Resolving Non-Uniqueness in Design Feature Histories

Vincent A. Cicirello* William C. Regli†

Geometric and Intelligent Computing Laboratory
Department of Mathematics and Computer Science
Drexel University
3141 Chestnut Street
Philadelphia, PA 19104
http://gicl.mcs.drexel.edu/

Abstract

Nearly all major commercial computer-aided design systems have adopted a feature-based design approach to solid modeling. Models are created via a sequence of operations which apply design features to incremental versions of a design model. Even surfacing, free-form surface shaping, and deformation operations are internally represented in modeling systems as features in a "history tree" that generates the final design. Much in the same manner that Constructive Solid Geometry (CSG) trees for an individual model can be non-unique, these design feature histories for solid models might be ordered in a number of ways and still result in the same final geometry and topology.

We formulate this problem symbolically and present geometric reasoning techniques to generate a canonical form for certain classes of design feature histories. We define this representation as a *Model Dependency Graph* (MDG) and show how it can be used as a basis for developing techniques for managing databases of solid models. Using the MDG, we introduce algorithms that can assess the similarity of solid models based on design features. We believe these techniques can be used to build intelligent CAD knowledge-bases and to identify meaningful part families from large sets of designs. Lastly, we describe experimental results and performance metrics for our approach.

Keywords: Modeling Families of Geometric Objects, Feature-based modeling, Computer-Aided Design, Constraint-Based and Parametric Modeling, Engineering Knowledge-bases, Product Data Management

1 Introduction

Nearly all major commercial computer-aided design systems have adopted a feature-based design approach to solid modeling. Models are created via a sequence of operations that apply design features to incremental versions of a design model. Surfacing, free-form surface shaping, and deformation operations are internally represented in modeling systems as features in a "history tree" that generate the final design. However, much in the same manner that Constructive Solid Geometry (CSG) trees for an individual model can be non-unique, these design feature histories for CAD models might be ordered in a number of ways and result in the same final geometry and topology.

One of the goals of this research is to develop algorithmic techniques to manage databases of CAD and Solid Models. As pictured in Figure 1, CAD databases and knowledge-bases are at the core of the modern engineering enterprise. These emerging digital libraries store all information relevant over a product's life-cycle (geometry, topology, features, revisions, etc.). While the lack of standard representation schemes for CAD data and features data is under significant study, little work has been done to address how to handle the great diversity that remains even in a turn-key CAD environment.

In order to efficiently store and retrieve solid models from a CAD knowledge-base, one requires a more uniform representation for the feature information used to describe the artifact. Note that we are not introducing yet another feature library—rather, this paper presents techniques for dealing with ambiguity and variation that are independent of feature definition. Given that, with a fixed feature library, one might be able to design an artifact in several alternative ways, we present techniques to convert these orderings into canonical form that is unique over several classes of variation in the design feature history. Once reduced to a unique form, solid models can be more efficiently hashed or indexed for storage.

We formulate the non-uniqueness symbolically and present geometric reasoning techniques to generate a canonical representation of a CAD model's design feature history. We call this representation a *Model Dependency Graph* (MDG). Based on the MDG, we introduce algorithms that can assess the similarity of solid models based on design features; measure the "distance" between two CAD models; index models for database storage; and identify meaningful part families from large sets of designs, such as are stored in engineering databases. Lastly, we describe experimental results and performance metrics for our approach.

This paper is organized as follows: Section 2 provides an overview of related work in solid modeling and feature-based modeling. Section 3 presents our formulation of the problem of ambiguous design history trees and introduces our approach to addressing it based on constraint and graph algorithms. Section 4 describes our experimental results and Section 5 presents our conclusions and plans for future work.

*URL: http://www.mcs.drexel.edu/∼uvcicire; Email: uvcicire@mcs.drexel.edu.
†URL: http://www.mcs.drexel.edu/∼regli; Email: regli@drexel.edu.

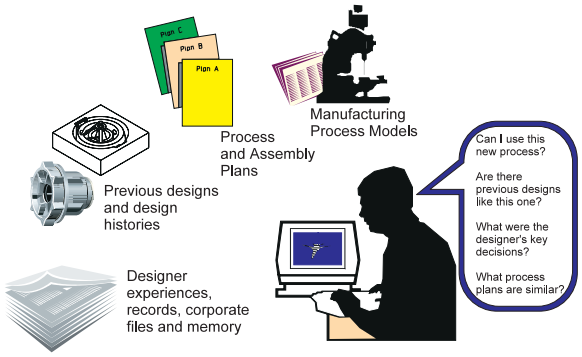


Figure 1: The Design Knowledge-Base Scenario: Engineers accessing libraries of project data to identify ideas and solutions to new problems. One aspect of this problem is how to retrieve similar designs and index CAD databases.

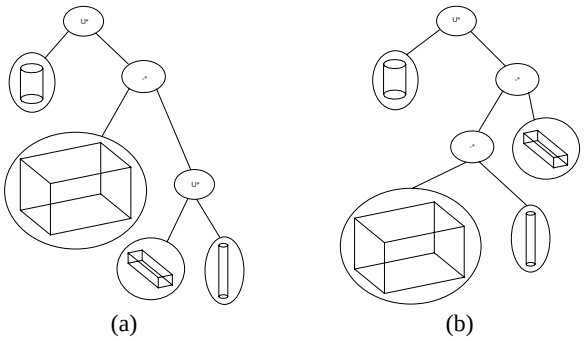


Figure 2: Examples of CSG trees: two different trees that create the same solid model.

2 Background and Related Work

2.1 Constructive Solid Geometry (CSG)

Constructive Solid Geometry (CSG) is a volumetric representation scheme for three-dimensional solid geometric models. Solids are represented as a set-theoretic boolean expression of primitive solid objects, of a simpler structure [8]. Regularized set boolean operations and motion operations are used to represent a composition of primitive geometric shapes. The standard primitives that are used in the CSG representation scheme are the parallelepiped or block, the triangular prism, the sphere, the cone, the cylinder, and the torus [8]. The set boolean operations that may be used are regularized union, regularized difference, and regularized intersection. The regularized boolean set operations are extensions of the typical boolean operations that prevent dangling edges and faces from resulting. A CSG representation of a solid model can be viewed as a tree. The primitive shapes used in representing the solid are the leaves of the tree; the boolean set operations and motion operations are the interior nodes, as shown in Figure 2 (a). A CSG representation of a three-dimensional solid model lacks uniqueness. There may be several ways to represent a single solid model with multiple CSG representations. One solid may be representable by several valid CSG representations, as shown in Figure 2 (b).

2.2 Boundary Representation (B-rep)

A solid can be represented unambiguously by describing its surface and topologically orienting it in such a way so that at any point one can tell on which side the solid interior lies. The boundary representation (B-Rep) consists of a topological description of the connectivity and orientation of the faces, edges, and vertices and a geometric description for embedding these surface elements in space. The vertices, edges, and faces are specified abstractly with their incidences and adjacencies indicated in the topological description. And in the geometric description, the equations of the surfaces of which the faces are a subset are specified [8].

The boundary representation (B-Rep) scheme represents three-dimensional solid objects by a hierarchical description of the faces, edges, and vertices that form the boundary of the model. A face is specified by the edges by which it is bounded. An edge is specified by the curve on which it lies and its vertices are points in three-dimensional coordinate space. The B-Rep representation scheme is unique, unlike that of the CSG representation [9].

2.3 Features

A feature can be defined differently depending on the context in which it is to be used. Machining features may differ from forging features [10]. Features of a solid geometric model are dependent on the use of the model. In the application of machining, some example features are holes, slots, and pockets. Each such feature of the solid model may correspond to some manufacturing procedure or step of the design process.

2.4 Feature Recognition from Three-dimensional Solid Models

Much research has been done in the area of automatic feature recognition from three-dimensional solid models [7, 14, 11, 12, 18]. Although there are other representation schemes, the most commonly used representations in systems that perform automatic feature recognition are the B-Rep and the CSG. This is in part due to the fact that a majority of solid modeling and CAD systems make use of either the B-Rep or the CSG in their representations. Some systems may incorporate both into the representation of the solid models. Boundary representations are often used for rendering and display purposes, while CSG-tree-like structures supply the design history of the operations performed in designing an artifact.

One technique used for feature recognition is through the use of attributed adjacency graphs (AAG) that are generated from the B-Rep of the solid model and is described in detail in [10]. In an AAG, each node represents a face of the solid model. Each edge in the solid model becomes an arc in the AAG where the endpoints are the nodes that represent the faces that share the edge. Each arc in the AAG is attributed and specifies whether the faces corresponding to the edge are concave or convex. The technique for feature recognition described in [10] uses graph-based techniques to search the AAG of the solid model in question for subgraphs which correspond to the AAG representations of primitive elements, and incorporates some special techniques for detecting interacting features. The use of AAGs for feature recognition is limited to polyhedral parts with polyhedral features [10].

Another technique for feature recognition similar to the use of AAG is presented in [11]. The approach presented incorporates what is termed a cavity graph. A cavity graph consists of nodes representing the faces of the solid. Links between two nodes represent nonconvexity of the corresponding faces. And each node is labeled to show the relative orientation of the faces in space. The method proposed uses a hypothesis generation and elimination approach. The hypotheses are generated by decomposing the cavity graph of

the object into maximal subgraphs and searching searching these subgraphs for the known cavity graphs of primitive components. Rule-based methods are used to eliminate incorrect hypotheses and generate new hypotheses. The methods described also incorporate the idea of using "virtual links" to aid in finding interacting features (i.e., additional links are added to the cavity graph) [11].

Convex-hull techniques use volumetric properties of solid models rather than surface features to extract features. The convex-hull technique relies on finding the materials that must be removed from a solid to form the model of the part. The feature extraction process uses convex decompositions. An object is represented as a set of convex components with alternating addition and subtraction of volumes. The convex decompositions are sometimes known as alternating sum of volumes (ASV) [9]. The technique first finds the convex hull of the object and then finds the set difference between the object and its convex hull. The technique is applied recursively to find the full decomposition of the object. The domain of geometric objects that ASV can handle is limited, however, as ASV will not always terminate. The removed volumes also do not always represent features. Volumes that may be shared by two or more interacting features will only be applied to one, for example [9].

In addition to the techniques for feature recognition that use B-rep as input, there exist techniques that use the CSG representation of the solid model to extract the features. These techniques must first overcome the problem that CSG representations are not unique. A single solid model may be representable by several different CSG trees. Another problem is that nearby nodes in the CSG tree do not necessarily correspond to features. In fact, it is possible that the nodes corresponding to one feature may be scattered about the tree. The set difference operation also does not necessarily correspond to the removal of manufacturing material, and some removal operations may be implicit without the use of a set difference operation. Most methods of performing feature recognition from the CSG representation begin by converting it to some other representation first. Although the potential exists for the CSG representation to more closely resemble machining operations, in practice there appears to be a lack of a general relationship between the primitives of a CSG and the features of the design [9].

2.5 Feature-based Modeling and Design

Modeling and design are typically performed by the addition and subtraction of primitive shape components from the solid model. Using this approach, features would be extracted later using a feature recognition process. However, this is not the only approach that may be used. Modeling by using design features is an alternate approach. This is known as feature-based modeling [15, 5].

In [15], it is pointed out that feature-based design has the advantage of keeping relevant information for applications during the design process. It is also pointed out that manufacturing concerns can be considered early in the design process. Using feature recognition, this may not have been possible. A model may have been designed with features that would be difficult to actually manufacture. In feature-based design, functional meaning is assigned to the parts of an object during the design phase rather than during the feature recognition [1].

[13] discusses a combined approach of feature-based design and design recognition. The feature-based design part of the described approach incorporates a feature library, consisting of predefined design features and user-defined design features. The predefined features consist of features such as cylindrical holes, rectangular pockets, and slots. User-defined features can be created by the designer to make up for a deficiency in the features in the library. These user-defined features can be created with either the feature modeler or a solid modeler.

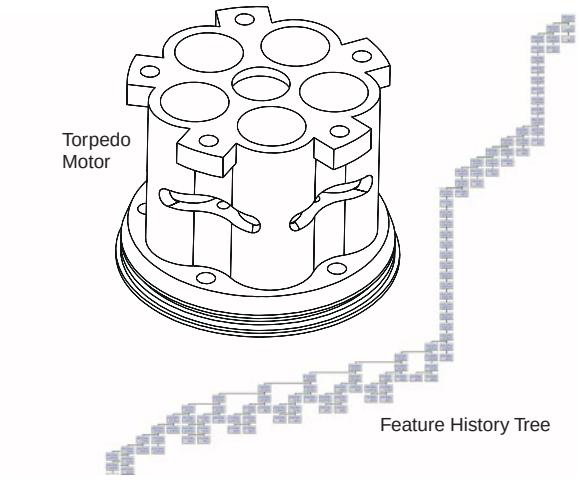


Figure 3: An illustration of a model of a torpedo motor housing and a snapshot of the design feature history tree for the torpedo motor (each box is a feature or operation on the model). This history tree was generated when the motor was modeled using Bentley Systems' MicroStation Modeler. The over one hundred features and operations make the history tree difficult to present in detail—for requiring more detail, this model is available through the National Design Repository at `http://repos.mcs.drexel.edu/`.

3 Problem Statement and Technical Approach

The same artifact may be designed in several different ways. One designer may do things in one order, and another designer in a different order. These different orderings of operations will result in history trees that can be drastically different and yet represent the same thing. For example, Figure 3 shows a solid model for a torpedo motor housing consisting of about one hundred design feature instances. Figure 3 also shows one possible design feature history tree for this model that can be used to define this part in a commercial CAD environment (there may be many other ways of designing this part). The non-uniqueness of the history tree poses a problem as to how to effectively index and retrieve CAD and solid model data based on feature information.

This problem is similar to that of the non-uniqueness of CSG models. In the feature recognition techniques that use CSG models rather than the B-reps, methods of converting the CSG to another representation or an ordered representation are usually incorporated to get around this disadvantage [16]. We will incorporate similar techniques in our use of history trees for similarity comparisons.

Some operations performed on the design are dependent on other previous operations and must be performed in a specific order. For example, it is not possible to create a hole in a block that does not yet exist. But there may be other operations that may be independent of all other operations that have been performed on a design. For example, if we had a block and wanted to subtract out a hole in one side and create a slot in the other side with no interaction occurring between this slot and hole, then it would make no difference to the final product which of these two operations occurred first.

The history tree is initially sorted in the relative order of when each operation was performed in time relation to each other. This ordering has no bearing on the order of operations performed during the manufacturing of the component. For example, a designer may take a block and subtract out a hole in one side and then a hole in the other side. The designer may then subtract out a second hole

in the first side. When the manufacturing plan is later devised by a machinist, he or she may decide that the two holes on the first side can be drilled at the same time, rather than following the exact steps taken in the design phase.

Hence, to retrieve engineering data from knowledge-bases using the design history as part of the retrieval probe, it becomes necessary to transform the design history tree in such a way that it is now ordered solely on the basis of dependencies rather than on an order based on temporal position.

3.1 Problem Formalization

A design, D, is defined as a tuple $D = \langle T, P, F \rangle$ where: T is the history tree of the artifact (a representation of the steps taken in the design phase of the modeling process); P is the geometric and topological model of the artifact (including the boundary representation of the component); F is a finite set $F = \{f_0, \ldots, f_n\}$ of design features. In the context of this paper, T can be thought of as a type of CSG tree and design features are local or global operations on part geometry.

The boundary representation (B-rep) of a component (or part), P, is a representation of the geometric and topological model of the artifact. T is another representation of the geometric model—related in some way to the steps that were involved in the design. Most CAD and three-dimensional geometric/solid modeling packages use either B-rep or CSG representations, and in some cases both. There also exist techniques for converting (1) CSG to B-Rep; (2) certain classes of solids from B-Rep and CSG; and (3) feature identification from solid models. Hence, history information is either readily available or can be produced, to a degree, via automated feature identification techniques [17, 6, 16].

The set F of design features is a finite set defined as $F = \{f_0, \ldots, f_n\}$. These design features can include any volumetric or surface operation typically used in a commercial CAD environment (i.e., holes, pockets, slots, bosses, etc.). A history tree, T, can be either a tree structured representation of the design phase of an engineering artifact or it can simply be a linear ordering of the steps taken to design the artifact. The nodes of this tree represent the primitive elements added to or subtracted from the component during the design phase, such as blocks, cones, and cylinders, operations on those primitives, and operations on the component as a whole entity. Some of the possible operations include blends, chamfers, fillets, extrusions, contouring, and free-form surface modeling.

3.2 Approach

We divide the problem into two phases:

Phase I: Defining a structure called a *Model Dependency Graph* using the history tree data;

Phase II: Creating algorithms for comparing models based on their Model Dependency Graphs.

3.2.1 Phase I: Model Dependency Graphs

As discussed earlier, design history trees, like CSG trees, are non-unique. That is, for a given solid model, there may be several ways to design it and result in the same final product, and thus there may be different design history trees that represent the same design.

To deal with this problem, we create a **Model Dependency Graph** (MDG). This graph is a directed acyclic graph which has some unique characteristics. The model history,[1] M, is defined as $M = \{m_0, \ldots, m_n\}$. The m_i is the complete model at stage i of the design. That is, m_i represents the solid model after feature f_i is applied to the model. There is an ordering inherent in the design history graph. In the case where it is not clear which operation or feature came before the other we impose a left-to-right ordering on the operations. The m_i may be generated and stored at design time. Or they may be easily generated from the design history. Let $vol(f_i)$ represent the "solid" volume that is either added or removed from the complete model by the application of feature f_i.

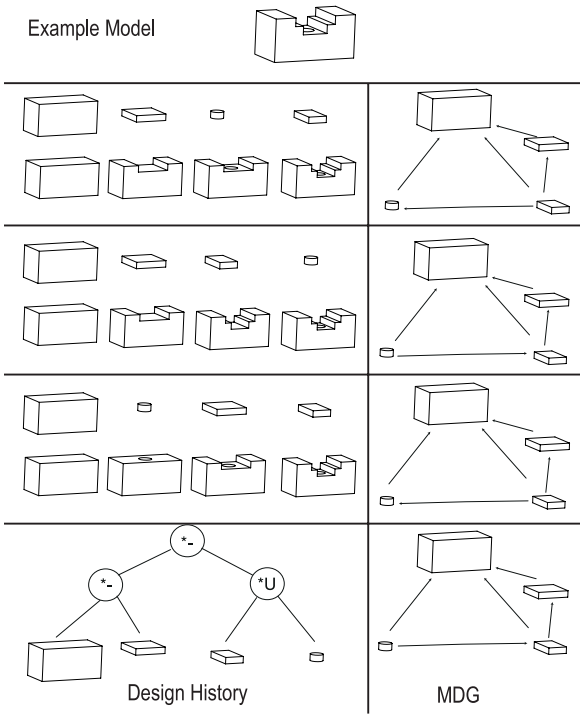


Figure 4: Pictured is a single solid model and several alternative design feature histories, and one possible CSG tree, that can produce it. On the right are the MDGs for each of these alternatives—note that they are all D-morphic to one another.

Definition 1: *Model Dependency Graph* - basic definition

A *Model Dependency Graph* (MDG) is defined as $G = (V, E)$. The vertex set is defined as $V = \{f_0, \ldots, f_n\}$. The indices on the f_i represent the order that the features were applied during the design process. The edge set can be defined as $E = \{(f_i, f_j)$ such that $i > j, vol(f_i) \cap vol(f_j) \neq \emptyset\}$. Note that $\cap$ is not a regularized intersection.

One limitation with the MDG as it has been defined in Definition 1 is that it assumes an explicit ordering on the features or design operations. In many cases this may be captured in the solid modeling application in the form of a design history. But can the MDG be used when dealing with CSG trees? The answer is yes and can be obtained by extending the definition of the MDG to work recursively down the CSG tree.

Definition 2: *Model Dependency Graph* - non-linear definition

Let $T = (op\ left\ right)$ be a CSG tree or some non-linear design history where op is an operation and $left$ and $right$ are CSG subtrees or primitives shapes. Let $G_1 = (V_1, E_1)$ be the MDG of $left$ that results from either the basic definition or the non-linear

[1] This concept is similar to that of a "design history" which has been much addressed in the engineering design community.

definition. Let $G_2 = (V_2, E_2)$ be the MDG of $right$ that results from either the basic definition or the non-linear definition. Then the MDG of T can be defined as $G = (V, E)$ such that $V = V_1 \cup V_2$ and $E = E_1 \cup E_2 \cup E_3$ where $E_3 = \{(v_2, v_1), v_1 \in V_1, v_2 \in V_2$ such that $vol(v_1) \cap vol(v_2) \neq \emptyset\}$. Note that $\cap$ is not a regularized intersection.

An example of a solid model with different possible design feature histories is shown in Figure 4. There is a property of the MDG that we will exploit in our similarity assessment of solid models: *digraph D-morphism*. 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 [4] 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 $(f(u), v') \in A_2$ then there exists a $v \in f^{-1}(v')$ for which $(u, v) \in A_1$.

Theorem 1: *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 4). G_1 and G_2 are D-morphic.

Proof: Pick any two orderings of the set $F = \{f_0, \ldots, f_n\}$ arbitrarily. Let these orderings be $L = \{l_0, \ldots, l_n\}$ and $H = \{h_0, \ldots, h_n\}$ where $\forall f_i \in F$, $\exists l_j \in L, h_k \in H$ such that $f_i = l_j = h_k$ and $\exists i, 0 <= i <= n$ such that $l_i \neq h_i$. Let $G_1 = (V_1, E_1)$ be the MDG that results from L and let $G_2 = (V_2, E_2)$ be the MDG that results from H. It is clear that $V_1 = V_2$. By the definition of the MDG, these vertex sets must be equal to the set F. Now take any two vertices $v_k, v_l \in V_1$. Pick out the vertices $v_m, v_p \in V_2$ such that $v_k = v_m = f_i$ and $v_l = v_p = f_j$. Note that $vol(v_k) = vol(v_m)$ and $vol(v_l) = vol(v_p)$. Therefore, $vol(v_k) \cap vol(v_l) = vol(v_m) \cap vol(v_p)$. Hence, from the definition of the MDG, if there is an edge $(v_k, v_l) \in E_1$ where $k > l$ then either $(v_m, v_p) \in E_2$ where $m > p$ or $(v_p, v_m) \in E_2$ where $p > m$. Therefore, G_1 and G_2 are D-morphic.

Some questions may arise given the definition of MDG and the proof of D-morphism. One such question is how to generate the MDG of a given model. A possibility is to generate the MDG at design time. Upon the addition of a feature to the design, a node must be added to the MDG. Along with this new node, edges must be added from the newly added node to any previously added node corresponding to any features for which there is a non-empty intersection with the newly added feature.

Another question that will arise is how to handle the possibility of the same model being designed two different ways or with different feature sets. The same model designed with different feature sets will have MDGs that are not necessarily D-morphic. And related to this question is the question of what to do if a design history is not available for a given model. One solution is to use a feature extraction system such as F-Rex [14] or Allied Signal's FBMach [2] to extract the features. In this way you can use one common set of features across the entire collection of models. Performing this feature extraction will result in a unique set of features for the given model. This set of features will become the node set of the MDG. You can then order this set arbitrarily and generate the edge set of the MDG by making use of feature interactions detected during the feature extraction phase.

3.2.2 Phase II: Comparison and Retrieval

We compare the similarity of 2 solid models by testing for a D-morphism or for a subgraph D-morphism. The general problem of determining if there exists a D-morphism for a given pair of directed graphs is NP-complete [4]. However, there are two aspects of this problem domain that we can exploit to significantly reduce this complexity:

- First, it is not necessary to completely solve the D-morphism problem: Since we are only concerned with similarity, knowing if two MDG's are "almost" D-morphic is sufficient. Hence, we can use a heuristic method for the D-morphism test. Specifically, we will develop an algorithm that is a variant of gradient descent (or hill-climbing) that exploits the feature information we have in the design feature history.

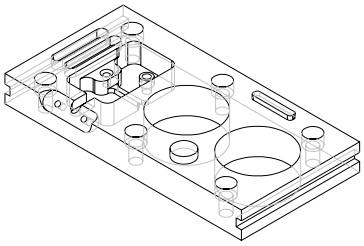
- Second, there is a great deal of domain knowledge present in the CAD model and in the feature history that can reduce the search space. For example, we will only consider mappings that compare similar feature types (i.e., holes map to holes, not to pockets). Additional constraints about vertex degree and size, location, and orientation can also be considered.

In testing for a D-morphism, we arbitrarily choose an initial set of pairings between the nodes of the two graphs (i.e., for each node of G_1 we choose at random a node of G_2 such that no two nodes of G_1 are "paired" with the same node of G_2). We then swap the pairings of the two nodes that reduce the value of our 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., we have reached a plateau), we 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. We are currently using $P = |V_1|^2$ (where V_1 is the vertex set in the smaller graph) but are experimenting with this value.

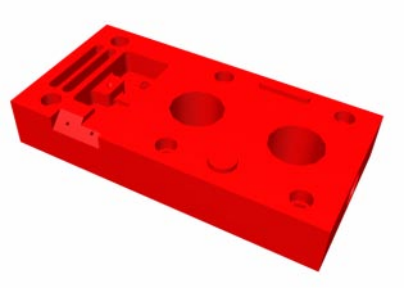
We use as our evaluation function the count of the number of mis-matched edges. That is, the evaluation function, $H = |E|$ such that $G_1 = (V_1, E_1)$ is the smaller of the two graphs being compared, $G_2 = (V_2, E_2)$ is the larger of the two graphs, and $E = \{(u, v) \in E_1$ such that $((paired(u), paired(v)) \notin E_2 \wedge (paired(v), paired(u)) \notin E_2)$
$\vee\ label(u) \neq label(paired(u)) \vee label(v) \neq label(paired(v)\}$. As a measure of similarity we employ the value $H^* = \frac{min\{H_1, \ldots, H_n\}}{|E_1|}$ where $H_1, \ldots, H_n$ are the values of H from up to n random restarts of the algorithm and E_1 is the edge set of the smaller graph. The function "paired(x)" above returns the node $y \in V_2$ that is currently paired with the node $x \in V_1$. The function "label(x)" used above returns the label of the node x.

Algorithm 1: D-Morphism Test
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 graphs are found to be D-morphic or if one is found to be subgraph D-morphic to the other. Otherwise, H is returned where H is the number of mis-matched edges when the algorithm halts.

D-MORPHISM(G_1, G_2, P)
(1) `Pairings` = GETRANDOMPAIRINGS(G_1, G_2)
(2) $i = 0$
(3) `BestResult` = $H(G_1, G_2$,`Pairings`)
(4) **while** (`BestResult` $> 0) \wedge (i < P)$
(5) **if** $H(G_1, G_2$,APPLYSWAP(`Pairings`,`BestSwap`)) $<$ `BestResult`
(6) `Pairings` = APPLYSWAP(`Pairings`,`BestSwap`)
(7) $i = 0$
(8) `BestResult` = $H(G_1, G_2,$ `Pairings`)
(9) **else**
(10) **if** $H(G_1, G_2,$ APPLYSWAP(`Pairings`,`BestSwap`)) = `BestResult`
(11) `Pairings` = APPLYSWAP(`Pairings`, `BestSwap`)
(12) $i = i + 1$
(13) **else**
(14) $i = P$
(15) **return** `BestResult`



(a) TEAM



(b) TEAM-2

Figure 5: Two of the test parts from the DOE TEAM Project. Both of these parts are available from the National Design Repository at `http://repos.mcs.drexel.edu`.

The node labels may contain as little or as much information as you choose. For our experiments, 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, you may restrict allowable pairings which will increase the algorithm's performance by reducing the search space. Incorporating more information in the node labels will also obtain a more meaningful similarity measure. For example, if some notion of dimension was incorporated into the labels then a really large block with a tiny hole will not be found similar to a little block with a larger hole.

Algorithm 1 is the algorithm we developed and described for the D-Morphism Test using gradient descent. In the algorithm, `Pairings` refers to the set of pairings between the nodes of the two graphs. And GETRANDOMPAIRINGS returns a random set of pairings as described above. H is the evaluation function that counts the number of mis-matched edges given two graphs and a set of pairings between the nodes in these two graphs. `BestSwap` is the swap from the set of all possible swaps between pairings that results in a set of pairings with the smallest value for H. APPLYSWAP returns the set of pairings that results from applying the given swap to the given set of pairings. To obtain a similarity measure, the smallest result of n executions of this algorithm is divided by the number of edges in the smaller of the graphs.

4 Experimental Results

We generated a family of solid 50 models using the ACIS 3D Toolkit running on 200MHz Pentium running Microsoft Windows NT 4.0. These models were pseudo-random variations on the US

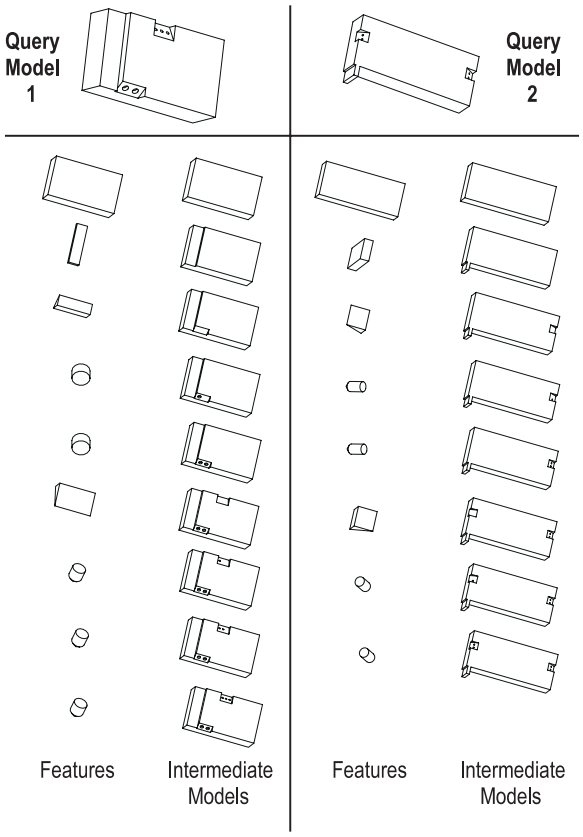


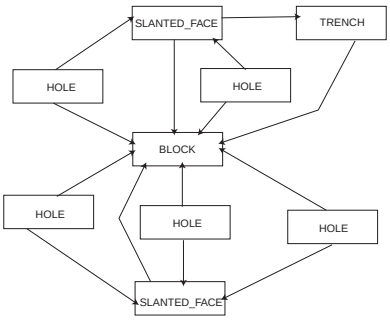
Figure 6: Two randomly generated query models with their design feature histories.

Department of Energy's Technologies Enabling Agile Manufacturing (TEAM) Project test parts pictured in Figure 5. These parts have a variety of standard feature types, such as pockets, slots, holes, counterbore holes, and bosses; in addition, many of the features interact and intersect, leading to a variety of different possible orderings for design feature histories and manufacturing process plans. The two parts pictured have several subtle differences that make them a useful target domain for experimentation.

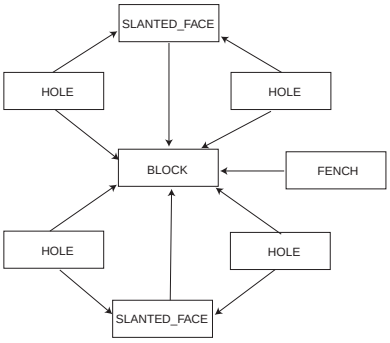
Our random "TEAM part" generator is based on the work of Alexei Elinson at the University of Maryland at College Park [3]. It operates by varying the number of features, the location features, and the number of different feature types over the part (depressions and protrusions, pockets, holes). For each of 50 models generated, we stored the design feature history of each model along with the intermediate δ_{f_i} and the $\delta^*_{f_i}$ models. Using this information, we calculated the MDG for each model.

Next, we selected two arbitrary Query Models from the set of 50 random models—these are shown in Figure 6. The figure shows the design histories of these parts; their MDG graphs are shown in Figures 7 (a) and 7 (b). Each of the query parts was compared to each part from the set randomly generated parts. To perform MDG comparison, a random restart gradient descent algorithm was used (as described in Section 3) with number of restarts fixed at 20. These matching tests searched for a subgraph of the larger of the query MDG and the given MDG from the set of 50 that was D-morphic to the smaller. The matching algorithms are implemented in C++ using the LEDA graph library. The tests were performed on a Sun UltraSPARC 30 workstation running Sun Solaris 2.6.

Figure 8 shows the results of these two queries. The histograms



(a) Query Model 1



(b) Query Model 2

Figure 7: The MDGs for the randomly generated Query Models.

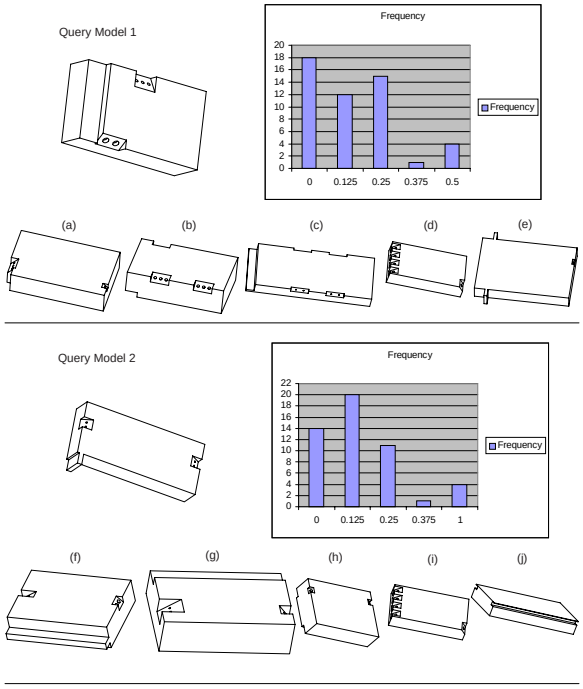


Figure 8: Example output data from examining D-morphism over the database of 50 solid models for the two query models in Figure 6. The histogram shows the number of models (from the 50 in the database) that fall into distance categories based on the D-morphism test. Read from left-to-right, the returned models are in order of decreasing similarity to the query model.

show that each query model partitioned the set of 50 random parts into distinct subsets, based on the result of the D-morphism test. For both query parts, there was a high percentage of parts found to be "similar." This is to be expected, since the set of parts consist of a family of parts generated at random from a limited set of operations based on the TEAM parts. For both queries, the query models were each was among the set of models D-morphic to the query.

Results for Query Model 1. For **Query Model 1**, 18 models were found such that they were subgraph D-morphic to the query model or that the query model was subgraph D-morphic to it. Among this set was the query model itself. Also among this set was model (a) in Figure 8. If you look at this model you will see that, like **Query Model 1**, it consists of two slanted faces, one with 2 holes and the other with 3 holes. Also common to both **Query Model 1** and (a) is a removal volume adjacent to one of the slanted faces. These two parts are very much alike. In fact, in this case, the parts were not only subgraph D-morphic, but were actually D-morphic.

Next, notice model (b). This model was among 12 models where the ratio of "mis-matched" edges to total edges at the completion of the matching test was greater than 0 but less than or equal to 0.125. The actual value of this particular case was 0.07. Aside from the interaction between one of the slanted faces and the removal volume in **Query Model 1**, the MDG for the query model would be D-morphic to that of model (b).

Models (c), (d), and (e) were in the next three groups shown on the histogram for query 1 respectively. Model (c) has an additive feature on one of its side faces while the query model had no such feature. Model (d) has 5 slanted faces and holes in each and lacks the slot that the **Query Model 1** has. Model (e) has two additive features on two of its side faces while the **Query Model 1** has no such additive features.

Results for Query Model 2. For **Query Model 2**, 14 models were found such that they were subgraph D-morphic to the query model or that the query model was subgraph D-morphic to it. Among this set was the query model itself. Also among this set was model (f) in Figure 8. If you examine these two models, you will see that each has an additive feature on one side face and each has two slanted faces with holes in each. They are very much alike.

Model (g) is one of 20 models with a ratio of mis-matched edges to total edges greater than 0 and less than or equal to 0.125. This ratio for model (g) was actually 0.09. The difference between these two models is that (g) has a subtractive feature while **Query Model 2** has an additive feature on on of its side faces.

Models (h), (i), and (j) are in the next three groups on the histogram. Model (h) has two subtractive features not in the query model and the query model has the additive feature on one of the side faces. Model (i) is the same model as (d). This model was about the same in dissimilarity as both query models. Every edge in the MDG for model (j) was mis-matched when compared to that of **Query Model 2**. Model (j) has a trench in two of its side faces and no other features.

Statistics. All 50 models used in this experiment along with their design history are available as ACIS `.sat` files at `http://repos.mcs.drexel.edu/SM99-DATA`.

To compare **Query Model 1** against all 50 models took a total of 243.77 seconds of CPU time on the Sun UltraSPARC 30. For the 18 comparisons that resulted in a D-morphic match, the median number of operators required to find the match was 4. The highest was 1364 moves. In that case, a number of restarts were necessary before the match was found. This case was actually the comparison of the query model to itself.

To compare **Query Model 2** to all 50 models took a total of 187.37 seconds of CPU time on the Sun UltraSPARC 30. For the 14 comparisons that resulted in a D-morphic match, the median number of moves made to find the match was again 4. However, the highest was only 6 moves. For the second query, no restarts were necessary for any of the comparisons.

5 Discussion and Conclusions

This paper has presented an approach to the problem of handling variation and non-uniqueness of design feature histories information. In particular, we introduced a data structure called a *Model Dependency Graph* and show how it can be used to manage knowledge-bases of CAD and Solid Modeling data.

Research Contributions.

Data Structures to Resolve Ambiguity: The MDG can be used to resolve certain classes of ambiguity in representation of the design feature histories of CAD and solid models. The MDG can then be used as a basis for interesting comparison among solid models.

Index and Query Scheme: The MDG is a useful mechanism for archival and retrieval of models in CAD databases. Using algorithms for detecting graph D-morphism, and introducing some engineering domain knowledge, we have created a general technique for archiving large numbers of solid models and retrieving them based on the similarity of their design features. We believe that this technique can be refined and will have impact on how CAD data is stored and managed.

Detection of Part Families: Based on the MDG, one can create query artifacts that partition the database of solid models into different D-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.

Future Directions.

Exploiting Geometry and Topology: Information about the feature locations, dimensions, and orientations have to be exploited fully. By leveraging some earlier work [3], we hope to incorporate additional feature attribute information into our indexing and comparison algorithms.

Use of Engineering Information: Currently, we are considering plain solid models. In future, we believe that additional domain knowledge can be used to refine our techniques. Information about engineering tolerances, surface finishes, constraints and parametrics, etc. all can be used to augment the basic techniques presented here.

Application to CSG trees: We believe that these techniques might have applicability to the well-known problem of the non-uniqueness of CSG trees. We conjecture that the MDGs for different CSG-based descriptions of the same solid model are D-morphic under different permutations of the same volumetric primitives and boolean operations.

Use of features extracted through feature recognition: If operating inside a CAD environment, one could plan to retain the design feature histories as new models are created. However, for legacy data and for solid models that are converted between modeling systems, there may not be any ready feature information. One avenue of future study will be to use automatic feature recognition techniques (such as FBMach from Allied Signal Inc. [2, 6]) to generate feature data to be used in the indexing algorithms.

Larger-Scale Experiments: Using the National Design Repository, along with feature information generated automatically via feature recognition (as noted above), we plan to perform larger-scale experiments involving a database containing thousands of solid models.

The authors anticipate having additional results over the coming months—in particular in the area of larger-scale experiments and extracted features. It is our hope that this research expands the understanding of this new problem domain in CAD and Solid Modeling and lays the foundation for exploring new techniques to enhance our ability to search and retrieve 3D CAD and solid model data.

Acknowledgements. Our thanks are extended to Dr. Steve Brooks of Allied Signal Corporation, Federal Manufacturing Technologies Program, in Kansas City for providing the National Design Repository with the ACIS models for the TEAM parts. Thanks also to Alexei Elinson for providing us with the code to generate random TEAM-like solid models.

This work was supported in part by National Science Foundation (NSF) CAREER Award CISE/IIS-9733545 and Grant ENG/DMI-9713718. Additional support was provided by the National Institute of Standards and Technology (NIST) under Grant 60NANB7D0092.

Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation or the other supporting government and corporate organizations.

References

[1] Willem F. Bronsvoort and Frederik W. Jansen. Feature modelling and conversion - Key concepts to concurrent engineering. *Computers in Industry*, 21:61–86, 1993.

[2] Steven L. Brooks and R. Bryan Greenway Jr. Using STEP to integrate design features with manufacturing features. In A. A. Busnaina, editor, *ASME Computers in Engineering Conference*, pages 579–586, New York, NY 10017, September 17-20, Boston, MA 1995. ASME.

[3] Alexei Elinson, Dana S. Nau, and William C. Regli. Feature-based similarity assessment of solid models. In Christoph Hoffman and Wim Bronsvoort, editors, *Fourth Symposium on Solid Modeling and Applications*, pages 297–310, New York, NY, USA, May 14-16 1995. ACM, ACM Press. Atlanta, GA.

[4] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, New York, 1979.

[5] Reinhold Geelink, Otto W. Salomons, Fjodor van Slooten, and Fred J. A. M. van Houten. Unified feature definition for feature-based design and feature-based modeling. In A. A. Busnaina, editor, *ASME Computers in Engineering Conference*, pages 517–534, New York, NY 10017, September 17-20, Boston, MA 1995. ASME.

[6] JungHun Han, William C. Regli, and Steve Brooks. Hint-based feature recognition. In *ASME Computers in Engineering Conference*, New York, New York, September 14-17, Sacremento, CA. 1997. ASME.

[7] JungHyun Han and Aristides A. G. Requicha. Integration of feature-based design and feature recognition. In A. A. Busnaina, editor, *ASME Computers in Engineering Conference*, pages 569–578, New York, NY 10017, September 17-20, Boston, MA 1995. ASME.

[8] Christopher M. Hoffmann. *Geometric and Solid Modeling: An Introduction*. Morgan Kaufmann Publishers, Inc., California, USA, 1989.

[9] Qiang Ji and Michael M. Marefat. Machine interpretation of cad data for manufacturing applications. *Computing Surveys*, 29(3):264–311, September 1997.

[10] S. Joshi and T. C. Chang. Graph-based heuristics for recognition of machined features from a 3D solid model. *Computer-Aided Design*, 20(2):58–66, March 1988.

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

[12] Michael Marefat and Rangasami L. Kashyap. Automatic construction of process plans from solid model representations. *IEEE Transactions on Systems, Man, and Cybernetics*, 22(5):1097–1115, September/October 1992.

[13] T. De Martino, B. Falcidieno, F. Giannini, S. Hassinger, and J. Ovtcharova. Feature-based modelling by integrating design and recognition approaches. *Computer Aided Design*, 26(8):3–13, August 1993.

[14] William C. Regli, Satyandra K. Gupta, and Dana S. Nau. Extracting alternative machining features: An algorithmic approach. *Research in Engineering Design*, 7(3):173–192, 1995.

[15] Otto W. Salomons, Fred J. A. M. van Houten, and H. J. J. Kals. Review of research in feature-based design. *Journal of Manufacturing Systems*, 12(2):113–132, 1993.

[16] Vadim Shapiro and Donald L. Vossler. Construction and optimization of csg representations. *International Journal of Computer Aided Design*, 23(1):1–20, January/February 1991.

[17] Vadim Shapiro and Donald L. Vossler. Separation for boundary to csg conversion. *ACM Transactions on Graphics*, 12(1):35–55, January 1993.

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