reduce the search-space size.

We exploit this manufacturing knowledge to create tractable methods for manufacturing similarity comparisons among solid models based on MDGs.

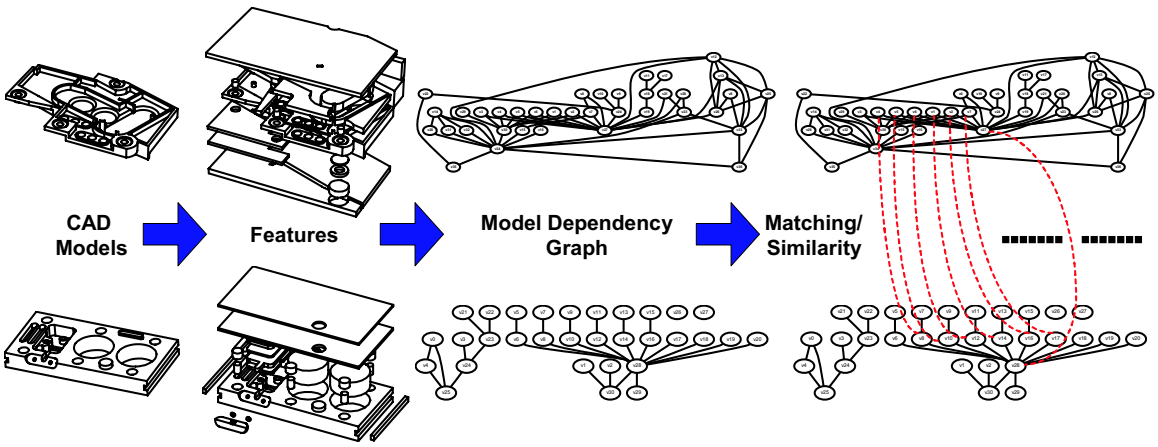


Figure 1: Our three-fold approach: (1) perform machining feature extraction to map the query solid model to a set of volumetric STEP AP 224 machining features; (2) construct a model dependency graph from the set of machining features; (3) partition the models of the Design Repository according to a measure of similarity to the query model using an iterative improvement search across the models of the Repository.

This paper presents the approach, our algorithms for model comparison and empirical results based on a set of 259 real-world CAD models from the National Design Repository. The primary contribution of this research is our overall approach to performing machining similarity comparisons among manufactured artifacts found in modern engineering databases. Two key contributions

within the approach include the formulation of the *model dependency graph* structure and the adaptation of heuristic and iterative improvement search algorithms to the comparison of these graphs. While the experimental data presented in this paper employs a specific feature identification technology, our approach is not limited to this technique and can be used with any feature recognition technique capable of finding interacting features.

We believe that this work is the first fully automated technique for machining feature-based comparisons of machined artifacts. To achieve this end, this work has created novel data structures, heuristic search and graph comparison techniques that can be applied to a number of important problems in engineering databases and manufacturing process planning (e.g., model indexing, variational process planning, process selection, and cost estimation). We believe that our work creates part of a foundation that will advance our abilities to manage digital data in distributed engineering enterprises.

This paper is organized as follows: Section 2 presents background research from the engineering, CAD and database communities. Section 3 introduces our technical approach to the problem, defining the *model dependency graph* structure and outlining the algorithmic techniques to compare them. Section 4 describes a set of experiments and our results when this technique is applied to a real dataset of CAD models. Section 5 gives our conclusions and directions for future work.

2 Background and Related Work

In engineering practice, indexing of parts and part families had been done with group technology (GT) coding (Snead, 1989). Group technology facilitated process planning and cell-based manufacturing by imposing a classification scheme on individual machined parts. GT codes specified classes using alphanumeric strings. These techniques were developed prior to the advent of inexpensive computer technology, hence they are not rigorously defined and are intended for human, not machine, interpretation. At one level, we see our research as augmenting traditional group technology coding schemes by providing a completely digital process for storage and comparison of solid models. It should be noted, however, that at other levels the approach we advocate is not limited to categorization of solid models of machined parts. In recent experiments, we have begun to study how our graph-based approaches can be used to index assemblies, design process knowledge, and CAD data from other domains (e.g., AEC).

2.1 Machining Feature Recognition

In the areas of manufacturing process planning and solid modeling past research efforts have developed a variety of techniques for reasoning about geometric and topological information. Most related to the work in this paper is research on hint-based and trace-based recognition of volumetric machining features. Marefat et al. introduced a novel way of integrating evidence-based reasoning with geometry for process and inspection planning (Marefat and Kashyap, 1990; Marefat and Kashyap, 1992; Marefat et al., 1993). Marefat's more recent work has focused on how to more effectively adapt new planning techniques to process planning (Britanik and Marefat, 1995). Vandenbrande, Han, and Requicha integrated knowledge-based systems with solid modeling to identify machining features and perform process planning for machined parts (Vandenbrande and Requicha, 1993; Vandenbrande and Requicha, 1994; Han and Requicha, 1994; Han and Requicha,

1995). Some of the author's past work on geometric reasoning for manufacturing feature identification and process planning includes (Gupta et al., 1994; Regli et al., 1995; Regli et al., 1997). Recent work by Gaines et al. generates volumetric feature instances directly from the shapes' machine tools from a tool library (Gaines and Hayes, 1999; Gaines et al., 1999; Gaines and Hayes, 2000).

There are many other research papers and prototype feature recognition systems—each with its own strengths and weaknesses. For comprehensive literature surveys on feature recognition for machining, interested readers are referred to (Allada and Anand, 1995; Ji and Marefat, 1997; Han et al., 2000; Shah et al., 2001).

2.2 Comparisons of Shape and Solid Models

The literature in this area is rather brief, consisting of results from engineering, computer science and, in particular, computer vision communities. Elinson et al. used feature-based reasoning for retrieval of solid models for use in variant process planning (Elinson et al., 1997). Cicirello and Regli examined how to develop graph-based data structures and create heuristic similarity measures among artifacts (Cicirello and Regli, 1999; Cicirello, 1999; Regli and Cicirello, 2000)—work which this paper extends to manufacturing feature-based similarity measurement. Other recent work from the Engineering community includes techniques for automatic detection of part families (Ramesh et al., 2000) and topological similarity assessment of polyhedra (Sun et al., 1995).

The mainstream computer vision research has typically viewed shape matching in approximate domains, such as from models generated from range and sensor data. Work at the University of Utah (Thompson et al., 1995; de St. Germain et al., 1996; Thompson et al., 1996) enables reverse engineering of designs by generating surface and machining feature information from range data collected from machined parts. Jain et al. have been working on handling multimedia data such as pictures (GIF, JPEG, etc.) and CAD data (Gupta and Jain, 1997). Their approach is based on the creation of "feature vectors" from 2D images that capture concepts such as color, density, and intensity patterns. Their work in extending these techniques to 3D CAD data treats the CAD information as sets of point clouds (such as generated with range data) to be compared. While we believe these techniques hold promise, they fail to exploit the availability of 3D solid models representing the CAD data, as well as the engineering information included with CAD data about tolerances, design/manufacturing features, and inter-part relationships that occur in assemblies.

One example where CAD data is employed is the 3D Base System (Cybenko et al., 1996; Cohen, 1996) from Dartmouth College. 3D Base operates by converting CAD models (a solid model or surface model) into an IGES-based neutral exchange file. A voxel (3D grid) representation is generated from the IGES data and then used to perform pair-wise comparisons among the CAD files using geometric moments of the voxels and by comparing other pre-computed part features (such as surface area). Their work operates only on the gross-shapes of single parts and does not operate directly on the solid models. It does not consider information pertaining to manufacturing or design features, tolerances, or design knowledge that might be present in the corporate database; the voxelization approach would be impractical to scale to electro-mechanical assemblies, where inter-part relationships and models of function and behavior are much more significant than gross shape properties.

Many other techniques find their roots in computer vision and computer graphics (Bhanu, 1982;

Stewman and Bowyer, 1990; Cohen and Wolfson, 1994). These, while powerful, are not suitable for off-the-shelf use in similarity matching of solid models.

3 Technical Approach

Figure 1 illustrates the phases of our approach: first recognize features, second generate model dependency graphs and third compare graphs for part retrieval. We address each of these components in turn through the following Sections.

3.1 Machining Feature Recognition

While our technical approach involves automatic recognition of machining features, this paper is not presenting specific new advances in feature recognition. In the context of this work, feature identification from solid models parallels that of image segmentation (Haralick and Shapiro, 1985) and its use in image databases, i.e., to find the semantically meaningful objects in a model (or image). The central contribution of this work is the methodology for machining similarity comparison of solid models, which is independent of which feature recognizer and which set of machining features one wishes to compare against. For our methodology, however, we assume three properties of the feature recognition module:

1. **Recognizes Machining Features:** Recognizers that return shape or form features are not of significant use in assessing machining similarity among artifacts. As has been noted previously (Gupta et al., 1995), machining features contain manufacturing process knowledge in addition to shape information.

2. **Recognizes Interacting Features:** Feature interactions significantly influence the process plans and selection of machining operations and fixtures (Regli and Pratt, 1996). Further, all but the most trivial of machined parts have interacting features.

3. **Potential to Return Multiple Feature Interpretations:** In some cases it might be useful to consider the *feature space* of available machining operations in order to assess the properties of an artifact. The problem of multiple interpretations has been widely studied (Han, 1997) and greatly influences selection of process plans. Using *feature cover* (i.e., the total set of possible machining features) to create the index allows comparisons among artifacts to broaden the consideration to include parts with process plans that come from similar *feature spaces*.

As noted earlier, there are many academic and research prototype feature identification systems (Allada and Anand, 1995; Ji and Marefat, 1997; Han et al., 2000; Shah et al., 2001). Rather than create a special-case feature recognizer, we use one of the few available industrial systems: the Feature-Based Machining Husk (FBMach) (Brooks and Wolf, 1994; Brooks and Greenway Jr., 1995) from Honeywell, Federal Manufacturing Technologies in Kansas City. FBMach is a robust library of machining features and feature recognition algorithms. It comprises over 15 man-years of effort and several hundred thousand lines of C++ and ACIS code.

FBMach is, fundamentally, a hint-based feature recognition system. It employs three different approaches to define features: (1) automatic recognition, (2) interactive recognition and (3) manual

identification. The automatic recognition uses a procedural algorithm to search for feature hints and then creates feature instances using the hints without user interaction. The interactive recognition allows the user to provide some hints for FBMach to use in generating the feature instances. For example, the user may identify a pocket by selecting its bottom face. The manual identification allows the user to create a feature instance by adding each face to the feature individually and defining each face's role in the feature (side, bottom, top, etc.). FBMach implements a *human-supervised reasoning* approach, which has also been explored by van Houten (van Houten, 1991) along a different direction. Such human-supervised reasoning is often quite useful for producing good feature models.

We used a slightly modified version of FBMach that translated unattributed ACIS `.sat`-based solid models into sets of STEP AP 224 (Slovensky, 1994) NC machining feature volumes. The feature sets returned by FBMach were used to create the *Model Dependency Graphs* described in the next Section and were the basis of the empirical results described in Section 4. Examples of the feature recognition output are shown in Figures 1 and 3.

3.2 Definition and Generation of Model Dependency Graphs

A **Model Dependency Graph** (MDG) (Cicirello and Regli, 1999) is a high-level, intermediate representation of a solid model, its features and their inter-dependencies. An MDG is a directed acyclic graph (DAG) with the following characteristics: (a) the nodes of this graph correspond to individual features, (b) an edge between two nodes corresponds to an *interaction* or dependence between the features—for example, in the case of volumetric features, this could be a non-empty intersection of the feature volumes. The direction on the edges capture the order that features were applied to create the model.

Note that a MDG, as defined above, encompasses any kind of feature for which an *interaction* can be defined. In this way, an MDG can refer to a part description in terms of *design* features (as shown in Figure 2) or machining features (as show in Figure 3). In both cases, the *precedence constraints* among the features induce the direction of the edges in the MDG. For example, the design features are ordered based on the sequence of the design operations; the machining features could be ordered based on a process plan.

Observation: *D-morphisms of Model Dependency Graphs.* Let G_1 and G_2 be two MDGs for the same solid model resulting from different orderings of a feature set $F = \{f_0, \ldots, f_n\}$ (such as shown in Figure 2). G_1 and G_2 are D-morphic. A proof of this property appeared previously in (Cicirello and Regli, 1999).

For a given pair of graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ a *D-morphism* is formally defined in (Garey and Johnson, 1979) as a function $f : V_1 \rightarrow V_2$ such that for all $(u, v) \in E_1$ either $(f(u), f(v)) \in E_2$ or $(f(v), f(u)) \in E_2$ and such that for all $u \in V_1$ and $v' \in V_2$ if $(f(u), v') \in E_2$ then there exists a $v \in f^{-1}(v')$ for which $(u, v) \in E_1$.

As defined above, the MDG is not immediately applicable to a set of machining features for which the order of the operations is not known. In this paper we are considering machining features that are extracted post facto: we do not know what order they were applied and will not attempt to designate some arbitrary and meaningless ordering on the machine features.[3] Rather we will

[3]This is the domain of Computer Aided Process Planning (CAPP).

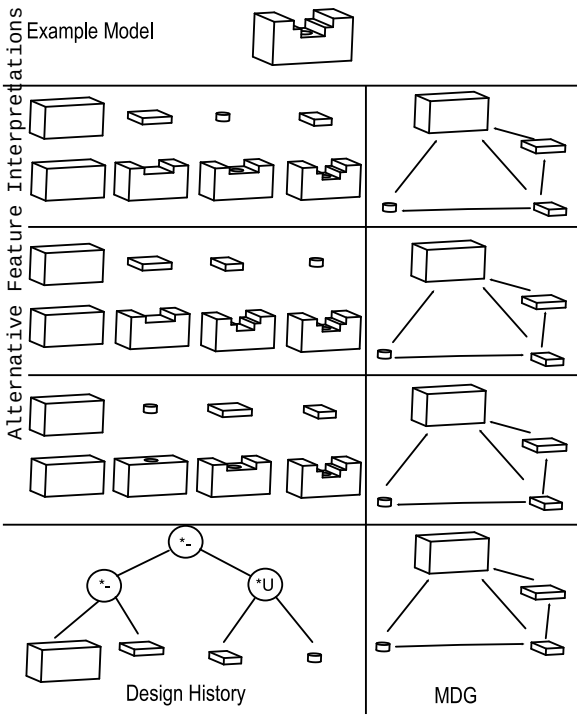


Figure 2: (From (Cicirello and Regli, 1999)) Pictured is a single solid model and several alternative feature-based models, and one possible CSG tree, that can produce it. On the right are the MDGs for each of these alternatives—note that they are all D-morphic to one another.

exploit an undirected version of the MDG which we shall call the **Undirected Model Dependency Graph**.

Undirected Model Dependency Graph: The undirected model dependency graph (UMDG) $G = (V, E)$ for a solid model is defined as a set of nodes $G = \{f_0, \ldots, f_n\}$ where the f_i are the machining features that have been extracted from the model. The edge set E of the UMDG can be defined as: $E = \{\{f_i, f_j\}\}$ such that $vol(f_i) \cap vol(f_j) \neq \emptyset\}$.

Figure 3(a) illustrates the UMDG and its generation for a solid model of a bracket from the Design Repository. In this Figure can be seen the bracket itself along with the STEP AP 224 machining features that have been extracted using FBMach. For each of these features a node has been added to the UMDG that is labeled with attributes of the feature such as its type and dimensions. For each pair of nodes, an edge has been added if the two nodes interact (i.e., have a non-empty intersection of the feature volumes). Figure 3(b) similarly illustrates the UMDG and its generation for the solid model of a torpedo motor.

3.3 Part Comparison and Retrieval

To compare the similarity of two solid models, compute the size of the largest common subgraph (LCS) of the corresponding UMDGs. The LCS problem is to find a pair of subgraphs, one from each of the two input graphs, such that the subgraphs in question are isomorphic to each other and the largest such subgraphs in terms of the number of edges. As noted earlier, the LCS is

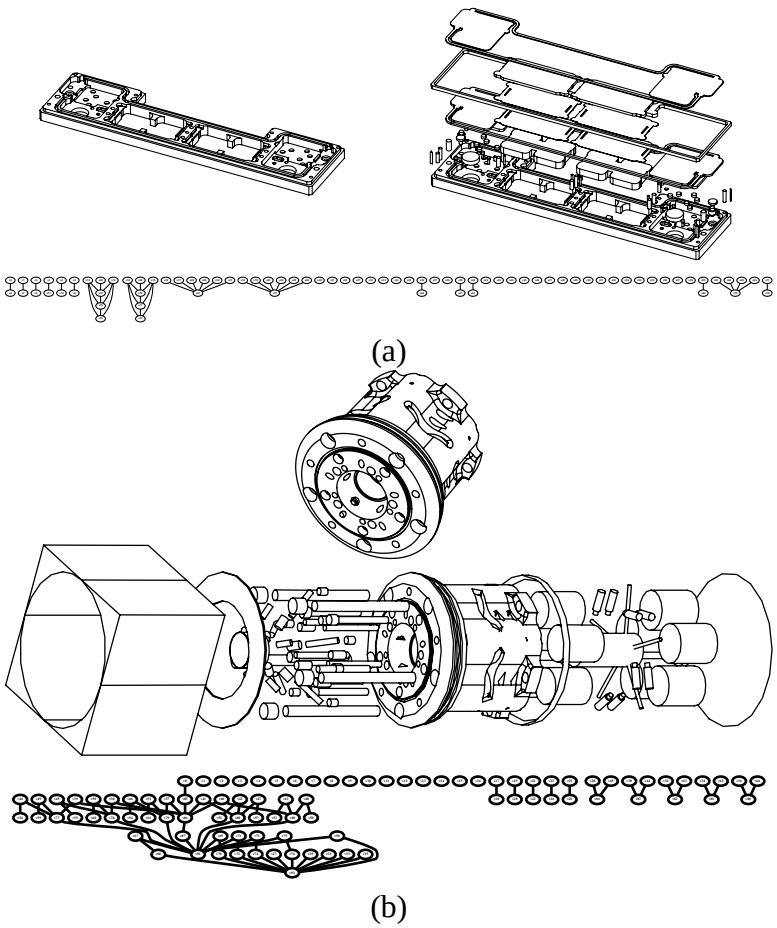


Figure 3: Example of the UMDGs generated from (a) the solid model of a bracket; and (b) the solid model of a torpedo motor.

NP-complete and an algorithm computable in polynomial time is not likely to exist.

We are primarily concerned with the manufacturing feature similarity among artifacts. Hence, knowing the absolute LCS of the UMDGs is not necessary. Instead, a "large enough" common subgraph is sought. There exist some inexact solutions in the literature including an approach using a two-stage Hopfield neural network (Shoukry and Aboutabl, 1996) and a meta-optimized genetic algorithm (Cicirello and Smith, 2000). There are also some inexact solutions to the very closely related problem of error-correcting isomorphism including a decision tree approach (Messmer and Bunke, 1996), a linear programming approach (Almohamad and Duffuaa, 1993), and a genetic algorithm (Wang et al., 1997).

We develop a heuristic method for the computation of the LCS based on a variant of iterative improvement search (specifically, hill-climbing/gradient descent) (Russell and Norvig, 1995). To further refine and narrow the search space, our algorithm utilizes the domain knowledge present in the CAD model. For example, we will only consider mappings that compare similar feature types (i.e., holes map to holes, not to pockets). Additional constraints about vertex degree and size, location, and orientation can also be considered.

The following sections present our approach. While this is not guaranteed to find the absolute

Input: $G_1 = (V_1, E_1), G_2 = (V_2, E_2)$, the two graphs being tested.
Output: true if the graphs are found to be isomorphic or false otherwise.
ULLMANN(G_1, G_2)
(1) M = INITULLMANN(G_1, G_2)
(2) **if** REFINE(M, G_1, G_2) = 0
(3) **return** false
(4) **else**
(5) N = SORT(V_1)
(6) **return** ULLMANNDFS(M, G_1, G_2, N, LENGTH(N)−1)

Figure 4: Ullmann's Subgraph Isomorphism Algorithm.

LCS, it allows for a *measure of manufacturing similarity* based on the best result obtained from executing some number of restarts of the algorithm.

3.3.1 MDG Exact Matching Algorithm

The problem of determining whether a subgraph isomorphism exists is, in practice, a much easier problem computationally than computing the size of the LCS. Thus, before computing the LCS, we check for a subgraph isomorphism. If one exists, it implies that the LCS is the smaller of the graphs being compared. If one does not exist, we use our approximation algorithm to compute the LCS. Our exact algorithm for the subgraph isomorphism problem is a greedy heuristic extension of Ullmann's depth-first search based algorithm (Ullmann, 1976). We now describe Ullmann's algorithm followed by the *Greedy Subgraph Isomorphism Checker (GSIC)* algorithm, our extension of Ullmann's algorithm.

Ullmann's Algorithm Ullmann's algorithm for subgraph isomorphism is perhaps the most widely used algorithm for subgraph isomorphism. Even today, more than 25 years after its development, researchers often compare the performance of their algorithms to that of Ullmann's. The complete original description of Ullmann's algorithm can be found in (Ullmann, 1976). Since our algorithm is essentially built upon that of Ullmann, we here present a brief description.

Ullmann's algorithm for subgraph isomorphism (see Figure 4 and Figure 5) is a depth first tree search. Each state in the search space is represented by a matrix M. This matrix is $n \times m$ where there are n nodes in the smaller graph and m nodes in the larger. The matrix elements m_{ij} are 0 if the corresponding nodes may not be mapped to each other in any isomorphism and 1 otherwise. The initial state is generated based on the degrees of the nodes. And although not discussed in (Ullmann, 1976), it is clear that to handle attributed graphs simply incorporate the attributes into the initialization stage. That is, if the attributes of node i of graph G_1 are inconsistent with those of node j of graph G_2, then m_{ij} is initialized to 0. Otherwise, if the the degree of node i of graph G_1 (the smaller graph) is greater than the degree of node j then m_{ij} is initialized to 0. If both of these conditions are false then m_{ij} is initialized to 1. This initialization is handled in line 1 of Figure 4.

Ullmann's algorithm then proceeds depth-first. Each successor state binds a node mapping

Input: $G_1 = (V_1, E_1), G_2 = (V_2, E_2)$, the two graphs being tested. M is the mapping matrix M previously described. N is a sorted list of the nodes of the smaller graph G_1. $Level$ is the current level of the DFS.
Output: true if the graphs are found to be isomorphic or false otherwise.

```
ULLMANNDFS(M, G1, G2, N, Level)
(1)  forall v2 in V2 do
(2)      if M[N[Level], v2] = 1
(3)          Mnew = BIND(N[Level], M, v2)
(4)          if REFINE(Mnew, G1, G2) = 0
(5)              DO NOTHING
(6)          else
(7)              if Level = 0
(8)                  return true
(9)              else
(10)                 if ULLMANNDFS(Mnew, G1, G2, N, Level − 1 ) = true
(11)                     return true
(12) return false
```

Figure 5: The depth-first search portion of Ullmann's Subgraph Isomorphism Algorithm.

by setting all but one "1" in a row of the matrix M to 0 (see line 3 of Figure 5). The bindings are considered in order of non-increasing node degree (see line 5 of Figure 4). After this binding is performed, an arc consistency check is applied iteratively to each remaining 1 in the matrix M. This check is referred to as Ullmann's neighborhood consistency check. For each $m_{ij} = 1$ in M it checks to ensure that for all neighbors x of i in graph G_1 there must exist a neighbor y of j in graph G_2 such that $m_{xy} = 1$. If this condition does not hold, then m_{ij} is changed to 0. This neighborhood consistency check is performed by the calls to "REFINE" in line 2 of Figure 4 and line 4 of Figure 5. "REFINE" returns false if there exists an all zero row in M after the neighborhood consistency check is performed and true otherwise. An all zero row implies that no subgraph isomorphism is consistent with the matrix M. If each row of M has exactly one 1 and all columns of M have no more than one 1, and if the neighborhood consistency check does not alter M, then M represents a subgraph isomorphism. Ullmann's algorithm is described in Figure 4 and Figure 5.

The worst case complexity of Ullmann's algorithm is exponential. This occurs if all or a large number of the elements of the M matrix are 1 and if the refinement procedure fails to reduce the search space. In practice, this worst case is far from typical; and Ullmann's algorithm is usually quite efficient.

Greedy Subgraph Isomorphism Checker (GSIC) As previously stated, Ullmann's algorithm for subgraph isomorphism is still widely used today. In the worst case, it requires exponential time. But in practice on graphs encountered in everyday applications, the time complexity is tractable. However, can we do better?

In an attempt to improve upon the performance of Ullmann's algorithm for subgraph isomor-

Input: $G_1 = (V_1, E_1), G_2 = (V_2, E_2)$, the two graphs being tested.
Output: true if the graphs are found to be isomorphic or false otherwise.

```
GSIC(G1, G2)
(1)  M = INITGSIC(G1, G2)
(2)  if REFINE2(M, G1, G2, H) = 0
(3)      return false
(4)  else
(5)      if LENGTH(H) = 0
(6)          return true
(7)      else
(8)          H = SORT(H)
(9)          return DFSGSIC(M, G1, G2, H )
```

Figure 6: The Greedy Subgraph Isomorphism Checker (GSIC) Algorithm. GSIC is a variation of Ullmann's Algorithm that includes a more extensive initialization phase and a greedy heuristic search.

phism, we describe the *Greedy Subgraph Isomorphism Checker* (GSIC). GSIC is a variation of Ullmann's algorithm that incorporates a greedy heuristic choice within the depth-first search of Ullmann's algorithm. It includes some other modifications as well. GSIC is described in Figure 6 and Figure 7.

The first of these modifications is in the initialization of the matrix M. Ullmann's algorithm initializes this matrix solely on the basis of the degrees of the nodes and the attributes of the nodes. GSIC will instead initialize M based on *n-Region Density*. The n-region density of a node v is the number of nodes reachable from v along a path no longer than n. If the graphs are of the same size and isomorphism is being tested then for all $n = 1, 2, \ldots, N-1$ the n-region density must be the same for any two nodes that are mapped to each other. And for subgraph isomorphism testing, a node in the larger graph must have at least as many nodes in its n-region density as does the node in the smaller graph to which it is mapped for all $n = 1, 2, \ldots, N-1$. The n-region density for all n and for all nodes of a graph is easily calculated in $O(N^3)$ time, where N is the number of nodes in the graph, using a calculation of "all pairs shortest paths". Also incorporated into the initialization stage is the sum of the degrees of the adjacent nodes of a node. For isomorphism testing this value must be equal and for subgraph isomorphism testing this value for a node in the smaller graph must be no larger than that of a node in the larger graph to which it is mapped. This new initialization stage is referenced in line 1 of Figure 6.

The next modification is the use of a greedy heuristic choice within the depth-first search procedure. Figure 7 shows this modification. Rather than choosing a node binding arbitrarily, GSIC chooses a node binding that produces an M with the fewest unbound nodes which can be accomplished algorithmically with the use of a priority queue of M matrices as seen in Figure 7. To calculate the heuristic, the refinement procedure of Ullmann's algorithm is modified such that both refinement and this calculation of the heuristic are computed simultaneously. The fourth parameter of the calls to "REFINE2" in line 12 of Figure 7 and line 2 of Figure 6 is an output parameter that contains a list of not yet bound nodes of graph G_1.

The complexity of GSIC in the worst case is still exponential and occurs under the same conditions as the worst case of Ullmann's algorithm. GSIC does however suffer from a higher complexity initialization stage. To calculate the neighbor degree sums, if the graph was completely

Input: $G_1 = (V_1, E_1), G_2 = (V_2, E_2)$, the two graphs being tested. M is the mapping matrix M previously described. H is a sorted list of the not yet bound nodes of the smaller graph G_1.
Output: true if the graphs are found to be isomorphic or false otherwise.

```
DFSGSIC(M, G1, G2, H)
(1)  S = CREATESTATE(M, H)
(2)  h(S) = LENGTH(H)
(3)  INITPQUEUE(Q)
(4)  ADDTOPQUEUE(Q, S, h(S))
(5)  while NOTEMPTY(Q)
(6)      S = REMOVEMIN(Q)
(7)      M = GETM(S)
(8)      H = GETH(S)
(9)      forall v2 in V2 do
(10)         if M[H[h(S) − 1], v2] = 1
(11)             Mnew = BIND(H[h(S) − 1], M, v2)
(12)             if REFINE2(Mnew, G1, G2, Hnew) = 0
(13)                 DO NOTHING
(14)             else
(15)                 if LENGTH(Hnew) = 0
(16)                     return true
(17)                 else
(18)                     Snew = CREATESTATE(Mnew, Hnew)
(19)                     h(Snew) = LENGTH(Hnew)
(20)                     ADDTOPQUEUE(Q, Snew, h(Snew) )
(21) return false
```

Figure 7: The greedy heuristic depth-first search of the GSIC Algorithm.

connected, would take $O(N)$ time. To compute the n-region density, you must first compute all pairs shortest paths in $O(N^3)$ time. The initialization stage is therefore $O(N^3)$. However, this added complexity during initialization serves the purpose of reducing the branching factor of the search. This, in combination with the greedy heuristic, is why GSIC outperforms Ullmann's algorithm.[4]

3.3.2 MDG Approximate Matching Algorithm

In searching for the LCS, first, arbitrarily choose an initial mapping between the nodes of the two graphs (i.e., for each node of G_1 choose at random a node of G_2 such that no two nodes of G_1 are mapped to the same node of G_2). Next, swap the mappings of the two nodes that reduce the value of the evaluation function the most. If there is no swap that reduces the value of the evaluation function, but there are swaps that result in the same value (i.e., a plateau has been reached), choose one of those at random. The algorithm ends when either every possible swap increases the value of the evaluation function or it makes P random moves on the plateau. Values of P ranging from

[4]See (Cicirello, 1999) for detailed computational comparison of Ullmann's and GSIC on randomly generated pairs of graphs as well as on MDGs.

Input: $G_1 = (V_1, E_1), G_2 = (V_2, E_2)$, the two graphs being tested. P is the number of moves to make on a plateau before giving up.
Output: $H = 0$ if the LCS is found to correspond to a subgraph isomorphism. Otherwise, H is returned where H is the number of mismatched edges when the algorithm halts.

```
LCSGRADIENTDESCENT(G1, G2, P)
(1)  Pairings = GETRANDOMPAIRINGS(G1, G2)
(2)  i = 0
(3)  BestResult = H(G1, G2,Pairings)
(4)  while (BestResult > 0) ∧ (i < P)
(5)      if H(G1, G2,APPLYSWAP(Pairings,BestSwap)) <
         BestResult
(7)          Pairings = APPLYSWAP(Pairings,BestSwap)
(8)          i = 0
(9)          BestResult = H(G1, G2, Pairings)
(10)     else
(11)         if H(G1, G2, APPLYSWAP(Pairings,BestSwap)) = BestResult
(12)             Pairings =
                 APPLYSWAP(Pairings, BestSwap)
(14)             i = i + 1
(15)         else
(16)             i = P
(17) return BestResult
```

Figure 8: Iterative improvement search algorithm for the Largest Common Subgraph (LCS) problem. It begins with a random initial node mapping and iteratively improves the mapping via pairwise exchanges.

constant values to $P = |V_1|^2$ (where V_1 is the vertex set of the smaller graph) have been tried.

Evaluation Function: The evaluation function is the count of the number of mismatched edges. That is, the evaluation function, $H = |E|$ such that $G_1 = (V_1, E_1)$ is the smaller of the two graphs being compared, $G_2 = (V_2, E_2)$ is the larger of the two graphs, and $E = \{(u, v) \in E_1$ such that $(((\text{paired}(u), \text{paired}(v)) \notin E_2 \wedge (\text{paired}(v), \text{paired}(u)) \notin E_2)) \vee \text{inconsistent}(u, \text{paired}(u)) \vee \text{inconsistent}(v, \text{paired}(v))$. The function paired$(x)$ above returns the node $y \in V_2$ that is currently mapped to the node $x \in V_1$. The predicate inconsistent(x, y) is true if nodes x and y are inconsistent in terms of their attributes and false otherwise. In the experiments considered in this paper, node consistency is based only on the type of features the nodes represent. Additionally, this can include domain knowledge such as tolerances, manufacturing attributes, surface finish specifications, etc.

Similarity Measure: As a measure of similarity employ the value $H^* = \frac{min\{H_1,\ldots,H_n\}}{|E_1|}$ where $H_1, \ldots, H_n$ are the final values of H from up to n random restarts of the algorithm and E_1 is the edge set of the smaller graph. A similarity measure of 0 implies that the smaller MDG is subgraph isomorphic to the larger MDG or equivalently that the LCS is the smaller MDG. This similarity measure may vary between 0 and 1. 0 is most similar and 1 is least similar.

The node attributes may contain as little or as much information as one chooses. For the ex-

Input: $G_1 = (V_1, E_1), G_2 = (V_2, E_2)$, the two graphs being compared. R is the number of restarts.
Output: $S = 0$ if the smaller of the two graphs is the largest common subgraph. Otherwise, S is returned where S is the smallest result of R restarts of LCSGradientDescent divided by the number of edges in the smaller of the two input graphs.

SIMILARITY(G_1, G_2, R)
(1) $i = 0$
(2) **if** GSIC(G_1,G_2)
(3) **return** 0
(4) `BestResultThusFar` = LCSGRADIENTDESCENT(G_1, G_2, P)
(5) **while** (`BestResultThusFar` > 0) $\wedge$ ($i < R$)
(6) `BestResultThusFar` = min {`BestResultThusFar`, LCSGRADIENTDESCENT(G_1, G_2, P) }
(7) $i = i + 1$
(8) **return** $\frac{BestResultThusFar}{\min\{|E_1|,|E_2|\}}$

Figure 9: The similarity assessment algorithm. This algorithm calls both GSIC and LCSGradient-Descent.

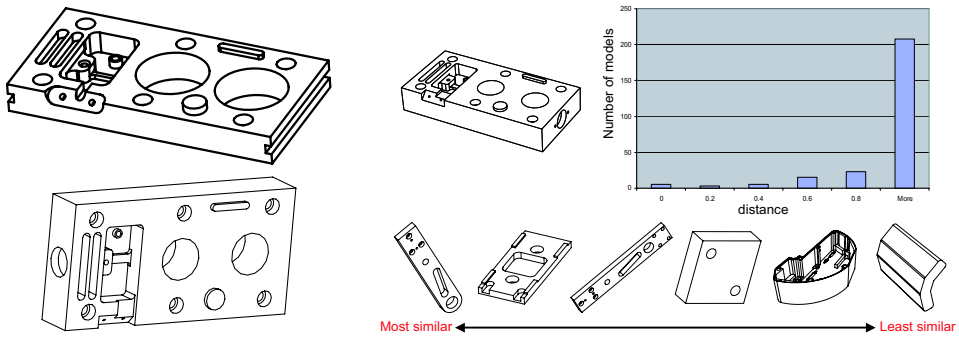
periments that are described later, the node labels were simply the type of feature, such as "hole" or "pocket". However, by incorporating more information into the node labels such as dimensions or orientation, we can further restrict allowable mappings which will increase the algorithm's performance by reducing the search space. Incorporating more information in the node attributes will also obtain a more meaningful similarity measure. For example, by incorporating a notion of dimension into the attributes then a really large block with a tiny hole will not be found similar to a little block with a larger hole.

Figure 8 is the algorithm developed and described for computing the largest common subgraph using iterative improvement. In the algorithm, `Pairings` refers to the mapping between the nodes of the two graphs. And GETRANDOMPAIRINGS returns a random mapping as described above. H is the evaluation function that counts the number of mismatched edges given two graphs and a mapping between the nodes in these two graphs. `BestSwap` is the swap from the set of all possible swaps between pairings that results in a mapping with the smallest value for H. APPLYSWAP returns the mapping that results from applying the given swap to the given mapping.

The algorithm is of polynomial time complexity. It takes $O(N^2)$ time to choose the best swap. In the worst possible case, by choosing the best swap at each step the evaluation function is simply reduced by one and therefore can look for the best swap as many as $|E|$ times. It takes time in $O(|E|)$ to compute the evaluation function. Also in this worst case, the algorithm reaches a plateau as often as possible and takes P random moves on each of these plateau before finding the swap that reduces the evaluation function. Therefore the worst case complexity of the algorithm is $O(P * E^2 + P * E * N^2)$. If P is a constant then the complexity is simply $O(E^2 + E * N^2)$. Note that this complexity is greatest when a graph is fully connected, making the first term redundant, and thus can be further simplified to $O(E * N^2)$.

To obtain a similarity measure, the smallest result of r executions of this algorithm is divided by the number of edges in the smaller of the graphs. Figure 9 is the random restart algorithm for similarity assessment. The worst case complexity is exponential due to the call to "GSIC" at the start, but in practice this step is performed very quickly as discussed previously and the bulk of the computation time is spent in the r restarts of the iterative improvement search. If one is worried

about the potentially exponential time call to GSIC, then it should be noted that it is possible to handle this by keeping track of the number of backtracks in the GSIC algorithm and abandoning the GSIC call if some threshold number of backtracks is exceeded without determining whether or not a subgraph isomorphism exists. Depending on how such a threshold is set, it is possible to ensure an overall complexity for the Similarity algorithm of $O(E * N^2)$.



(a) TEAM and TEAM-2

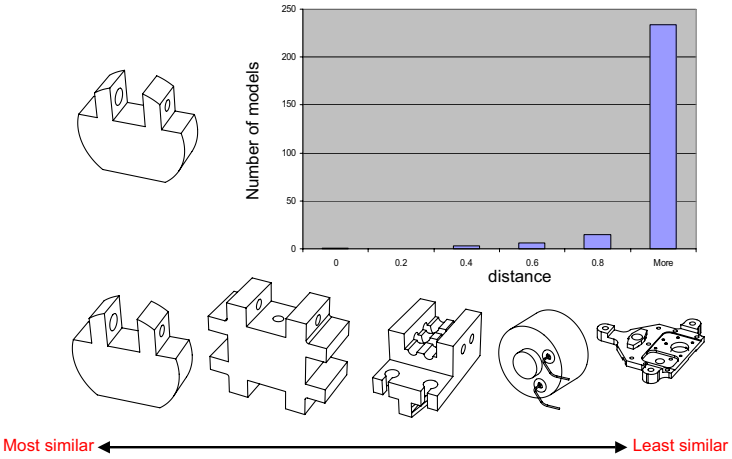
(b) Query Results

Figure 10: Two of the test parts from the DOE TEAM Project. Note that, while they appear very similar, they have variations in features, geometry and topology—as well as orientation with respect to the world coordinate system. Both of these parts are available from the National Design Repository.

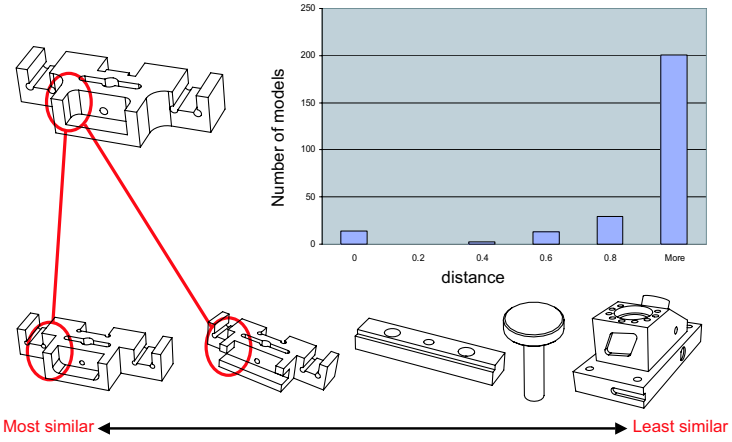
4 Empirical Results

We ran the FBMach feature recognizer over a set of 259 solid models for real world and realistic machined parts chosen from the National Design Repository. The feature output was used to generate MDGs for each of the models—these MDGs formed the index against which we performed queries. The parts were selected based on their diversity and that they were machined parts, many of which were from industry. We conducted several "query by example" experiments, selecting a representative solid model as a target and letting our matcher estimate the distance from the other models in the dataset to the target. Queries consisted of a linear sweep ($O(n)$) across all of the 259 models in the dataset.

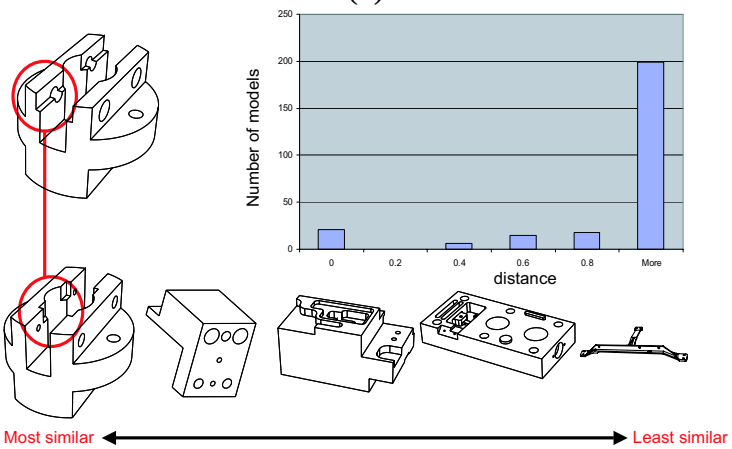
Testbed Description: The National Design Repository. The National Design Repository (Regli and Gaines, 1997b; Regli and Gaines, 1997a) is a digital library of Computer-Aided Design (CAD) models and engineering designs from a variety of domains. The objective is to further the state-of-the-art in academic and industrial research in Computer-Aided Engineering by building a public catalog of real-world design examples. The Repository provides benchmark designs in a variety of formats to serve as reference data to aide developers and students.



(a)



(b)



(c)

Figure 11: Three additional examples: (a) The Simple Bracket example, originally from (Gupta and Nau, 1995). When used as a query, the query processor identifies several parts, including variations on the bracket, in the "most similar" category; (b) The Bracket example: variations on the bracket appear in the near hits buckets; and (c) The Socket example—a variation on the socket appears in the nearest hit bucket.

The Repository currently contains over 55,000 files maintained in multiple data file formats.[5] Contributions have been made by many major research laboratories, companies and academic institutions. Currently there are 10 gigabytes of CAD models and related information. All data is freely available for users around the world. More information is available at `http://www.designrepository.org`.

How to interpret the histograms and model matching results. Figures 10 and 11 give histograms of the results of running a comparison of the query model (shown in the upper-left of each Figure and Subfigure) against the MDGs of all of the other models indexed by FBMach in the Design Repository. Reading the histograms from left-to-right, the parts on the left are more similar to the query and the parts in the right most buckets are the least similar from a machining standpoint.

The distance between solid models is measured as the minimal percentage of mismatched edges as calculated over the course of several runs of the LCS approximate MDG matcher. Referring to the histograms, the vertical axis (Y axis) is a count of the number of solid models falling into each one of the six buckets; the horizontal X axis is the percentage of mismatched MDG edges (i.e., the ratio of mismatched edges to total number of edges in the smaller of the two MDGs). For example, referring to Figure 10 (b), there were 5 models whose MDG's exactly matched (or were embeddable in) that of the query model; 3 models with 20% or fewer mismatched edges; 5 models with 21%-to-40% mismatched edges; 15 with 41%-60% mismatches; 23 models with 61%-to-80%; and 208 models with greater than 80% mismatches. In this way, the histograms show a partition of the parts into groups based on the estimated distance between their MDG and the MDG of the query object.

The comparisons, as noted in Section 3, amounts to a subgraph isomorphism check and distance measure, the CPU time needed to execute these checks were quite reasonable—often running in real time. We hypothesize this performance is achieved because we are operating on labeled graphs and that the iterative improvement approach allows us to control the duration of the search. In this class of experiments, we ran the search with 10 random restarts and took the best matches of the 10 trials. Specific CPU times from a Sun UltraSPARC 30 workstation are noted below with each example.

Four Example Queries. Four (4) of the parts in the Repository were selected as query models. Their selection was based on the knowledge that, in each case, there were models existing in the Repository which indeed would have similar process plans.

TEAM Part: Figures 10 (a) and (b) show the two TEAM Parts, from the US Department of Energy's Technologies Enabling Agile Manufacturing (TEAM) Project. Note that they are both three axis machined parts with minor variations in features, geometry and topology. When TEAM2 was used as a query (Figure 10 (b)), TEAM (Figure 10 (a)) and several other three axis machined parts with similar features and topology appear in the near hit category. For this query, it took 304.71 CPU seconds to search across all 259 models. The GSIC stage of the algorithm took 1.48 CPU seconds so almost instantly we were able to find the parts

[5] This includes solid models in STEP AP 203, ACIS `.sat`, Autodesk `.dxf` and `.dwg`, IGES, Bentley `.dgn`, Parasolid `.xmt` and other formats.

in the most similar partition of the Repository. GSIC found 5 parts for which a subgraph isomorphism exists.

Simple Bracket: Figure 11 (a) shows a simple bracket, a part with about a dozen possible machining features that can be made in three or four set ups (depending on tolerance constraints). The histogram shows the result of comparing the features in this model to those of the others in our Repository. For this query, it took 5.4 CPU seconds to search across all 259 models (0.08 seconds for the GSIC stage of the algorithm). GSIC only found a single part for which a subgraph isomorphism exists. The hill climber, however, found 3 parts in the next most similar category.

Bracket: Figure 11 (b) shows a more complex bracket structure. The histogram of results from this query show two variations on the bracket appearing in the left-most classes of models. Note that these variations change the setups and the feature types (drilled holes to machined slots) but have similar shape properties. Additionally, there are two other variations on the bracket (not shown here) that are found in the left-most buckets. Further note that the vast majority of models fall into the least similar categories. For this query, it took 21.27 CPU seconds to search across all 259 models (0.23 seconds for the GSIC stage of the algorithm). GSIC found 14 parts for which a subgraph isomorphism exists.

Socket: Figure 11 (c) shows the Socket example, consisting of 22 machining feature instances and machinable in four setups. Note that a near duplicate of the socket appears in the most similar category. For this query, it took 95.44 CPU seconds to search across all 259 models (0.49 seconds for the GSIC stage of the algorithm). GSIC found 21 parts for which a subgraph isomorphism exists.

Discussion. In each of the four experiments, the results returned do indeed correspond, subjectively, to our intuition about the machining process. Particularly noteworthy is that for each query, all variations of the query part contained in the database appear in the nearest hits buckets. Also, on average for a query, well over 80% of the database appears in the least similar category. We should expect such a large quantity of parts to be dissimilar to a query part given the diversity of data sources contained in the National Design Repository. This shows that our approach is capable of finding the obviously structurally similar parts (i.e., the variations on the query part) while simultaneously filtering out the vast majority of non-similar parts. The current experiments exploit feature parameters, dimensions and other geometry and shape information. In reality, however, tolerance information will be critical to consider. We see the above results as a proof-of-concept, hinting that tolerance and other more sophisticated engineering knowledge could be incorporated into the approach to great benefit—such as to reduce the search-space size and to require matches to be more exact.

Validation of this work, thus far, has only been empirical. The Design Repository, while it contains thousands of CAD models, few of these models contain tolerances or process plan information—most are unattributed boundary representations. Additional work needs to be done to properly and objectively evaluate the information content and scalability of these and other solid model retrieval techniques.

5 Conclusions and Future Research Directions

This paper presented our approach to using machining features as an index-retrieval mechanism for storing solid models. We believe that this work represents the first fully automated technique for machining feature-based comparisons of machined artifacts. Our hope is that this type of approach, combining raw b-Rep data with machining feature knowledge, will enable radical changes in the way in which design data is managed in modern engineering enterprises.

Our approach enables us to interrogate large databases of solid models and perform queries based on manufacturing similarities among the artifacts. While we have not yet performed a comprehensive analysis of the manufacturing data for each of the parts in our Design Repository, our empirical results using a subset of this dataset suggest that this is a promising approach to information and data management for design and manufacturing process knowledge.

Research contributions include both our overall approach to the problem (i.e., depicted in Figure 1) and the specific elements we used to realize it: formalization of the model dependency graphs and the new heuristic search techniques we developed to compare them. We believe that these data structures and algorithms will have applicability beyond their original use in this project. Some of the areas we are current exploring for future work include:

A Model Indexing and Query Scheme: We showed that the MDG is a useful mechanism for the indexing and retrieval of models in CAD databases and can be employed using a "query by example" paradigm. Using algorithms for computing the largest common subgraph, and introducing some engineering domain knowledge, we have created a general technique for indexing large numbers of solid models and retrieving them based on the similarity of their machining features. We believe that this technique can be refined and will have impact on how CAD data is stored and managed.

Variational Process Planning Search Engine: Based on the MDG, one can create query artifacts that partition the database of solid models into different morphism classes—based on how similar in structure each model is to the query model. We believe that this approach can be refined to detect meaningful part classes and families in large sets of engineering models. This can form the basis for more intelligent Product Data Management (PDM) systems and tools for variational design and variant process planning.

Process Selection and Cost Estimation Engine: Corporate Design and Manufacturing databases often store manufacturing cost and process data. With the techniques presented, we can design Active CAD Agents that assist designers during detailed design by comparing the in-process design artifact to those previously created. In this way, the agent can provide feedback on potential manufacturing process choices (e.g., when a design that is categorized as similar was made with stereolithography, perhaps stereolithography should be considered for the new design) and expected cost (e.g., using data from the existing process plan for similar parts).

Currently, our research considers only plain, unattributed solid models—there are no tolerances, manufacturing attributes, surface finish specifications, etc. In the future, we believe that additional domain knowledge can be used to refine our techniques. Information about engineering tolerances, surface finishes, constraints and parametric, etc. all can be used to augment the basic

techniques presented here. Such information can be used to further prune the search-space by further limiting which features can be matched by the MDG comparison algorithm. For example, currently a feature of type “A” in one model can be matched to any feature of type “A” in a second model. The incorporation of additional attributes can limit which features of type “A” are allowed to match. In addition to reducing the search-space size and thus the computation time required by the search, this added information can also be used to allow the user to specify how close of a match they are interested in and which attributes are important to the user. Further, we would like to explore how to implement multiple feature views onto the Repository by including feature data generated by different feature recognition techniques.

Acknowledgements. Our thanks are extended to Dr. Steve Brooks of Honeywell, Federal Manufacturing Technologies Program, in Kansas City for providing the National Design Repository with the ACIS models for the TEAM parts. This work was supported in part by National Science Foundation (NSF) Knowledge and Distributed Intelligence in the Information Age (KDI) Initiative Grant CISE/IIS-9873005; CAREER Award CISE/IIS-9733545 and Grant ENG/DMI-9713718. Additional support is being provided by the Office of Naval Research under award N00014-01-1-0618. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation or the other supporting government and corporate organizations.

References

Allada, V. and Anand, S. (1995). Feature-based modelling approaches for integrated manufacturing: State-of-the-art survey and future research directions. *International Journal of Computer Integrated Manufacturing*, 8(6):411–440.

Almohamad, H. A. and Duffuaa, S. O. (1993). A linear programming approach for the weighted graph matching problem. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(5):522–525.

Bhanu, B. (1982). Surface representation and shape matching of 3-d objects. In *PRIP82*, pages 349–354.

Britanik, J. and Marefat, M. (1995). Hierarchical plan merging with applications to process planning. In *International Joint Conference on Artificial Intelligence*.

Brooks, S. L. and Greenway Jr., R. B. (1995). Using STEP to integrate design features with manufacturing features. In Busnaina, A. A., editor, *ASME Computers in Engineering Conference*, pages 579–586, New York, NY 10017. ASME.

Brooks, S. L. and Wolf, M. L. (1994). Overview of Allied Signal's XCUT system. In Shah, J., Mäntylä, M., and Nau, D., editors, *Advances in Feature Based Manufacturing*, pages 399–422. Elsevier/North Holland.

Cicirello, V. and Regli, W. C. (1999). Resolving non-uniqueness in design feature histories. In Anderson, D. and Bronsvoort, W., editors, *Fifth Symposium on Solid Modeling and Applications*, New York, NY, USA. ACM, ACM Press. Ann Arbor, MI.

Cicirello, V. A. (1999). Intelligent retrieval of solid models. Master's thesis, Drexel University, Geometric and Intelligent Computing Laboratory, Department of Mathematics and Computer Science, Philadelphia, PA 19104.

Cicirello, V. A. and Smith, S. F. (2000). Modeling GA performance for control parameter optimization. In *GECCO-2000: Proceedings of the Genetic and Evolutionary Computation Conference*.

Cohen, E. and Wolfson, H. (1994). Partial matching of 3-d objects in cad/cam systems. In *ICPR94*, pages A:483–487.

Cohen, K. D. (1996). Feature extraction and pattern analysis of three-dimensional objects. Master's thesis, Dartmouth College, Thayer School of Engineering, Hanover, NH.

Cybenko, G., Bhasin, A., and Cohen, K. D. (1996). 3d base: An agent-based 3d object recognition system. http://comp-eng-www.dartmouth.edu/$\tilde{3}$d.

de St. Germain, H. J., Stark, S. R., Thompson, W. B., and Henderson, T. C. (1996). Constraint optimization and feature-based model construction for reverse engineering. In *Proceedings of the ARPA Image Understanding Workshop*.

Elinson, A., Nau, D. S., and Regli, W. C. (1997). Feature-based similarity assessment of solid models. In Hoffman, C. and Bronsvoort, W., editors, *Fourth Symposium on Solid Modeling and Applications*, pages 297–310, New York, NY, USA. ACM, ACM Press. Atlanta, GA.

Gaines, D. M., Castaño, F., and Hayes, C. C. (1999). MEDIATOR: A resource adaptive feature recognizer that interleaves feature extraction and manufacturing analysis. Transactions of the ASME, Journal of Mechanical Design, 121(1):145–158.

Gaines, D. M. and Hayes, C. C. (1999). Custom-cut: A customizable feature recognizer. *Computer-Aided Design*, 31(2):85–100.

Gaines, D. M. and Hayes, C. C. (2000). Simultaneous identifi cation of planning subgoals and construction of operator instantiations for more flexible planning. *Artificial Intelligence for Engineering Design, Analysis, and Manufacturing (AIEDAM)*, 14(4):305–322.

Garey, M. R. and Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 41 Madison Avenue, New York, NY 10010.

Gupta, A. and Jain, R. (1997). Visual information retrieval. *Communications of the ACM*, 40(5):71–79.

Gupta, S. K., Kramer, T. R., Nau, D. S., Regli, W. C., and Zhang, G. (1994). Building MRSEV models for CAM applications. *Advances in Engineering Software*, 20(2/3):121–139. *Special issue on Feature-Based Design and Manufacturing*.

Gupta, S. K. and Nau, D. S. (1995). A systematic approach for analyzing the manufacturability of machined parts. *Computer Aided Design*, 27(5):323–342.

Gupta, S. K., Regli, W. C., and Nau, D. S. (1995). Manufacturing feature instances: Which ones to recognize? In Rossignac, J., Turner, J., and Allen, G., editors, *Third Symposium on Solid Modeling Foundations and CAD/CAM Applications*, pages 141–152, New York, NY, USA. ACM SIGGRAPH and the IEEE Computer Society, ACM Press. Salt Lake City, Utah.

Han, J. (1997). On multiple interpretations. In *4th ACM SIGGRAPH Symposium on Solid Modeling and Applications*, pages 311–321. ACM Press. Atlanta, Georgia.

Han, J. and Requicha, A. A. G. (1994). Incremental recognition of machining features. In Ishii, K., editor, *ASME Computers in Engineering Conference*, pages 143–150. ASME.

Han, J. and Requicha, A. A. G. (1995). Integration of feature-based design and feature recognition. In Busnaina, A. A., editor, *ASME Computers in Engineering Conference*, pages 569–578, New York, NY 10017. ASME.

Han, J.-H., Regli, W. C., and Pratt, M. J. (2000). Algorithms for feature recognition from solid models: A status report. *IEEE Transactions on Robotics and Automation*, 16(6):782–796.

Haralick, R. M. and Shapiro, L. G. (1985). Image segmentation techniques. *Computer Vision, Graphics, and Image Processing*, 29:100–132.

Ji, Q. and Marefat, M. M. (1997). Machine interpretation of cad data for manufacturing applications. *Computing Surveys*, 29(3):264–311.

Marefat, M. and Kashyap, R. L. (1990). Geometric reasoning for recognition of three-dimensional object features. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(10):949–965.

Marefat, M. and Kashyap, R. L. (1992). Automatic construction of process plans from solid model representations. *IEEE Transactions on Systems, Man, and Cybernetics*, 22(5):1097–1115.

Marefat, M., Malhotra, S., and Kashyap, R. L. (1993). Object-oriented intelligent computer-integrated design, process planning, and inspection. *IEEE Computer*, pages 54–65.

Messmer, B. and Bunke, H. (1996). Fast error-correcting graph isomorphism based on model precompilation. Technischer Bericht IAM-96-012, Institut für Informatik, Universität Bern, Schweiz.

Ramesh, M. M., Yip-Hoi, D., and Dutta, D. (2000). A decomposition methodology for machining feature extraction. In *ASME Design Engineering Technical Conferences, Computers in Engineering Conference*, New York, NY, USA. American Assocation of Mechanical Engineers, ASME Press. DETC2000/CIE-14645.

Regli, W. C. and Cicirello, V. (2000). Managing digital libraries for computer-aided design. *Computer Aided Design*, 32(2):119–132. Special Issue on *CAD After 2000*. Mohsen Rezayat, Guest Editor.

Regli, W. C. and Gaines, D. M. (1997a). A national repository for design and process planning. In *National Science Foundation Design and Manufacturing Grantees Conference*, pages 673–674, Seattle, Washington. January 7-10.

Regli, W. C. and Gaines, D. M. (1997b). An overview of the NIST Repository for Design, Process Planning, and Assembly. *Computer Aided Design*, 29(12):895–905.

Regli, W. C., Gupta, S. K., and Nau, D. S. (1995). Extracting alternative machining features: An algorithmic approach. *Research in Engineering Design*, 7(3):173–192.

Regli, W. C., Gupta, S. K., and Nau, D. S. (1997). Toward multiprocessor feature recognition. *Computer Aided Design*, 29(1):37–51.

Regli, W. C. and Pratt, M. J. (1996). What are feature interactions? In McCarthy, J. M., editor, *ASME Design Engineering Technical Conferences, Design for Manufacturability Symposium*, New York, NY. ASME International. 96-DETC:DFM-1285.

Russell, S. and Norvig, P. (1995). *Artificial Intelligence: A Modern Approach*. Prentice Hall. ISBN 0-13-103805-2.

Shah, J., Anderson, D., Kim, Y. S., , and Joshi, S. (2001). A discourse on geometric feature recognition from cad models. *ASME Transactions, the Journal of Computer and Information Science in Engineering*, 1(1):41–51.

Shoukry, A. and Aboutabl, M. (1996). Neural network approach for solving the maximal common subgraph problem. *IEEE Transactions on Systems, Man, and Cybernetics: Part B—Cybernetics*, 26(5):785–790.

Slovensky, L. (1994). Mechanical product definition for process planning using form features. Technical Report ISO/WD 10303-224, International Organization for Standardization. Working draft.

Snead, C. S. (1989). *Group Technology: Foundations for Competitive Manufacturing*. Van Nostrand Reinhold, New York.

Stewman, J. and Bowyer, K. (1990). Direct construction of the perspective projection aspect graph of convex polyhedra. *CVGIP*, 51(1. July 1990):20–37.

Sun, T.-L., Su, C.-J., Mayer, R. J., and Wysk, R. A. (1995). Shape similarity assessment of mechanical parts based on solid models. In Gadh, R., editor, *ASME Design for Manufacturing Conference, Symposium on Computer Integrated Concurrent Design*, pages 953–962. ASME.

Thompson, W. B., Owen, J. C., and de St. Germain, H. J. (1995). Feature-based reverse engineering of mechanical parts. Technical Report UUCS-95-010, The Univeristy of Utah, Department of Computer Science.

Thompson, W. B., Riesenfeld, R. F., and Owen, J. C. (1996). Determining the similarity of geometric models. In *Proceedings of the ARPA Image Understanding Workshop*.

Ullmann, J. R. (1976). An algorithm for subgraph isomorphism. *Journal of the Association of Computing Machinery*, 23(1):31–42.

van Houten, F. J. A. M. (1991). *PART: A Computer Aided Process Planning System*. PhD thesis, University of Twente.

Vandenbrande, J. H. and Requicha, A. A. (1994). Geometric computations for the recognition of spatially interacting manufacturing features. In Shah, J., Mäntylä, M., and Nau, D., editors, *Advances in Feature Based Manufacturing*. Elsevier/North Holland.

Vandenbrande, J. H. and Requicha, A. A. G. (1993). Spatial reasoning for the automatic recognition of machinable features in solid models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(12):1269–1285.

Wang, Y. K., Fan, K. C., and Horng, J. T. (1997). Genetic-based search for error-correcting graph isomorphism. *IEEE Transactions on Systems, Man, and Cybernetics: Part B—Cybernetics*, 27(4):588–597.

Vincent A. Cicirello

Vincent Cicirello is a Ph.D. student in the Robotics Institute at Carnegie Mellon University and expects to complete the Ph.D. by Summer 2003. Vincent received a M.S. in Computer Science and a B.S. in Computer Science and Mathematics from Drexel University. Vincent's research interests include stochastic search, multi-agent / multi-robot systems, planning / scheduling, applied artificial intelligence, and anytime computation. For his dissertation research, he is investigating stochastic sampling in combinatorial optimization domains, tools for their analysis, and algorithms with the ability to use these tools for self-guidance. Vincent is a member of AAAI, ACM, IEEE, IEEE Computer Society, SIAM.

William C. Regli

William Regli is an Associate Professor in the Department of Computer Science at Drexel University. Dr. Regli received the Ph.D. in Computer Science from the University of Maryland at College Park; and has held positions at AT&T Labs, Carnegie Mellon and NIST. Dr. Regli's research address basic computer science problems that arise in design and manufacturing and is supported by NSF, ONR, NIST and DOE. He is recipient of the NSF CAREER Award and is a member of ACM, IEEE Computer Society, AAAI, and Sigma Xi. He has authored or co-authored over 100 technical publications.